Vorwort zur 2. Ausgabe

Dies ist sie nun, die erste überarbeitete Ausgabe des Windowsskript. Anstelle einer Reihe von Beispielen ist nun ein Beispielprojekt getreten.
das Projekt mag an einigen Stellen nicht ganz so viele Aspekte und Konzepte abdecken wie die Vielzahl an Beispielen, über die sonst referiert wurde, die Möglichkeit ein Programm in seiner Entsehung zu verfolgen erscheint jedoch nicht minder lehrreich.
An diesem Skript haben, viele Menschen mitgewirkt – Freunde, Bekannte, Kollegen und Kursteilnehmer. Alle haben mich eifrig mit Verbesserungsvorschlägen – und nach mühseliger Detektivarbeit auch mit Fehlerkorrekturen versorgt. Ihnen allen gilt mein Dank für die wertvolle Unterstützung.
Ein ganz besonderer Dank gebührt meiner Frau, die es seit Jahren erträgt, daß ich jeden Urlaub und viele Wochenenden dazu nutze, Skripte zu überarbeiten und zu erweitern, so daß es von seiner „Erstausgabe“ mit knapp 10 Seiten auf den heutigen Umfang wachsen konnte.

1 Von DOS zu Windows

Wechselt man von DOS zu Windows, so besteht der Hauptunterschied in der graphischen Oberfläche von Windows (zumeist mindestens 640x480 Bildpunkte, mit 16 Farben), gegenüber der zeichenorientierten Oberfläche von DOS (meist 80x25 Zeichen, monochrom).
Neben diesem offensichtlichen Unterschied gibt es jedoch noch eine Vielzahl von internen Verarbeitungsunterschieden, die dazu führen, dass Windows Programme ganz anders aufgebaut werden müssen als Programme für DOS.
Einer der Faktoren mit dem stärksten Einfluss ist das Multitasking[1], d.h. die Fähigkeit von Windows, mehrere Programme gleichzeitig im Speicher zu halten und mit Eingaben zu bedienen. abhängig von der eingesetzten Windows-Version gibt es aber auch beim Multitasking selbst erhebliche Unterschiede.

1.1 nonpreemptive Multitasking

Das nonpreemptive[2] Multitasking (bis einschließlich Windows 3.xx) sorgte dafür, dass mehrere Programme gleichzeitig geladen sein konnten. Allerdings konnte immer nur ein Prozess zur Zeit den Zentralprozessor (die CPU) belegen. Dies führte dazu, dass man gelegentlich z.B. einen etwas langwierigeren Berechnungsprozess startete und dann geneigt war, auf eine andere Anwendung zu wechseln, um sich während der Wartezeit mit anderen Dingen zu beschäftigen.
Dies führt dann aber dazu, dass der gerade gestartete Berechnungsprozess zugunsten der jetzt aktivierten Anwendung unterbrochen wird, um möglichst schnell auf Eingaben reagieren zu können. Erst wenn die Berechnungsanwendung wieder in den Vordergrund geholt wird (also aktiviert wird), kann der Berechnungsprozess fortgesetzt werden. Einzige Ausnahme sind einige Systemprozesse (wie Hardwareansteuerung, z.B. Druck) und Programme, die sogenannte Timer oder Interrupts verwenden.

1.2 preemptive Multitasking

Das in Windows NT und Windows 95 vorhandene preemptive Multitasking sorgt über ein Zeitscheiben-Verfahren dafür, dass alle Programme immer wieder ein wenig Rechenzeit bekommen und somit gleichmäßig, (fast) parallel ausgeführt werden. D.h. alle laufenden Prozesse teilen die zur Verfügung stehende Prozessorzeit unter sich auf. Dies führt natürlich dazu, dass alle Programme immer langsamer werden, je mehr Prozesse zur selben Zeit laufen. Die relative Größe der Zeitscheiben zueinander kann über Prioritäten geregelt werden, Programme mit höherer Priorität erhalten dann größere Zeitscheiben.
Die Entscheidung Windows zunächst mit einem nonpreemptive Multitasking auszustatten ist sicherlich (neben der erheblich höheren Komplexität eines „echten“ Multitasking) in der nicht allzu hohen Prozessorleistung der ersten Windows-Systeme zu suchen. Es machte

[1] Task [engl.]: Aufgabe, in der EDV die Ausführung eines Programms
[2] Preemptive [engl.]: präventiv, zuvorkommend, bevorzugt

schlicht keinen Sinn, auf einem „langsamen“ (25 MHz) i286er Prozessor zu versuchen mehrere Programme „echt“ parallel laufen zu lassen, da jedes Programm für sich so langsam wurde, dass ein vernünftiges Arbeiten nicht mehr möglich war. Mit den rasant gestiegenen Leistungsdaten neuerer Prozessoren (ab i486er) allein war es jedoch auch noch nicht getan, denn mit der Prozessorleistung steigen auch die allgemeinen Anforderungen an die CPU - so müssen auch höhere Farbtiefen und bessere Bildschirm-Auflösungen zumindest teilweise von der CPU aufgefangen werden. Zwar übernehmen die Graphikkarten die Darstellung, die Berechnung der dargestellten Information obliegt aber meist weiterhin der CPU (und typischerweise wird eine größere Bildschirmauflösung ja dazu benutzt mehr Informationen darzustellen).

1.3 Eine andere Art zu Programmieren...

Ein weiterer Aspekt, der sich aus der Tatsache ergibt, dass mehrere Programme gleichzeitig laufen (spätestens unter Windows 95), ist das Problem dass ein Prozess den Prozessor nicht blockieren darf. D.h. jedes Programm hat dafür zu sorgen, dass die CPU sich um andere Dinge kümmern kann, wenn für es selbst keine Tätigkeiten (meist eingaben des Benutzers) anliegen. Typische Vorgehensweisen in Programmen, die man sich in der DOS-Welt angewöhnt hat, sind daher unter Windows verboten. So hat man Benutzereingaben unter DOS typischerweise wie folgt realisiert:

```
#include <conio.h>
#include <iostream.h>

#define RETURN      13
#define STRINGENDE   0
#define LENGTH     200

void main (void)
{
   int  i                       = 0;
   char cZeichen                = STRINGENDE;
   char sZeichenkette [LENGTH] = "";

   while ((!cZeichen == RETURN) && (i < LENGTH-1))
   {
      cZeichen = getch();     // BORLAND-BEFEHL! Kein Standard
      sZeichenkette[i++] = cZeichen;
      sZeichenkette[i]   = STRINGENDE;
   }
   cout << sZeichenkette;
}
```

Eine solche und andere Endlos-Schleifen (Polling[3]) sind unter Windows höchst problematisch, da sie Prozessor bis zum Eintreten eines bestimmten Ereignisses (hier bis zur Eingabe der Taste RETURN) belegt halten. Resultat ist also ein „hängen bleiben“ des gesamten Windows-Systems (jedenfalls unter Windows 3.xx).

[3] to poll [engl.]: befragen, in der EDV wird das wiederholte Abfragen einer Schnittstelle (hier der Tastatur) in einer Schleife, als Polling bezeichnet.

Statt dessen sind solche Schleifen fast ausschließlich dem Windows-Kernel (der Windows-Verwaltung) erlaubt, mit Ausnahme der Message-Loop[4], die in jedem Windowsprogramm vorhanden sein muss (s.u.). Diese wird benötigt, damit das Anwendungsprogramm darüber unterrichtet werden kann, dass eine zu verarbeitende Eingabe vorliegt – oder besser ein zu verarbeitendes Ereignis, denn es gibt neben Texteingaben des Benutzers noch jede Menge anderer Messages.

1.4 Ungarische Notation

In der Windowsprogrammierung hat sich die sogenannte „Ungarische Notation“ durchgesetzt, bei der die Datentypen als Teil des Variablen-, Funktions- und Parameternamens kodiert werden:

Kürzel	***Bedeutung***	***C/C++-Typen***
a	Array, Feld, Struktur, Record (alle zusammensetzten Datentypen)	
b	Boolean Wert (Wahrheitswert)	int
c	Character, Zeichen	char
dw	double word	unsigned long
f	Fließkommawerte	float, double, long double
fn	Funktion, Unterprogramm	function
h	Handle	
l	Long Wert	long
n	Numerischer Wert	short, int, long
p	Pointer	
r	Referenz	
s	String, Zeichenkette	char [] / char *
w	Word (Speicherwort ohne Vorzeichen)	unsigned int
Von Microsoft verwendet:		
cb	Byte-Anzahl (Count of Bytes)	
fw	gesetztes Bit im Bitfeld (Flag Word)	unsigned int
hbr	Handle auf einen Brush	
lpsz	Long pointer (32 bit) auf einen String	char *
sz	String (Null-begrenzte Zeichenkette)	char [] / char *
tag	Typvereinbarung	struct
x, y / cx, cy	Koordinaten / Längen, Count	short, int
Häufig alternativ oder zusätzlich verwendet:		
by	Byte (Speicherzelle ohne Vorzeichen)	unsigned char
i	Integer Wert	int
lf	Double Wert	double
r	Record, Datensatz	struct

Für die ungarische Notation gibt es keine verbindlichen Standards, vielmehr einigen sich die an einem Projekt Beteiligten auf eine Notation oder sie wird von der Firma oder dem Auftraggeben vorgeschrieben.
Zu empfehlen ist jedoch ein Mindestmaß, welches in der Tabelle oben im ersten Teil aufgeführt sind. Die zusätzlich oder alternativ

[4] Message-Loop [engl.]: Nachrichten-Schleife

verwendeten Kürzel haben den Vor- und Nachteil detaillierter zu sein. Der Vorteil kommt dann zum tragen, wenn man detaillierte Information braucht (z.B. wissen muss, ob ein Wert vom Typ **int** oder **long** ist). Der Nachteil ist natürlich, dass man den Namen der Variablen im gesamten Programm ändern muss, wenn man den Variablentyp in der Größe anpasst, also z.B. von **int** auf **long** ändert. Dies kommt, anders als eine Typänderung von Ganzzahl auf Fließkommazahl, relativ häufig vor.

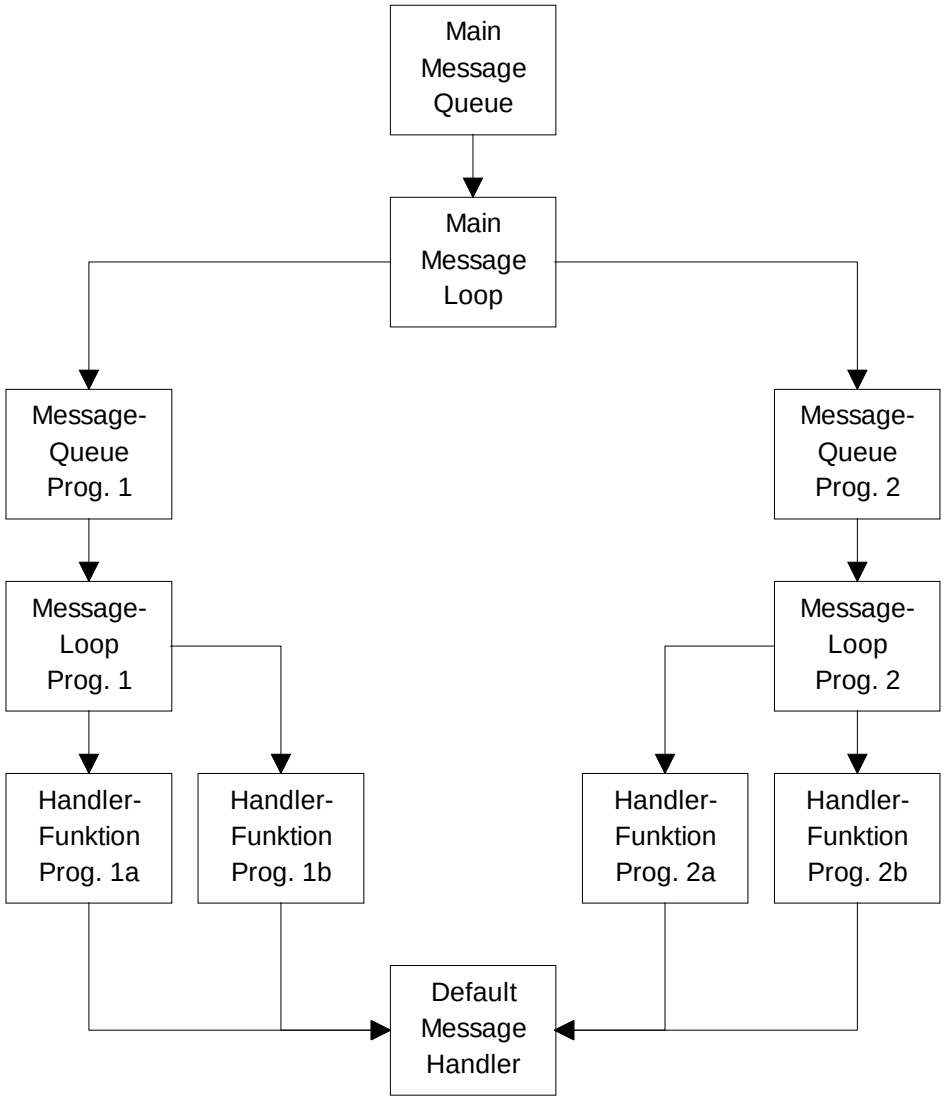
2 Messagebehandlung

Die folgenden Abschnitte zeigen, wie der „universelle“ Nachrichtenmechanismus von Windows funktioniert.

2.1 Message-Loop und Message-Queue

Alle Nachrichten von Windows an ein Anwendungsprogramm werden über sogenannte Message-Loops (Nachrichtenschleifen) abgearbeitet. Message-Loops sind die einzigen „Endlosschleifen“, die ein Windows-Programm realisieren darf.
Sie bestehen im Grunde genommen nur aus einer Anfrage bei der zugeordneten Message-Queue[1] (Nachrichten-Warteschlange), ob eine Nachricht für das Programm vorliegt, oder nicht. Ist dieses der Fall, so wird die Nachricht aus der Message-Queue entnommen und verarbeitet.

[1] Queue [engl.]: Warteschlange



Alle entstehenden Nachrichten werden zunächst an die Main-Message-Loop (Haupt-Nachrichten-Schleife) gesendet, die diese dann anhand der zugeordneten Fensterhandles[2] (s.u.) auf die – den jeweiligen Programmen zugeordneten – Message-Queues verteilt. Dort werden die Messages von den Message-Loops der Applikationen[3] der Reihenfolge nach abgeholt und an die zuständigen Handlerfunktionen weitergereicht. Von diesen Handlerfunktionen kann eine Applikation mehrere besitzen, meist ist es eine Funktion pro Dialog[4].
Bis zu diesem Augenblick sind die Messages noch ungefiltert, d.h. in der Message-Loop der Applikation landen alle Messages, egal ob sie für die Anwendung von Interesse sind oder nicht. Erst die Handlerfunktion entscheidet, was mit einer Nachricht geschieht. Ist die Message von Interesse (z.B. ob ein Button gedrückt wurde), so wird in der Funktion entschieden, was zu tun ist. Ist die Nachricht hingegen

---

[2] to handle [engl.]: (mit etwas) umgehen, in Windows eine Kennziffer zum Umgang mit etwas, was vom Windows-Kernel verwaltet aber vom Programm benutzt wird.
[3] Application [engl.]: Anwendung, in der EDV ein Programmpaket
[4] in Windows werden (der Client-Area) untergeordnete Fenster meist als Dialoge bezeichnet.

uninteressant, so wird sie an den Default-Message-Handler[5] weitergeleitet. Um diese Form der „Bürokratie“ im System zu realisieren, gibt es im Windows-Kernel einen globalen Windowshandler, der sich um alle durch Benutzeraktionen entstandenen Botschaften kümmert (z.B. Mausbewegungen), die man nicht selbst verarbeiten will. Die meisten dieser an den globalen Windowshandler weitergereichten Nachrichten werden von Windows schlicht als „erledigt“ gekennzeichnet und gelöscht, andere in einer Standardform ausgewertet. Dieses Verfahren ist sinnvoll, da dadurch der Anwender z.B. nicht selbst dafür sorgen muss, dass bei Aktivierung eines Menüs die Menüeinträge aufgeblendet werden. Statt dessen ist dieses voreingestellte Verhalten ein Teil der Funktionalität des Default-Message-Handlers.
Windows verlangt außerdem, dass jede Botschaft, die an ein Programm geschickt wurde quittiert wird, da sie erst anschließend im System vernichtet wird (dieses geschieht über einen entsprechenden Returncode (Rückgabewert) nach der Message-Verarbeitung).

2.2 Aufbau von Messages

Wie bereits beschrieben, erzeugt jede Aktion in Windows sogenannte Messages, die über die Main-Message-Loop an alle angemeldeten Programme weitergereicht werden, für die diese Nachricht von Interesse ist. Wird z.B. die Maus bewegt, so erhält jedes Fenster eine Flut von MOUSEMOVE Messages. Man erhält z.B. auch Nachrichten, wenn sich die Maus über einem Editfeld befindet, wenn die Maustasten gedrückt oder losgelassen wurde usw. Die Art der Messages ist vielfältig und das eigene Programm wird geradezu von solchen Nachrichten überflutet, da ja erst der Handler entscheidet was von Interesse ist. Der Aufbau einer Message ist relativ einfach, sie besteht aus einer Reihe von Werten (Parametern):

```
typedef struct tagMSG
   {HWND   hwnd;    UINT  message; WPARAM wParam;
    LPARAM lParam; DWORD time;    POINT  pt;  } MSG;
```

Die Bedeutung der Parameter:	
hwnd	Handle des Fensters, dessen Handler die Message empfängt
message	Message-Nummer
wParam	Zusätzliche Message-Information. Die Bedeutung hängt von der Message-Nummer ab
lParam	Zusätzliche Message-Information. Die Bedeutung hängt von der Message-Nummer ab
time	(interner Parameter) Zeitpunkt, an dem die Message erzeugt wurde
pt	(interner Parameter) Screenkoordinaten, an denen sich der Cursor befand, als die Message generiert wurde

[5] default [engl.]: Voreinstellung

2.3 Handlerfunktionen

Die Bearbeitung von Messages (Nachrichten) erfolgt, wie oben erwähnt, immer in sogenannten Messagehandlern, die jeweils einem bestimmten Fenster zugeordnet sind. Als „Fenster“ wird dabei nahezu jedes Windowselement gezählt - unter anderem auch solche, die man gar nicht dafür halten würde. So sind z.B. Listboxen und Edit-Felder eigentlich eigenständige Fenster, bekommen aber üblicherweise die Handlerfunktion des übergeordneten Fensters zugewiesen.

```
//-------------------------------------------------------------
// (Fast) minimale Handler-Funktion
//-------------------------------------------------------------
LRESULT CALLBACK WndProc (HWND hWnd, UINT nMsg,
                          UINT wParam, LONG lParam)
{
   HDC         hDC ;
   PAINTSTRUCT aPS ;

   switch (message)
   {
      case WM_PAINT :
           hDC = BeginPaint (hWnd, &aPS);
           EndPaint (hWnd, &aPS);
           return 0;

      case WM_DESTROY :
           PostQuitMessage (0);
           return 0;
   }
   return DefWindowProc (hWnd, nMsg, wParam, lParam);
}
```

Wenn man möchte kann man aber für jedes Fenster eigene Handler schreiben – zum Glück ist ein solches Vorgehen nur sehr selten notwendig.
Der wichtigste Handler ist jener, welcher der Client-Area[6] zugeordnet wurde. Dieser stellt sozusagen die zentrale Schaltstelle eines jeden Programms (Applikation) dar. Botschaften, die an andere Handler weitergereicht wurden und dort nicht verarbeitet werden konnten, können an einen hierarchisch übergeordneten Handler weitergereicht werden, bis sie schließlich beim Windows-Defaulthandler landen müssen. Dies ist notwendig, um z.B. Sondertasten (F1 als Hilfe-Taste) programmweit auswerten zu können.

2.4 Messagecodes und Stylecodes

Die folgenden Anfangsbuchstaben von #defines listen die am häufigsten verwendeten Messagecodes und (Parameter) auf, die in Windowsprogrammen zu finden sind.

#define-Beginn	Bedeutung
WM_	Windows-Messages

[6] Client-Area [engl.]: Kunden-Bereich, bezeichnet in Windows das Hauptfenster eines Programms

#define-Beginn	Bedeutung
BM_	Button-Messages (Checkboxen)
BN_	Button-Notify-Messages (Aktion des Benutzers)
BS_	Button-Style
CB_	Combobox-Messages
CBN_	Combobox-Notification-Messages
CBS_	Combobox-Stil
CF_	Zwischenablage-Messages
CS_	Windows-Klassen-Stil
CW_	Create-Windows-Stil
DT_	Drawtext-Stil (Art der Textausgabe)
EM_	Editfeld-Messages
ES_	Editfeld-Stil
FW_	Fontgewicht (Strichstärke)
HS_	Hatch-Stil (Schraffurstil für Brushes)
IDC_	Cursor-Identifikationsnummer (interne Standardcursor)
IDI_	Icon-Identifikationsnummer (interne Standardicons)
LB_	Listbox-Messages
LBN_	Listbox-Notification-Messages
LBS_	Listbox-Stil
MB_	Messagebox-Stil
MF_	Menü-Stil
MK_	Mouse-Messages
OF_	Datei-Messages
OUT_	Outline-Font-Stile
PS_	Zeichenstift-Stil (Penstyle)
R2_	Zeichenmodus (z.B. Binärverknüpfungen)
RGN_	Regionsverknüpgung (Binärverknüpfungen von Flächen)
SB_	Scrollbar-Messages
SC_	System-Command-Messages
SM_	System-Metric-Messages
SS_	Static-Text-Stil
SW_	Show-Windows-Stil
TA_	Textout-Stil (Text-Alignment)
VK_	Virtual-Key-Messages
WS_	Windows-Stil

2.5 Messagehandling

Unter Messagehandling versteht man die Auswertung der von Windows gesendeten Messages im Messagehandler. Grundsätzlich werden nur die Botschaften ausgewertet, die für das Programm von Belang sind (schließlich gibt es mehrere Hundert verschiedene Messagetypen).
Die Windows-Programmierung unterscheidet sich in einigen Punkten jedoch entscheidend von der „normalen“ Programmierung unter DOS. Da es sich bei Windows um ein Multitasking Betriebsystem handelt (für Windows 3.1 und Vorfahren trifft dies nur bedingt zu), verbieten sich Warteschleifen (z.B. beim Einlesen von Werten oder Benutzereingaben) von selbst, denn dies würde den Prozessor exklusiv

für unser eigenes Programm benötigen. Ein solches Vorgehen ist in einem Multitasking-System, in dem sich mehrere Programme die „knappen“ Systemresourcen[7] (wie CPU-Zeit) teilen müssen, natürlich nicht angebracht.
Man wartet statt dessen, bis man von Windows eine Botschaft bekommt, die besagt, dass der Anwender etwas eingegeben hat (oder irgend etwas anderes geschehen ist, auf das man reagieren möchte). Windows sorgt automatisch dafür, dass fortwährend alle möglichen Messages erzeugt werden.
Umgekehrt gilt natürlich auch, dass man ein Editfeld nicht direkt bearbeiten kann (z.B. auslesen), denn dessen wirkliche Adresse im Speicher ist unbekannt. Statt dessen sendet man z.B. eine Message, die das Editfeld dazu auffordert, den enthaltenen Text auf eine (mitgelieferte) Variable zu kopieren.
Wie man jetzt sieht, besteht die Windowsprogrammierung hauptsächlich darin Messages zu empfangen und zu versenden.

2.6 SendMessage und PostMessage

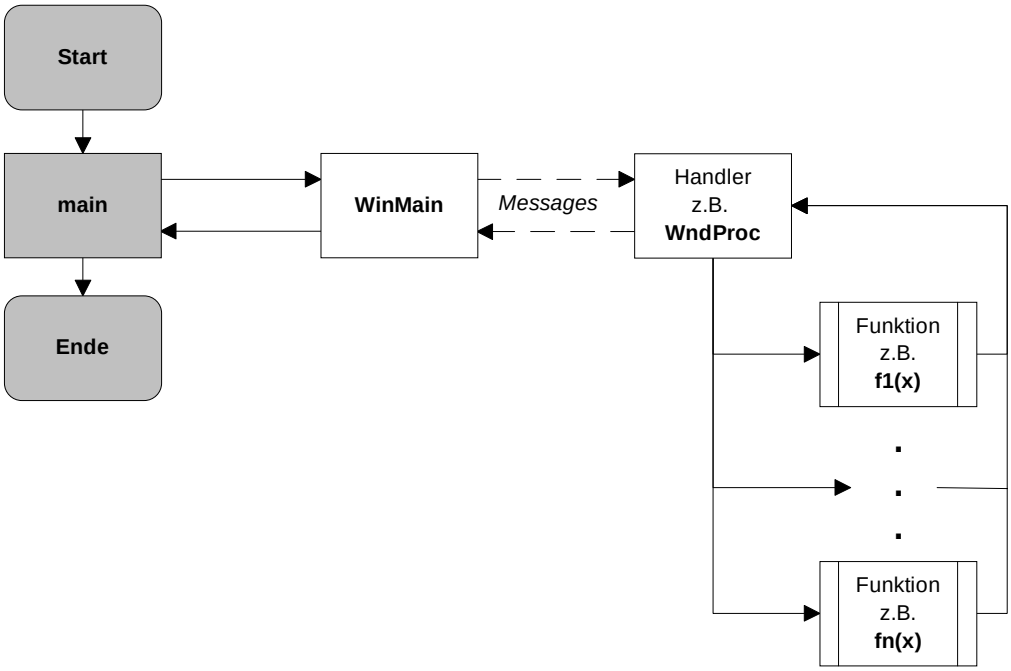
Die Messagebefehle sind das wichtigste Werkzeug bei der Programmierung eines Windows-Programms. Sie ermöglichen es dem Anwendungsentwickler mit jedem einzelnen der vielen Fenster (dies beinhaltet die Dialog-Controlelemente[8]) zu kommunizieren. Die SendMessage-Funktion sendet eine Befehlsnachricht unmittelbar (unter Umgehung der Message-Queue) an ein Window. Abfragen auf Inhalte (z.B. das Auslesen eines Editfeldes) erfolgt stets über SendMessage-Befehl, da man die gewünschte Information benötigt um z.B. den Programmverlauf zu steuern.
SendDlgItemMessage (Send Dialog Item Message) ist eine Variante der SendMessage-Funktion, die es dem Programmierer erspart, jedes Mal das Handle (s.u.) eines Controls per Hand ermitteln zu müssen.
PostMessage[9] hingegen stellt die Message in die Message-Queue. Der Befehl zum Beenden eines (WM_QUIT) Windowsprogramms wird immer „geposted“ - also wie ein vom Anwender erzeugtes Ereignis normal in die Windows-Nachrichtenschlange eingestellt. Dies ist nötig, damit bereits ausgelöste Hintergrundprozesse korrekt beendet werden können.

```
SendDlgItemMessage (hWindowhandle, nControlID, BM_SETCHECK,
                    TRUE, 0);
SendMessage (hControlhandle, BM_SETCHECK, TRUE, 0);
PostMessage (hControlhandle, WM_QUIT, 0, 0);
```

[7] resource [engl.]: Mittel, Bodenschatz – in Windows werden „knappe Güter“, die sich mehrere Programme unter Umständen teilen müssen (wie z.B. Prozessorzeit oder Graphikspeicher) als Resourcen bezeichnet.
[8] in Windows werden Elemente die der Benutzer bedienen kann (z.B. Knöpfe, Menüs) verallgemeinert als „Controls“ bezeichnet.
[9] to post [engl.] anschlagen, aushängen, veröffentlichen

3 Standardvorgehen für Windowsprogramme

Der erste Unterschied der einem C/C++ Programmierer beim Betrachten eines Windows-Programms auffällt, ist das Fehlen der Hauptfunktion ***main***. Sie wird in Windows ersetzt durch die Funktion ***WinMain***.
Da jeder C/C++ Compiler die Funktion ***main*** zwingend benötigt, ist diese natürlich auch in Windows-Programmen enthalten, sie wurde lediglich dem Zugriff das Anwendungsentwicklers entzogen und sorgt nun für eine ordnungsgemäße Initialisierung des Windows-Systems beim Programmstart.



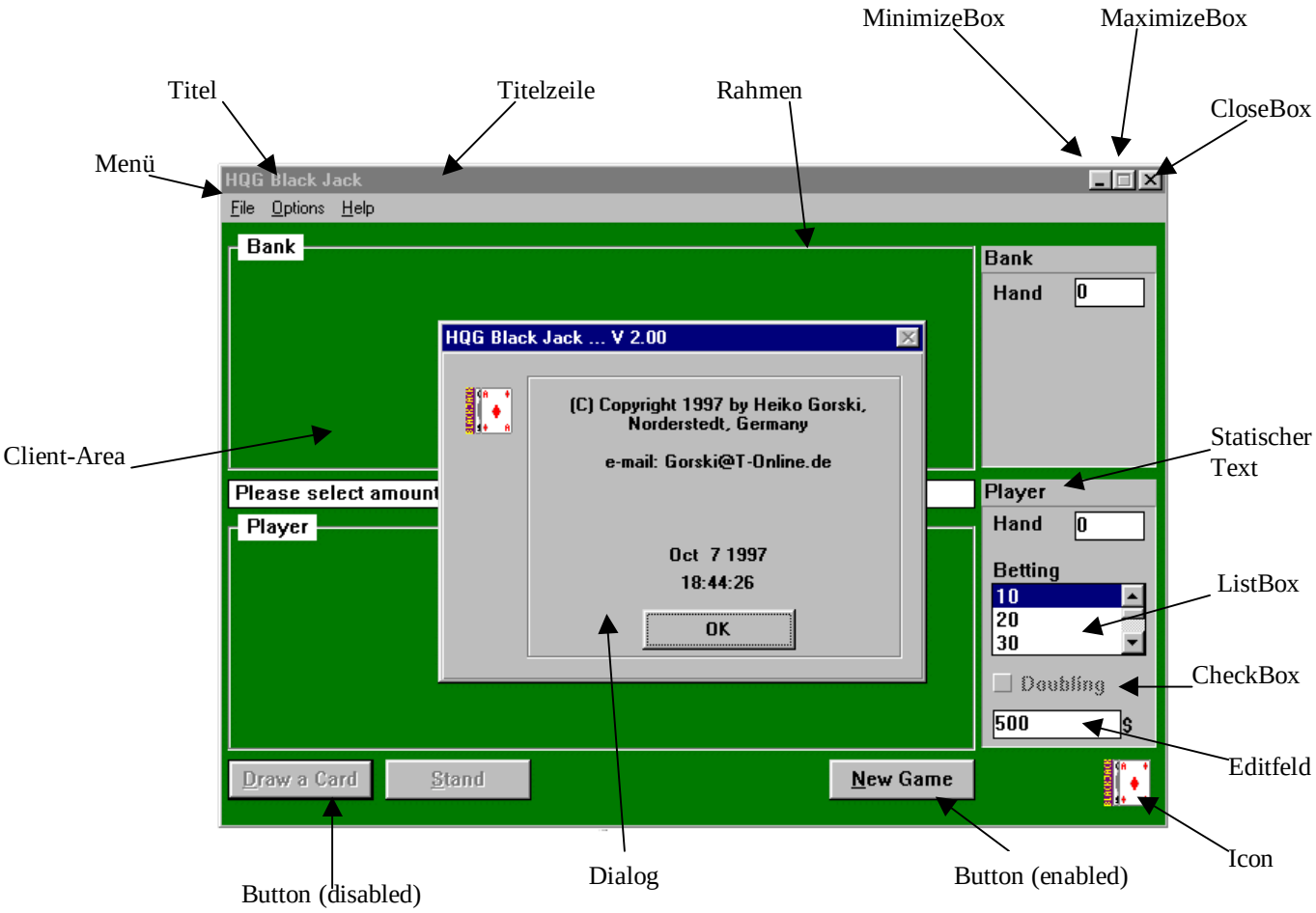
So wird in ***main*** u.a. festgestellt, ob Windows überhaupt verfügbar ist (wer einmal versucht hat ein Windows-Programm unter DOS zu starten, kennt die Fehlermeldung „Dieses Programm benötigt Microsoft Windows“) und die benötigten Windows-Kernel-DLLs werden geöffnet.

Um dieses zu erreichen, wird nach dem Compile-Vorgang ein sogenannter Startup-Code zum Programm hinzugelinkt. Dieser Startup-Code enthält letztlich nur eine bereits vorcompilierte ***main***-Funktion, die an gegebener Stelle die Anwenderfunktion ***WinMain*** aufruft. Wird die Nachrichtenschleife in WinMain beendet, so erhält die ***main***-Funktion des Startup-Codes die Kontrolle zurück und weiß, dass das Programm nun zu beenden ist (das Ende der Nachrichtenschleife ist immer gleichbedeutend mit dem Schließen des Windowsprogramms) – alle automatisch geöffneten Systembibliotheken werden wieder geschlossen, Systemresourcen freigegeben usw.

Beim Borland-Compiler liegen diese Startup-Codes übrigens im Libs-Verzeichnis und haben die Dateierweiterung (Extension) obj.

Es gibt eine ganze Reihe von Startup-Codes, je nach Einstellungen wählt der Compiler automatisch den richtigen Code (der Standard-Startup-Code für 32bit-Windows-Programme ist z.B. *C0W32.OBJ*).

3.1 Fenster



3.2 Definieren einer Fensterklasse

Soll ein Windows-Programm über Ein- oder Ausgabewerte verfügen, so muss (mindestens) eine Fensterklasse definiert werden. Die Fensterklasse bestimmt das Aussehen und die Grundeigenschaften jeder Programminstanz[1] und der Client-Area. Die hier festgelegten Fenstereigenschaften werden bei der Fensterregistrierung (s.u.) von Windows gespeichert und dienen jeder Programmkopie als Vorlage. Erst nach Beendigung der letzte Programmkopie wird die Vorlage wieder aus der Fensterliste entfernt.

Windows benötigt zur Registrierung der Fensterklasse einen eindeutigen Namen (den Namen der Fensterklasse), um feststellen zu können, ob das gleiche Programm bereits im Speicher vorhanden ist.

Durch Kopieren und Umbenennen der ausführbaren Datei (***exe***) ist es daher nicht möglich Windows zu überlisten.

[1] wird das gleiche Programm mehrfach aufgerufen, so wird jede laufende Kopie (Task) als Instanz des Programms bezeichnet.

```
static char sWindowclassname [] = ”Win_01”;
WNDCLASS    aWindowclass;

aWindowclass.style         = CS_HREDRAW | CS_VREDRAW;
aWindowclass.lpfnWndProc   = WndProc;
aWindowclass.cbClsExtra    = 0;
aWindowclass.cbWndExtra    = 0;
aWindowclass.hInstance     = hInstance;
aWindowclass.hIcon         = LoadIcon (NULL,IDI_APPLICATION);
aWindowclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
aWindowclass.hbrBackground = GetStockObject (WHITE_BRUSH);
aWindowclass.lpszMenuName  = NULL;
aWindowclass.lpszClassName = sWindowclassname;
```

Erste Komponente der WNDCLASS-Struktur ist der Class-Style, der bestimmte Grundeigenschaften des Programms festlegt.

```
aWindowclass.style = CS_HREDRAW | CS_VREDRAW;
```

Der Style ist ein binäres Flagfeld, d.h. jedem Bit des Feldes kommt eine bestimmte Bedeutung zu, je nachdem ob es gesetzt ist oder nicht (der #define CS_HREDRAW entspricht z.B. 0x0002 und CS_VREDRAW hat den Wert 0x0001). Sollen mehrere Stile parallel verwendet werden, so werden die Styles mit einem binären ODER miteinander verbunden. Dieses Vorgehen ist zwar scheinbar mit einer Addition identisch, da es aber auch #define-Werte gibt, die bereits mehrere einzelne Flags zu typischen Flaggruppen zusammenfassen, würde eine „echte“ Addition ein falsches Ergebnis liefern. Die wichtigsten Class-Styles sind:

Addition über binär-ODER (Binär-Darstellung)	„normale“ Addition (Binär-Darstellung)
0000 0001 (= 1) | 0000 0011 (= 3) = 0000 0011 (= 3)	0000 0001 (= 1) + 0000 0011 (= 3) = 0000 0100 (= 4)

Class-Style	Bedeutung
CS_HREDRAW	Gibt an, dass das gesamte Fenster neu zu zeichnen ist, wenn Das Fenster in horizontaler Richtung bewegt oder in seiner Breite verändert wurde
CS_VREDRAW	Gibt an, dass das gesamte Fenster neu zu zeichnen ist, wenn Das Fenster in vertikaler Richtung bewegt oder in seiner Höhe verändert wurde
CS_NOCLOSE	Schaltet das Schließsymbol in der Titelzeile ab
CS_DBLCLKS	Weist Windows an, eine Doppelklick-Message an die Message-Handler-Function des Programms zu schicken, wenn der Benutzer einen Doppelklick in irgendeinem der zur Anwendung gehörenden Fenster eingibt.
Eine vollständige Liste kann der Windows-Hilfe des Compilers entnommen werden.	

Zweite Komponente der Struktur ist der Name der Handler-Funktion, die dem Hauptfenster zugeordnet ist. Der Name WndProc wird üblicherweise für die Handler-Funktion der Client-Area verwendet. Prinzipiell kann hier aber ein beliebiger Name stehen.

```
aWindowclass.lpfnWndProc = WndProc;
```

Ist der Name der Handler-Funktion in der WNDCLASS Struktur eingetragen, so wird diese Funktion über DispatchMessage automatisch aufgerufen, wenn eine Nachricht für die Client-Area ansteht (die Strukturkomponente lpfnWndProc entspricht einem Pointer auf die gewünschte Handlerfunktion).

Dritte und vierte Komponente der Struktur sind Angaben über Speicherbereiche, die Windows an der WNDCLASS Struktur für Benutzerinformationen bereithalten soll. Einmal für die Klassenstruktur (einmalig) und dann für jedes instanziierte Fenster (pro Programmkopie). Diese Komponenten werden nur selten verwendet und daher üblicherweise auf Null gesetzt.

```
aWindowclass.cbClsExtra = 0;
aWindowclass.cbWndExtra = 0;
```

Als fünfte Komponente wird die Instanzennummer übergeben.

```
aWindowclass.hInstance = hInstance;
```

Als sechste Komponente kann die ID eines Icons angegeben werden, welches von Windows benutzt werden soll, um das Programm zu repräsentieren (z.B. im Explorer und der Taskleiste). Will man kein eigenes Icon zeichnen, so kann man einfach das Standard-Icon verwenden, dessen Nummer global festgelegt ist (IDI_APPLICATION).

```
aWindowclass.hIcon = LoadIcon (NULL, IDI_APPLICATION);
```

Als siebte Komponente kann die ID eines Mauszeigers angegeben werden, welcher von Windows benutzt werden soll, wenn die Maus über die Client-Area des Programms bewegt wird. Im Normalfall wird einfach der Standard-Mauszeiger verwendet, dessen Nummer global festgelegt ist (IDC_ARROW).

```
aWindowclass.hCursor = LoadCursor (NULL, IDC_ARROW);
```

Als achte Komponente kann die ID eines Pinsels (Brush) angegeben werden, mit dessen Muster die Fläche der Client-Area bei Programmstart gefüllt wird. Meist wird hier ein Standardbrush verwendet.

```
aWindowclass.hbrBackground = GetStockObject (WHITE_BRUSH);
```

Als neunte Komponente kann der Name einer Menü-Struktur aus den Anwendungsresourcen (s.u.) angegeben werden. Will man kein Menü erzeugen, so wird hier als Wert NULL übergeben.

```
aWindowclass.lpszMenuName = NULL;
```

Als letztes wird der Name der Fensterklasse mit übergeben. Die meisten Programmierer verwenden hier den Namen des Programms.

```
aWindowclass.lpszClassName = sWindowclassname;
```

3.3 Registrierung dieser Fensterklasse(n)

Mit der Registrierung der Fensterklasse wird diese in den Listen der Windows-Verwaltung (im Kernel bekannt gemacht). Wird das gleiche Programm mehrfach gestartet (z.B. Word), dann kann Windows dies anhand der Fensterklasse erkennen und darauf verzichten, das komplette Programm ein zweites mal in den Speicher zu laden. Statt dessen wird nur ein neuer Datenbereich (Datensegment) erzeugt und der Programmbereich (Programmsegment) von beiden Instanzen gemeinsam benutzt.

```
if (!RegisterClass (&aWindowclass))
{
   MessageBox (NULL, "RegisterClass", "FEHLER", MB_OK);
   return 0;
}
```

3.4 Erzeugung eines oder mehrerer Fenster

Der nächste Schritt nach der Registrierung besteht üblicherweise in der Erzeugung eines Fensters der eben registrierten Fensterklasse(n). Dies initialisiert einen von Windows bereitgestellten Speicherblock, der eine Struktur vom Typ Window enthält, mit den Einstellungen der registrierten Fensterklasse(n).

```
HWND hWindow;

hWindow = CreateWindow (sWindowclassname, "Hurra",
          WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT,
          CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, hInstance,
          NULL);

if (!hWindow)
{
   MessageBox (NULL, "CreateWindow", "FEHLER", MB_OK);
   return 0;
}
```

Über die CreateWindow Funktion kann zudem festgelegt werden, wie das Fenster aussehen soll. Als Ergebnis der Funktion erhält man ein Handle auf das erzeugte Fenster, also eine eindeutige Identifikationsnummer, mit der das Fenster angesprochen werden kann. Diese ID wird u.a. auch dafür benötigt, um die auf der Client-Area befindlichen Controls ansprechen zu können.

```
CreateWindow (sWindowclassname,            //  1. Parameter
              "Titel",                      //  2. Parameter
              WS_OVERLAPPEDWINDOW,          //  3. Parameter
              CW_USEDEFAULT,                //  4. Parameter
              CW_USEDEFAULT,                //  5. Parameter
              CW_USEDEFAULT,                //  6. Parameter
              CW_USEDEFAULT,                //  7. Parameter
              NULL,                         //  8. Parameter
              NULL,                         //  9. Parameter
              hInstance,                    // 10. Parameter
```

```
                NULL);                          // 11. Parameter
```

Als erster Parameter wird angegeben, von welcher WNDCLASS das zu erzeugende Fenster sein soll.

Der zweite Parameter gibt an, welcher Text in der Titelzeile des Fensters erscheinen soll (sofern es eine Titelzeile gibt...)

Der dritte Parameter legt den Window-Style fest, also das Aussehen des Fensters. Der Style ist wieder ein binäres Flagfeld, sollen mehrere Styles verwendet werden, so werden die Styles mit einem binären ODER miteinander verbunden. Die wichtigsten Window-Styles sind der folgenden Tabelle zu entnehmen:

Styles für Titelzeilen-Elemente	**Bedeutung**
WS_CAPTION	Fenster mit Titelzeile (beinhaltet zugleich WS_BORDER)
WS_MAXIMIZEBOX	Fenster hat Vergrößerungsbutton in Titelzeile
WS_MINIMIZEBOX	Fenster hat Verkleinerungsbutton in der Titelzeile
WS_SYSMENU	Erzeugt ein Fenster mit dem System-Menü in der Titelzeile. Dazu muss Stil WS_CAPTION gesetzt sein
Styles für Rand-Elemente	**Bedeutung**
WS_BORDER	Fenster mit schmalem Rand
WS_THICKFRAME	Fenster mit breitem Rand, über den Fenstergröße eingestellt werden kann Identisch mit WS_SIZEBOX
WS_SIZEBOX	Identisch mit WS_THICKFRAME
WS_HSCROLL	Erzeugt ein Fenster mit einer horizontalen Scrollbar
WS_VSCROLL	Erzeugt ein Fenster mit einer senkrechten Scrollbar
Styles für Programmstart	**Bedeutung**
WS_ICONIC	Erzeugt Fenster, das zu Beginn als Icon vorliegt
WS_MAXIMIZE	Erzeugt ein Fenster maximaler Größe
WS_MINIMIZE	identisch mit WS_ICONIC
WS_DISABLED	Fenster ist abgeschaltet
Zusammengefasste Styles	**Bedeutung**
WS_OVERLAPPED	Erzeugt sogenanntes OVERLAPPED Fenster Hat immer Titelzeile (WS_CAPTION) und Rand (WS_BORDER) Identisch mit WS_TILED
WS_TILED	Identisch mit WS_OVERLAPPED
WS_DLGFRAME	Fenster mit dem typischen Aussehen einer Dialogbox. Kann keine Titelzeile (WS_CAPTION) haben
Styles Pakete	**Bedeutung**
WS_OVERLAPPEDWINDOW	Erzeugt ein Fenster mit den Stilen WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX, und WS_MAXIMIZEBOX
WS_TILEDWINDOW	identisch mit WS_OVERLAPPEDWINDOW
Styles für Childfenster	**Bedeutung**
WS_CHILD	Definiert ein Unterfenster (Dieser Style kann nicht auf die Client-Area angewendet werden!) Schließt sich mit WS_POPUP gegenseitig aus
WS_CHILDWINDOW	identisch mit WS_CHILD
WS_CLIPCHILDREN	Die Bereiche von Childfenstern werden

	ausgelassen, wenn das übergeordnete Fenster neu gezeichnet wird
WS_CLIPSIBLINGS	Die Bereiche von überlappenden Childfenstern werden beachtet, so dass nicht in die überlappenden Bereiche geschrieben/gezeichnet wird
WS_POPUP	Erzeugt ein Dialogfenster. (nicht für Client-Area möglich!) Schließt sich mit WS_CHILD gegenseitig aus
WS_POPUPWINDOW	Erzeugt ein Dialogfenster mit Standardaussehen Beinhaltet die Styles WS_BORDER, WS_POPUP, und WS_SYSMENU Muss mit WS_CAPTION kombiniert werden, um das System-Menü sichtbar zu machen
Styles für Dialogelemente	**Bedeutung**
WS_GROUP	Bezeichnet das erste Fenster einer Child-Gruppe Wird für die Fenster von Buttons und andere Dialogelemente verwendet Zwischen Fenstern der Gruppe kann der Focus mit den Cursortasten bewegt werden Das nächste Fenster mit dem WS_GROUP beendet die Gruppe wieder und startet eine neue Gruppe
WS_TABSTOP	Besagt, dass Dialogelement mittels Tabulatortaste erreicht werden kann
WS_VISIBLE	Erzeugt sichtbares Dialogelement Unsichtbare Elemente sind immer zugleich auf WS_DISABLED geschaltet
Eine vollständige Liste kann der Windows-Hilfe des Compilers entnommen werden.	

Der vierte und fünfte Parameter legen die Fensterposition fest (x- und y-Koordinate). Wird hier die Konstante CW_USEDEFAULT übergeben, so verwendet Windows die Standard-Voreinstellungen.

Der sechste und siebte Parameter legen die Fensterbreite und Fensterhöhe fest (in Pixeln). Wird hier die Konstante CW_USEDEFAULT übergeben, so verwendet Windows die Standard-Voreinstellungen.

Der achte Parameter ist das Handle des übergeordneten Fensters. Dies ist nötig, damit Windows beim Schließen eines Dialoges auch alle abhängigen Fenster (z.B. Controls) feststellen und schließen kann. Hier ist NULL einzutragen, wenn es sich um die Client-Area handelt.

Der neunte Parameter ist in seiner Bedeutung davon abhängig, ob es sich um ein Childfenster (z.B. ein Control) handelt oder um das Hauptfenster (Client-Area). Bei der Client-Area ist hier das Handle des darzustellenden Menüs einzutragen (oder NULL wenn kein Menü gewünscht wird). Bei dynamisch erzeugten Childfenstern ist hier eine eindeutige ID einzutragen, mit der das Childfenster angesprochen werden kann.

Parameter zehn ist das Instanzenhandle. Hier wird eingetragen, zu welcher Aufrufinstanz das Fenster gehört. Das ist wichtig, damit in den Dialogen die richtigen Datenbereiche angezeigt werden.

Parameter elf enthält bei Bedarf den Pointer auf die in der Klasse reservierten Speicherbereiche (s.o.). Dieser Parameter wird nur selten genutzt.

3.5 Anzeigen des Fensters

Das Erzeugen des Fensters beinhaltet nicht, dass das Fenster auch angezeigt wird. Gelegentlich kann es von Vorteil sein, im Hintergrund eine Verarbeitung vorzunehmen und das zugehörige Fenster nur bei Bedarf anzeigen. Deshalb muss das Aufblenden des Fensters ausdrücklich aufgerufen werden.
Als Parameter übergibt man das Handle des erzeugten Fensters (Identifikationsnummer) und den Darstellungsmodus, der als Eigenschaft (z.B. "als Icon") von Windows im Parameter nCmdShow an WinMain übergeben wurde. Dieser Befehl ShowWindow darf nur an einer einzigen Stelle, hier im WinMain verwendet werden), Erst danach wird das Fenster auch für den Benutzer sichtbar.

```
ShowWindow (hWindow, nCmdShow);

if (!UpdateWindow (hWindow))
{
   MessageBox (NULL, "UpdateWindow", "FEHLER", MB_OK);
   return 0;
}
```

Das Aufblenden des Fensters mit ShowWindow zeigt im Grunde nur die Fläche des Fensters (ohne Inhalte) an. Aus Geschwindigkeitsgründen werden die Fensterinhalte nur bei Bedarf oder bei ausdrücklicher Anforderung gemalt. Erst der Befehl UpdateWindow veranlasst Windows das ganze Fenster neu zeichnen (Es wird eine WM_PAINT Message ausgelöst - s.u.)

3.6 Einrichten der lokalen Message-Loop

Das Fenster ist nach UpdateWindow bereit und gezeichnet, jetzt ist auf alle ankommenden (interessanten) Botschaften zu reagieren. Dazu verwendet man meist folgende MessageLoop:

```
while (GetMessage (&aMessage, NULL, 0, 0))
{
   TranslateMessage (&aMessage);
   DispatchMessage  (&aMessage);
}
return aMessage.wParam;
```

Diese besteht aus einer Endlosschleife, die Nachrichten von der Main-Message-Loop des Windows-Systems erhält (Receiving[2], abholen der Nachrichten für das Gesamtprogramm), wenn nötig übersetzt (Translation, einige Tastaturcodes müssen gesondert behandelt werden) und programmintern an die einzelnen Handlerfunktionen

[2] to receive [engl.]: empfangen (einer Nachricht)

verteilt (Dispatching[3], Verteilung an untergeordnete Fenster und ihre Message-Bearbeitungsfunktionen).
GetMessage gibt solange den Wert TRUE zurück, bis eine WM_QUIT Message anzeigt, dass das Programm beendet wurde. Die WM_QUIT Message ist die letzte Message, die ein Programm erhält.
Bei ungültigen Messages (z.B. Nachrichten an nicht mehr vorhandenes Window-Handle) ergibt die Auswertung in GetMessage den Wert –1.

Bei Bedarf kann eine MessagePreprocess-Funktion eingeführt werden, welche die ankommenden Messages nach Nachrichten von systemweiter Bedeutung durchsucht (z.B. Hilfetaste). Üblicherweise sind dies Tastaturcodes, die überall im System eine gleiche Bedeutung haben sollen und auch über eine einheitliche Funktion abgearbeitet werden können.

```
while (GetMessage (&aMessage, NULL, 0, 0))
{
   MessagePreprocess (hWindow, aMessage);
   TranslateMessage  (&aMessage);
   DispatchMessage   (&aMessage);
}
return aMessage.wParam;
```

Das Abfangen des Tastaturcodes muss vor dem Message-Dispatching erfolgen, da der Tastaturcode sonst an das entsprechende Controlwindow (Dialogelement) geschickt wird. Dort könnte die Taste zwar ebenfalls abgefragt werden, dann jedoch müsste man diese Abfrage an jedem einzelnen Control wiederholen, um von jeder Stelle der Dialogmaske aus die globale Funktion ausführen zu können. Ein solches Vorgehen wäre jedoch sehr umständlich, es ist einfacher die gewünschten Tastaturcodes vorher (hier mit der Funktion MessagePreprocess) herauszufiltern.

TranslateMessage übersetzt die Windows-Tastaturcodes (Virtual Keys) in „echte“ Characterdaten. Natürlich nur, wenn solche Daten auch in der Message vorliegen. Der TranslateMessage-Aufruf muss gemacht werden, da man sonst mit den Keymessages nichts anfangen kann.

DispatchMessage gibt die Message-Struktur an die Handlerfunktion weiter, die in der Windowsklasse eingetragen wurde (s.o.). Erst dort wird die Nachricht endgültig verarbeitet. Sind mehrere Fenster vorhanden, so entscheidet der Dispatcher anhand des in der Message eingetragenen Handles, an welche Handlerroutine die Message zu gehen hat.

Mit dem Ende der MessageLoop ist das Programm beendet, es ist Windows lediglich noch mitzuteilen, dass die WM_QUIT Message erfolgreich verarbeitet wurde, indem man den wParam der Message an Windows zurückgibt.

[3] to dispatch [engl.]: verteilen

3.7 Ein minimales Windows-Programm

Das nachstehende Programm ist ein (fast) minimaler Windows-Programmrumpf. Verzichtbar sind allenfalls noch die Fehlermeldungen, die Funktion ***MessagePreprocess*** (Wenn man keine Sondertasten abfangen möchte) und der ***case***-Fall WM_PAINT in der Funktion ***WndProc***, da es in einem Rumpfprogramm natürlich noch keine benutzerdefinierten Graphikausgaben geben kann.

```
#include <windows.h>

LRESULT CALLBACK WndProc (HWND, UINT, UINT, LONG);
BOOL MessagePreprocess (HWND hWnd, MSG &aMessage);

//-------------------------------------------------------------
WINAPI WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                     LPSTR sCmdParam, int nCmdShow)
{
   static char sWindowclassname [] = "P1";
   HWND        hWindow;
   MSG         aMessage;
   WNDCLASS    aWindowclass;

   if (!hPrevInstance)
   {
      aWindowclass.style         = CS_HREDRAW | CS_VREDRAW;
      aWindowclass.lpfnWndProc   = WndProc;
      aWindowclass.cbClsExtra    = 0;
      aWindowclass.cbWndExtra    = 0;
      aWindowclass.hInstance     = hInstance;
      aWindowclass.hIcon         = LoadIcon (NULL,
                                        IDI_APPLICATION);
      aWindowclass.hCursor       = LoadCursor (NULL,
                                         IDC_ARROW);
      aWindowclass.hbrBackground = GetStockObject
                                           (WHITE_BRUSH);
      aWindowclass.lpszMenuName  = NULL;
      aWindowclass.lpszClassName = sWindowclassname;

      if (!RegisterClass (&aWindowclass))
      {
         MessageBox (NULL, "RegisterClass", "FEHLER", MB_OK);
         return 0;
      }
   }

   hWindow = CreateWindow (sWindowclassname, "Rumpfprogramm",
         WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT,
         CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, hInstance,
         NULL);

   if (!hWindow)
   {
      MessageBox (NULL, "CreateWindow", "FEHLER", MB_OK);
      return 0;
   }

   ShowWindow (hWindow, nCmdShow);

   if (!UpdateWindow (hWindow))
   {
      MessageBox (NULL, "UpdateWindow", "FEHLER", MB_OK);
      return 0;
```

```
   }

   while (GetMessage (&aMessage, NULL, 0, 0))
   {
      MessagePreprocess (hWindow, aMessage);
      TranslateMessage (&aMessage);
      DispatchMessage (&aMessage);
   }
   return aMessage.wParam;
}

//-------------------------------------------------------------
BOOL MessagePreprocess (HWND hWnd, MSG &aMessage)
{
   if (WM_KEYUP == aMessage.message)
   {
      if (VK_F1 == (int)aMessage.wParam)
      {
         WinHelp (hWnd, "C:\\WINDOWS\\HELP\\WINHLP32.HLP",
                HELP_INDEX, 0);
         return TRUE;
      }
   }
   return FALSE;
}

//-------------------------------------------------------------
LRESULT CALLBACK WndProc (HWND hWnd, UINT nMsg,
                          UINT wParam, LONG lParam)
{
   HDC         hDC;
   PAINTSTRUCT aPS;

   switch (nMsg)
   {
      case WM_PAINT :
           hDC = BeginPaint (hWnd, & aPS);
           EndPaint (hWnd, & aPS);
           return 0;

      case WM_DESTROY :
           PostQuitMessage (0);
           return 0;
   }
   return DefWindowProc (hWnd, nMsg, wParam, lParam);
}
```

3.8 Windows-Programmstart

Im Zuge des Programmstarts erhält jedes Windowsprogramm bereits eine ganze Reihe von Messages, die der Anwendungsprogrammierer benutzen kann, um das Programm in gewünschter Weise vorzubereiten und zu initialisieren.
Startet man das oben abgebildete Minimalprogramm, so erhält man (in der angegebenen Reihenfolge) die folgenden Nachrichten:

Message	Bedeutung
WM_GETMINMAXINFO	Die WM_GETMINMAXINFO Nachricht wird benutzt, um die maximale und minimale Größe zu ermitteln, die ein Fenster annehmen kann.
WM_NCCREATE	Die WM_NCCREATE Nachricht wird vor der WM_CREATE Nachricht gesendet, wenn ein

Message	Bedeutung
	Fenster das erste Mal erzeugt wird (D.h. die Fensterklasse noch nicht registriert war).
WM_NCCALCSIZE	Die WM_NCCALCSIZE wird gesendet, wenn die Größe und Position der Client-Area berechnet werden muss. Wenn ein Programmierer diese Nachricht abfängt und verarbeitet, kann die Größe und Position der Client-Area kontrolliert und beeinflußt werden, wenn sich Größe oder Position des Hauptfensters ändern.
WM_CREATE	Diese Nachricht wird gesendet, wenn das Programm über die Funktionen CreateWindowEx oder CreateWindow ein Fenster öffnen will. Die Handle-Funktion erhält diese Nachricht **nachdem** das Fenster erzeugt wurde, aber **bevor** es sichtbar wird. Die Nachricht wird erzeugt, bevor die Funktionen CreateWindowEx bzw. CreateWindow ihren Returnwert an die aufrufende Funktion zurückgeben, d.h. der Erhalt der Nachricht sagt nichts darüber aus, ob das Fenster erzeugt werden konnte.
WM_SHOWWINDOW	Die WM_SHOWWINDOW Nachricht wird gesendet, wenn ein Fenster angezeigt oder verborgen werden soll, bzw. wenn ein Fenster zwischen voller Bildschirmgröße und Icon-Zustand umgeschaltet werden soll.
WM_WINDOWPOSCHANGING	Die WM_WINDOWPOSCHANGING Nachricht wird an ein Fenster gesendet, dessen Größe, Position oder Überlagerungsreihenfolge sich aufgrund eines Funktionsaufrufes ändern soll (z.B. durch Aufruf von SetWindowPos).
WM_WINDOWPOSCHANGING	***Diese Nachricht wird tatsächlich 2x geschickt!*** Die WM_WINDOWPOSCHANGING Nachricht wird an ein Fenster gesendet, dessen Größe, Position oder Überlagerungsreihenfolge sich aufgrund eines Funktionsaufrufes ändern soll (z.B. durch Aufruf von SetWindowPos).
WM_ACTIVATEAPP	Die WM_ACTIVATEAPP Nachricht wird gesendet, wenn ein Fenster aktiviert werden soll, das zu einem anderen als dem gerade aktiven Task gehört. Die Nachricht wird an alle Hauptfenster der zu aktivierenden und des zu deaktivierenden Tasks gesendet.
WM_NCACTIVATE	Die WM_NCACTIVATE Nachricht wird an ein Fenster geschickt, wenn der Non-Client-Area-Bereich (z.B. die Titelzeile) geändert werden muss, um anzuzeigen, dass der Task aktiviert oder deaktiviert ist (u.a. angezeigt durch den Farbzustand der Titelzeile)
WM_GETTEXT	WM_GETTEXT wird aufgerufen, um den Titeltext eines Fensters aus der Fensterstruktur auszulesen (Hier Titeltext des Programms).
WM_ACTIVATE	Die WM_ACTIVATE Nachricht wird gesendet, wenn ein Fenster aktiviert werden soll. Die Nachricht wird zuerst an das zu deaktivierenden Fenster gesendet, anschließend an das zu aktivierende Fenster.
WM_SETFOCUS	Die WM_SETFOCUS Nachricht wird an ein Fenster gesendet, nachdem das Fenster den

Message	Bedeutung
	Eingabefokus bekommen hat, d.h. dass das Fenster jetzt Eingaben des Benutzers entgegennehmen kann.
WM_NCPAINT	Die WM_NCPAINT Nachricht besagt, dass die Zeichnung der Non-Client-Area (Rahmen) aufgefrischt werden muss.
WM_GETTEXT	***Da der Rahmen neu gezeichnet wird, muss auch der Titel neu geschrieben werden.*** WM_GETTEXT wird aufgerufen, um den Titeltext eines Fensters aus der Fensterstruktur auszulesen (Hier Titeltext des Programms).
WM_ERASEBKGND	Die WM_ERASEBKGND Nachricht wird immer dann gesendet, wenn der Hintergrund der Client-Area zu löschen ist (z.B. nach einer Änderung der Fenstergröße oder einem Verschieben des Fensters). Die Nachricht wird gesendet, um einen invalidieren Bereich auf das Neuzeichnen vorzubereiten.
WM_WINDOWPOSCHANGED	Die WM_WINDOWPOSCHANGED Nachricht wird an ein Fenster gesendet, dessen Größe, Position oder Überlagerungsreihenfolge sich aufgrund eines Funktionsaufrufes geändert hat. (z.B. durch Aufruf von SetWindowPos).
WM_SIZE	Die WM_SIZE Nachricht wird an ein Fenster gesendet, wenn sich dessen Größe verändert hat.
WM_MOVE	Die WM_MOVE Nachricht wird an ein Fenster gesendet, wenn sich dessen Position verändert hat.
WM_PAINT	Die WM_PAINT Nachricht besagt, dass das Fenster aufgefrischt werden muss, weil es invalidierte Bereiche gibt. Die Nachricht wird gesendet, wenn entweder die Funktion UpdateWindow oder die Funktion RedrawWindow aufgerufen wurde.

4 Fensterverwaltung

Die folgenden Abschnitte gehen kurz auf interessante, ausgewählte Aspekte der Fensterverwaltung unter Windows ein.

4.1 Handles

Bisher ist immer noch die Frage offen, woher Windows denn weiß, an welchen der vielen Handler im System eine bestimmte Botschaft gehen soll.
Dies ist eines der fundamentalen Elemente von Windows, die sogenannten Handles. Jedes Fenster besitzt ein Handle, womit schlicht eine eindeutige Identifikationsnummer gemeint ist, die das Fenster und/oder Programm identifiziert. Da auch Controls nur eine Sonderform von Fenstern sind, heißt dies letztlich, dass auch jedes Editfeld, jeder Schalter, jede Listbox usw. über eine eindeutige Identifikation verfügen.
Der Vorteil der Handles gegenüber einem „herkömmlichen Ansatz“ (der vermutlich Adressen speichern würde) liegt darin, dass die Code- und Datensegmente nicht an bestimmten Speicherstellen liegen, sondern von Windows nahezu beliebig verlagert werden können (siehe Speicherverwaltung).
Da der Bildschirm zumeist eine ganze Reihe von Fenstern und Controls aufweist, ist er geradezu überzogen mit einem Flickwerk von kleinen, rechteckigen Arealen, die jeweils über ein eigenes Handle verfügen.
Jedes dieser Programmfenster hat wiederum ein hierarchisch übergeordnetes Fenster (einschließlich der Client-Area), deren höchste Stufe der Programm-Manager selbst ist.
Windows führt dementsprechend eine interne Liste, welche die umschließenden Rechtecke der einzelnen Fenster in der korrekten hierarchischen (übereinanderliegenden) Form enthält. D.h. wenn sich der Mauszeiger an einer bestimmten Stelle des Bildschirm befindet, kann Windows den zugehörigen Handle anhand der Bildschirmkoordinaten eindeutig ermitteln.
Windows schickt sodann die betreffenden Messages an das zugehörige Programm. Die Applikation empfängt die Botschaft (in der Message-Loop) und wertet wiederum das in der Botschaft enthaltene Handle aus. Gemäß dieses Handleeintrags ermittelt es aus der zugehörigen Fensterstruktur die Adresse der zuständigen Handlerfunktion und ruft diese bei gleichzeitiger Weiterleitung der Message auf.

4.2 Paint-Message und Invalidierung

Von herausragender Bedeutung ist die WM_PAINT Message, die Windows immer dann sendet, wenn ein Teil des Fensters neu gezeichnet werden muss. Dies ist z.B. der Fall, wenn ein Fenster ein anderes ganz oder teilweise verdeckt. Wird das überlagernde Fenster geschlossen, so muss der dahinterliegende Bildschirmteil neu gezeichnet werden. Dazu sendet Windows allen betroffenen Programmen eine Nachricht, die dazu auffordert, den eigenen Bildschirmbereich wieder herzustellen. Diese Botschaft heißt WM_PAINT. Nun ist das Zeichnen des Bildschirms eine sehr

zeitaufwendige Angelegenheit, so dass man unter Windows versucht hat, den entsprechenden Aufwand zu minimieren. Das Zauberwort dazu nennt sich „Invalidierung“[1] und beschreibt den Umstand, dass Windows mehr oder weniger automatisch alle veränderten Bereiche als ungültig markiert. Nur die ungültigen Bereiche werden anschließend (beim nächsten WM_PAINT) neu gezeichnet. dadurch hält sich der Zeitaufwand meist in erträglichen Grenzen. Die Standard-Controls (wie Listboxen, Buttons etc.) behandelt Windows automatisch (schließlich zieht das Setzen eines Editfeldes logischerweise auch dessen Invalidierung nach sich) - es sei denn man möchte ungewöhnliche (Nichtstandard) Verhaltensweisen, wie z.B. individuelle Hintergrundfarben pro Editfeld.
In diesem Fall muss man in der WM_PAINT Behandlung dem System ein wenig auf die Sprünge helfen.
Da der WM_PAINT Befehl so zeitaufwendig ist, sollte man nie selbst eine WM_PAINT Message senden, sondern lediglich den neuzuzeichnenden Bereich invalidieren. Windows schickt dann eine WM_PAINT Message, sowie die entsprechenden Resourcen es zulassen.

4.3 Device-Context

Der Device-Context[2] ist eine weitere, praktische Eigenheit von Windows, die es dem Anwendungsentwickler ermöglicht unabhängig von den technischen Eigenschaften des Zielcomputers zu arbeiten. Der Grundgedanke dahinter ist, dass der Entwickler die Zeichen-Funktionen (wobei die Textausgabe als Sonderform des Zeichnens gewertet wird) implementiert, ohne dass er das Ausgabegerät kennen muss. Wird das Zeichnen im Zusammenhang mit einem Monitor-Context aufgerufen, so landet die Ausgabe auf dem Bildschirm, wird dein Printer-Context verwendet, so erfolgt eine Druckausgabe. Die notwendigen spezifischen Kenntnisse (z.B. über die Auflösung des Ausgabegerätes) werden über den Device-Context definiert und immer gleich abgearbeitet. Der windowskonforme Gerätetreiber hat in diesem Fall dafür zu sorgen, dass die entsprechenden Informationen auf Anfrage des Programms zur Verfügung gestellt werden. Dadurch kann jede systemkonform programmierte Anwendung (theoretisch) mit jeder Graphikkarte und jedem Drucker zusammenarbeiten, ohne Kenntnisse über dessen Eigenheiten haben zu müssen.

[1] Invalid [engl.]: ungültig
[2] device [engl.]: Gerät
context [engl.]: Bezug, Zusammenhang

5 Weitere Windowseigenschaften

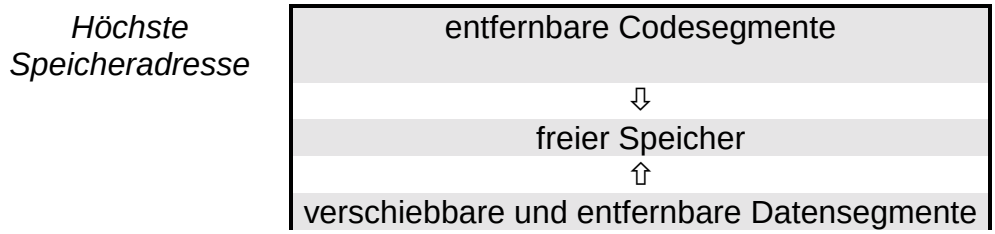
Die folgenden Abschnitte gehen kurz auf grundlegende Aspekte der Speicherverwaltung unter Windows ein.

5.1 Speicherverwaltung

Windows unterscheidet grundsätzlich zwischen Code- und Datensegmenten. Diese Unterscheidung macht durchaus Sinn, denn die von einem Programm verwendeten Codesegmente bleiben (fast immer) konstant, die Datensegmente hingegen werden fortwährend verändert. Von dieser Basis ausgehend hat man in Windows einen Mechanismus installiert, der es ermöglicht ein Programm mehrfach aufzurufen, die zugehörigen Codesegmente aber nur ein einziges Mal zu laden. Die Datensegmente hingegen werden für jede Instanz (Programmaufruf) neu angelegt. Windows unterscheidet die einzelnen Programmkopien anhand eines Instanzenzählers, der mit jedem Programm-Modul mitgeführt wird.

Zusätzlich sind Speicheradressen in Windows nicht festgelegt. Vielmehr sind die einzelnen Code- und Datensegmente verschiebbar (moveable). Dieses Vorgehen hat den Hintergrund, dass die Windows-Programmierer damit erreichen wollten, dass sehr große Programme auch mit sehr wenig Speicher ausführbar sein sollten. Der resultierende Mechanismus ist der sogenannte „virtuelle Speicher“ (Virtual Memory[1]). Immer wenn nötig (bei Platzbedarf) verschiebt Windows ein als „moveable“ gekennzeichnetes Segment, welches gerade nicht benötigt wird, in den virtuellen Speicher (auf die Festplatte). Wird ein Teil dieses Segmentes angesprochen, so lädt das Betriebssystem das Segment wieder in den Hauptspeicher, zumeist im Austausch gegen ein anderes Segment. Dabei kann man natürlich nicht davon ausgehen, dass das Segment sich anschließend noch an der gleichen Stelle befindet. entsprechend Schwierig gestalten sich natürlich die Speicherzugriffe. Glücklicherweise kümmert sich Windows um diese Problematik, so dass man nur selten auf „fixed“ Segmente (nicht auslagerbare Blöcke) zurückgreifen muss.

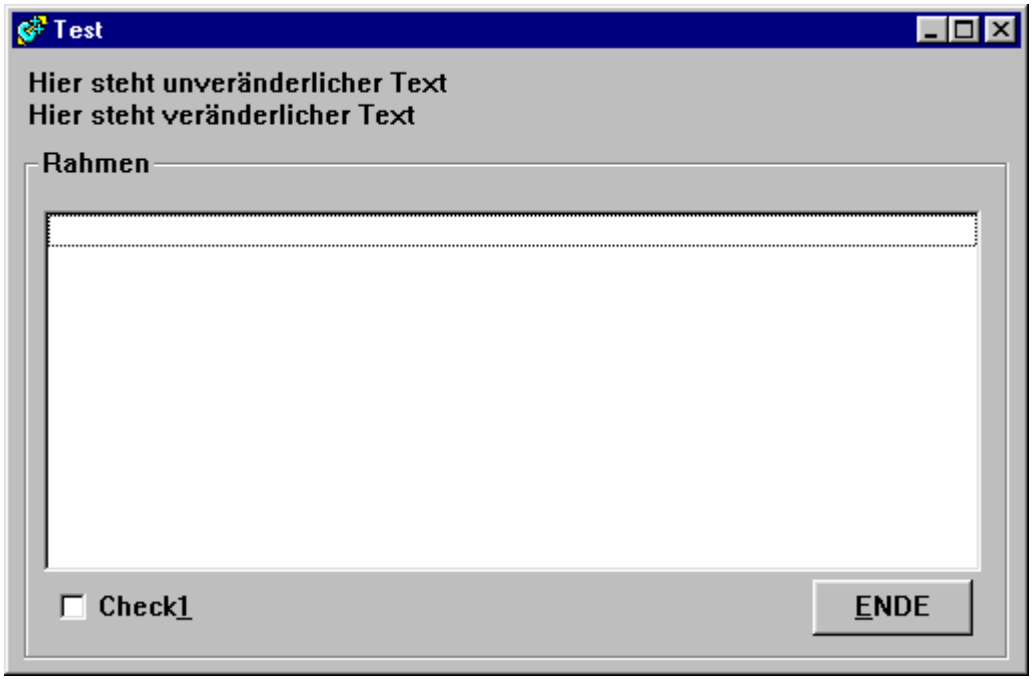
Bei Codesegmenten geht Windows sogar noch weiter, indem es die Codesegmente als „discardable“ kennzeichnet. Anstatt das Codesegment in den virtuellen Speicher zu verschieben, wird das komplette Segment gelöscht und bei Bedarf aus der EXE-Datei nachgeladen.



[1] **virtual** [engl.]: wirklich, tatsächlich, eigentlich, virtuell – in der EDV bezeichnet der Begriff „virtual memory“ die Verwendung von Festplattenbereichen als Erweiterung des Speichers. In Windows auch als „Auslagerungsdatei“ bekannt.
memory [engl.]: Gedächtnis, Speicher

5.2 Anwendungs-Resourcen

Anwendungs-Resourcen sind in Windows eingeführt worden, um die Übersetzung von Windows Programmen in andere Sprachen und somit die weltweite Vermarktung von Anwendungen zu erleichtern. Dabei werden die Fensteroberflächen in einem kodierten Format (an die bereits in Maschinensprache übersetzten Programme) angehängt.
Ziel war es, die bereits fertigen Programme nachträglich (im Aussehen, nicht in der Funktionalität) verändern zu können. Sofern das Aussehen der Dialoge nicht (hart codiert) programmintern verändert wird, genügt es die angehängten Teile z.B. gegen fremdsprachige Dialoge auszutauschen. Üblicherweise verwendet man dazu sogenannte Resource-Editoren, die für jeder gewünschte Control eine Identifikationsnummer (Resource-ID) erzeugen und in einem Headerfile abspeichern (meist mit der Datei-Endung ***rh*** oder ***rch***). Die Definition der gewünschten Controls wird als spezielles Textformat in einer Resourcedatei gespeichert (meist mit der Datei-Endung ***rc***).
Diese Resourcedatei (gegebenenfalls auch mehrere) nimmt man in das Projekt mit auf. Anhand der Datei-Endung erkennt der Compiler, dass es sich um eine Resourcedatei handeln soll und ruft den Resource-Compiler auf, der die Textdatei in ein spezielles Format übersetzt (die Ergebnisdatei trägt meist die Datei-Endung ***res***) – dieser Vorgang ist vergleichbar mit einem normalen Compiler-Lauf, der aus einem Quelltext (***cpp***) eine Objektdatei (***obj***) macht.
Die compilierte Resourcedatei wird beim Linking an das fertige Programm angehängt und die einzelnen Control-Elemente werden vom Programm aus allein über ihre ID-Nummer angesprochen. Da es für jedes Control einen festen Aufbau der Eigenschaften gibt (vergleichbar einem Datensatz), kann man ein Control (ein geeignetes Werkzeug vorausgesetzt) nachträglich, d.h. im fertigen Programm, noch verändern – z.B. eine Listbox vergrößern oder Beschriftungen übersetzen. Dies funktioniert natürlich nur, wenn die Codenummern der dort abgelegten Dialogelemente unverändert erhalten bleiben, denn über diese Nummern werden die einzelnen Controls angesprochen und verarbeitet. Beschreibende Control-Elemente (wie z.B. Buttonbeschriftungen) können aber problemlos ausgetauscht werden.
Die folgenden Listings zeigen die notwendigen Einträge, um die unten abgebildete Dialogbox als Resourcen abzulegen.




```
/*********************************************************
test.rh
produced by Borland Resource Workshop
*********************************************************/
#define IDD_TEST     101
#define LB_LISTBOX1  102
#define CK_CHECK1    103
#define B_ENDE       104
#define ST_TEXT1     105
#define ID_GROUPBOX1 106
```

```
/*********************************************************
test.rc
produced by Borland Resource Workshop
*********************************************************/

#include "test.rh"

IDD_TEST DIALOG 0, 0, 252, 154
         STYLE WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU |
               WS_MINIMIZEBOX | WS_MAXIMIZEBOX | WS_THICKFRAME
               CAPTION "Test"
{
   CONTROL "Rahmen", ID_GROUPBOX1, "BUTTON",
           BS_GROUPBOX | WS_CHILD | WS_VISIBLE,
           3, 24, 246, 128
   CONTROL "ListBox1", LB_LISTBOX1, "listbox",
           LBS_STANDARD | LBS_HASSTRINGS | LBS_USETABSTOPS |
           WS_HSCROLL | WS_TABSTOP,
           8, 40, 234, 96
   CONTROL "Check&1", CK_CHECK1, "BUTTON",
           BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE |
           WS_TABSTOP, 12, 133, 40, 12
   CONTROL "&ENDE", B_ENDE, "BUTTON",
           BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP,
           200, 132, 40, 14
   CONTROL "Hier steht unveränderlicher Text", -1, "STATIC",
           SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP,
           4, 4, 240, 8
   CONTROL "Hier steht veränderlicher Text", ST_TEXT1,
```

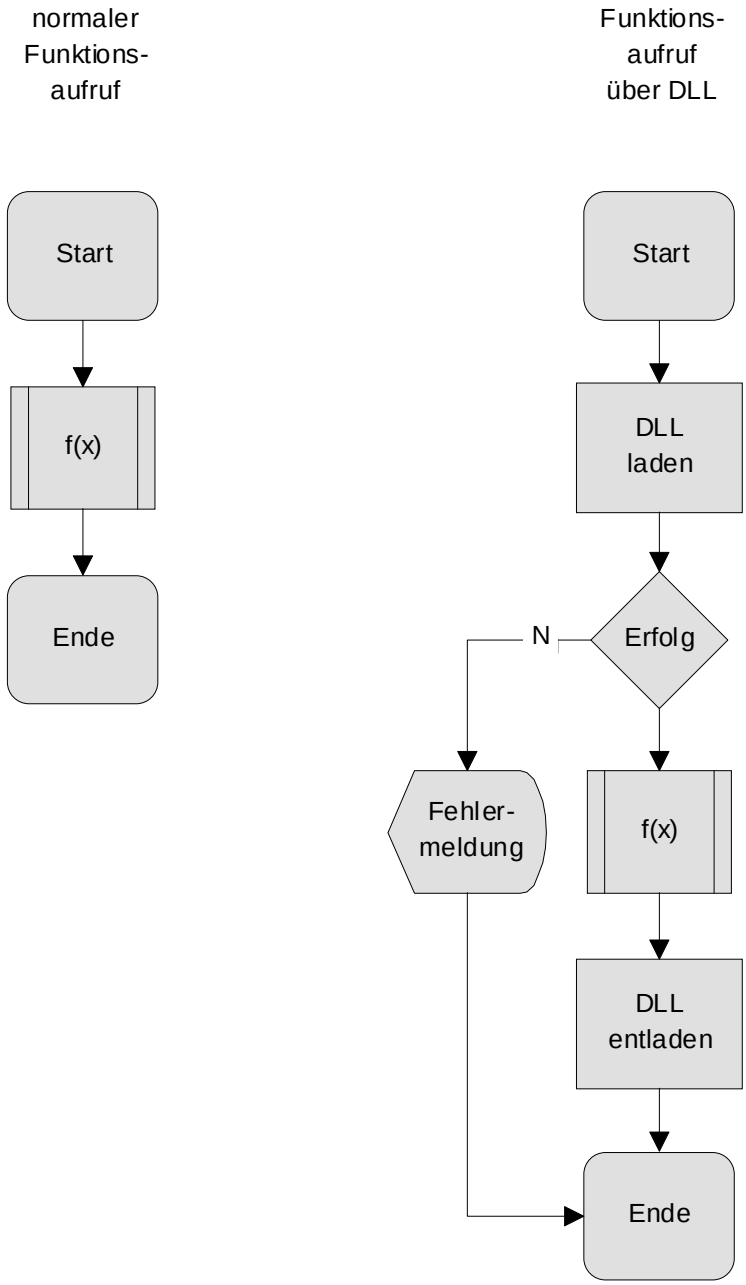
```
            "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE |
            WS_GROUP, 4, 12, 244, 8
}
```

5.3 Dynamic Link Libraries (DLLs)

Eine DLL ist eine Funktionssammlung, die nicht fest in ein Programm integriert ist. Statt dessen wird die Funktionssammlung erst bei Bedarf geöffnet. Dieses Konzept hat gleich zwei Vorteile:

- Die Belastung des Hauptspeichers geringer, da die Programmcodes erst dann geladen werden wenn sie gebraucht werden und nach Gebrauch sofort wieder entladen werden können.
- Die DLL kann, wenn Fehler entdeckt und entfernt wurden, ausgetauscht werden, ohne dass das aufrufende Hauptprogramm neu übersetzt werden muss.

Um mit DLL-Techniken zu arbeiten müssen natürlich trotzdem einige Spielregeln eingehalten werden. So kann z.B. eine DLL nur solange ausgetauscht werden, solange sich die Parameterleisten der darin enthaltenen Funktionen nicht ändern.
Zudem erfordert die Verwendung von DLLs natürlich auch mehr Sorgfalt bei der Programmierung, denn vor Nutzung einer Funktion die in einer DLL gespeichert ist, muss der Programmierer dafür sorgen, dass die entsprechende DLL geladen und später wieder entfernt wird (siehe Graphik).



5.4 API

API ist die Abkürzung für „Application Programmers Interface“. Ein API ist nichts weiter als eine Sammlung von Programmfunktionen (in C/C++ würde man von einer Funktions-Bibliothek sprechen). Das wichtigste API ist das Windows-API, welches alle Grundfunktionen von Windows zur Verfügung stellt. Das Windows-API umfasst das Windows-Kernel, sowie alle Erweiterungen (das sind üblicherweise die DLLs aus dem Windows-System-Verzeichnis).

5 Weitere Windowseigenschaften

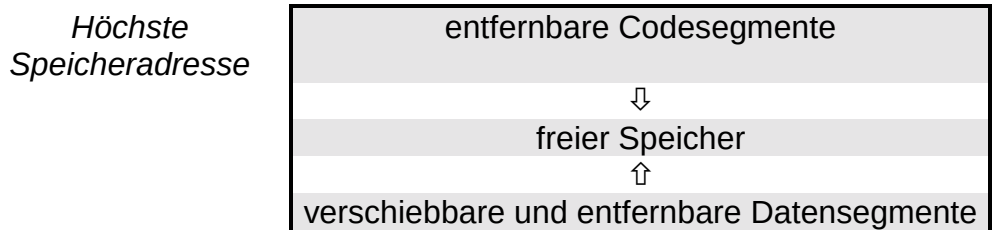
Die folgenden Abschnitte gehen kurz auf grundlegende Aspekte der Speicherverwaltung unter Windows ein.

5.1 Speicherverwaltung

Windows unterscheidet grundsätzlich zwischen Code- und Datensegmenten. Diese Unterscheidung macht durchaus Sinn, denn die von einem Programm verwendeten Codesegmente bleiben (fast immer) konstant, die Datensegmente hingegen werden fortwährend verändert. Von dieser Basis ausgehend hat man in Windows einen Mechanismus installiert, der es ermöglicht ein Programm mehrfach aufzurufen, die zugehörigen Codesegmente aber nur ein einziges Mal zu laden. Die Datensegmente hingegen werden für jede Instanz (Programmaufruf) neu angelegt. Windows unterscheidet die einzelnen Programmkopien anhand eines Instanzenzählers, der mit jedem Programm-Modul mitgeführt wird.

Zusätzlich sind Speicheradressen in Windows nicht festgelegt. Vielmehr sind die einzelnen Code- und Datensegmente verschiebbar (moveable). Dieses Vorgehen hat den Hintergrund, dass die Windows-Programmierer damit erreichen wollten, dass sehr große Programme auch mit sehr wenig Speicher ausführbar sein sollten. Der resultierende Mechanismus ist der sogenannte „virtuelle Speicher“ (Virtual Memory[1]). Immer wenn nötig (bei Platzbedarf) verschiebt Windows ein als „moveable“ gekennzeichnetes Segment, welches gerade nicht benötigt wird, in den virtuellen Speicher (auf die Festplatte). Wird ein Teil dieses Segmentes angesprochen, so lädt das Betriebssystem das Segment wieder in den Hauptspeicher, zumeist im Austausch gegen ein anderes Segment. Dabei kann man natürlich nicht davon ausgehen, dass das Segment sich anschließend noch an der gleichen Stelle befindet. entsprechend Schwierig gestalten sich natürlich die Speicherzugriffe. Glücklicherweise kümmert sich Windows um diese Problematik, so dass man nur selten auf „fixed“ Segmente (nicht auslagerbare Blöcke) zurückgreifen muss.

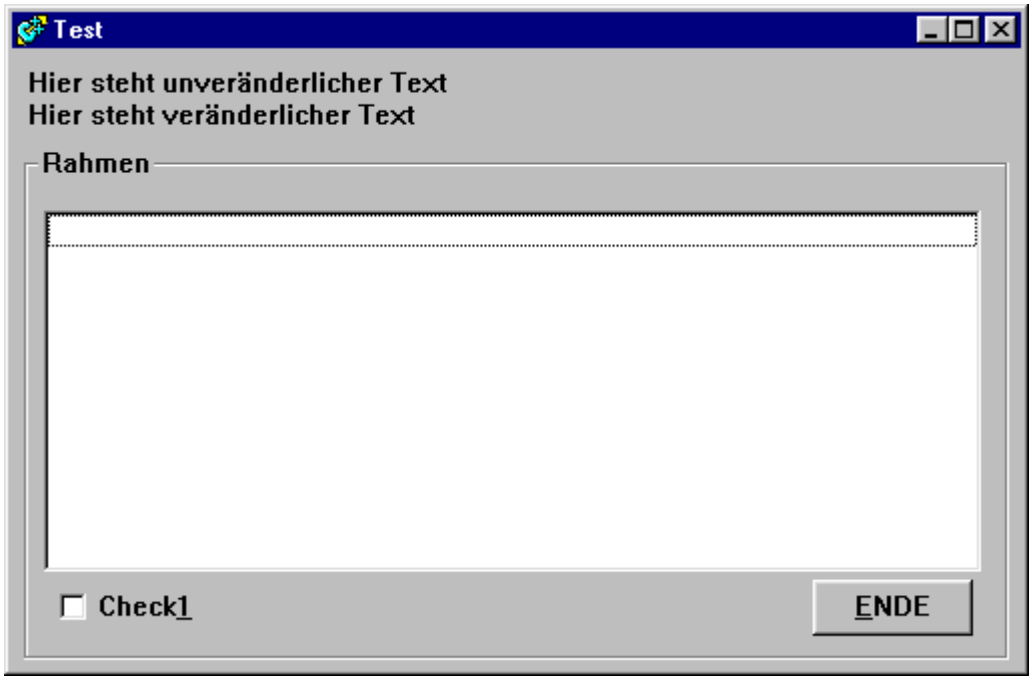
Bei Codesegmenten geht Windows sogar noch weiter, indem es die Codesegmente als „discardable“ kennzeichnet. Anstatt das Codesegment in den virtuellen Speicher zu verschieben, wird das komplette Segment gelöscht und bei Bedarf aus der EXE-Datei nachgeladen.



[1] **virtual** [engl.]: wirklich, tatsächlich, eigentlich, virtuell – in der EDV bezeichnet der Begriff „virtual memory“ die Verwendung von Festplattenbereichen als Erweiterung des Speichers. In Windows auch als „Auslagerungsdatei“ bekannt.
memory [engl.]: Gedächtnis, Speicher

5.2 Anwendungs-Resourcen

Anwendungs-Resourcen sind in Windows eingeführt worden, um die Übersetzung von Windows Programmen in andere Sprachen und somit die weltweite Vermarktung von Anwendungen zu erleichtern. Dabei werden die Fensteroberflächen in einem kodierten Format (an die bereits in Maschinensprache übersetzten Programme) angehängt.
Ziel war es, die bereits fertigen Programme nachträglich (im Aussehen, nicht in der Funktionalität) verändern zu können. Sofern das Aussehen der Dialoge nicht (hart codiert) programmintern verändert wird, genügt es die angehängten Teile z.B. gegen fremdsprachige Dialoge auszutauschen. Üblicherweise verwendet man dazu sogenannte Resource-Editoren, die für jeder gewünschte Control eine Identifikationsnummer (Resource-ID) erzeugen und in einem Headerfile abspeichern (meist mit der Datei-Endung ***rh*** oder ***rch***). Die Definition der gewünschten Controls wird als spezielles Textformat in einer Resourcedatei gespeichert (meist mit der Datei-Endung ***rc***).
Diese Resourcedatei (gegebenenfalls auch mehrere) nimmt man in das Projekt mit auf. Anhand der Datei-Endung erkennt der Compiler, dass es sich um eine Resourcedatei handeln soll und ruft den Resource-Compiler auf, der die Textdatei in ein spezielles Format übersetzt (die Ergebnisdatei trägt meist die Datei-Endung ***res***) – dieser Vorgang ist vergleichbar mit einem normalen Compiler-Lauf, der aus einem Quelltext (***cpp***) eine Objektdatei (***obj***) macht.
Die compilierte Resourcedatei wird beim Linking an das fertige Programm angehängt und die einzelnen Control-Elemente werden vom Programm aus allein über ihre ID-Nummer angesprochen. Da es für jedes Control einen festen Aufbau der Eigenschaften gibt (vergleichbar einem Datensatz), kann man ein Control (ein geeignetes Werkzeug vorausgesetzt) nachträglich, d.h. im fertigen Programm, noch verändern – z.B. eine Listbox vergrößern oder Beschriftungen übersetzen. Dies funktioniert natürlich nur, wenn die Codenummern der dort abgelegten Dialogelemente unverändert erhalten bleiben, denn über diese Nummern werden die einzelnen Controls angesprochen und verarbeitet. Beschreibende Control-Elemente (wie z.B. Buttonbeschriftungen) können aber problemlos ausgetauscht werden.
Die folgenden Listings zeigen die notwendigen Einträge, um die unten abgebildete Dialogbox als Resourcen abzulegen.




```
/*********************************************************
test.rh
produced by Borland Resource Workshop
*********************************************************/
#define IDD_TEST     101
#define LB_LISTBOX1  102
#define CK_CHECK1    103
#define B_ENDE       104
#define ST_TEXT1     105
#define ID_GROUPBOX1 106
```

```
/*********************************************************
test.rc
produced by Borland Resource Workshop
*********************************************************/

#include "test.rh"

IDD_TEST DIALOG 0, 0, 252, 154
         STYLE WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU |
               WS_MINIMIZEBOX | WS_MAXIMIZEBOX | WS_THICKFRAME
               CAPTION "Test"
{
   CONTROL "Rahmen", ID_GROUPBOX1, "BUTTON",
           BS_GROUPBOX | WS_CHILD | WS_VISIBLE,
           3, 24, 246, 128
   CONTROL "ListBox1", LB_LISTBOX1, "listbox",
           LBS_STANDARD | LBS_HASSTRINGS | LBS_USETABSTOPS |
           WS_HSCROLL | WS_TABSTOP,
           8, 40, 234, 96
   CONTROL "Check&1", CK_CHECK1, "BUTTON",
           BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE |
           WS_TABSTOP, 12, 133, 40, 12
   CONTROL "&ENDE", B_ENDE, "BUTTON",
           BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP,
           200, 132, 40, 14
   CONTROL "Hier steht unveränderlicher Text", -1, "STATIC",
           SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP,
           4, 4, 240, 8
   CONTROL "Hier steht veränderlicher Text", ST_TEXT1,
```

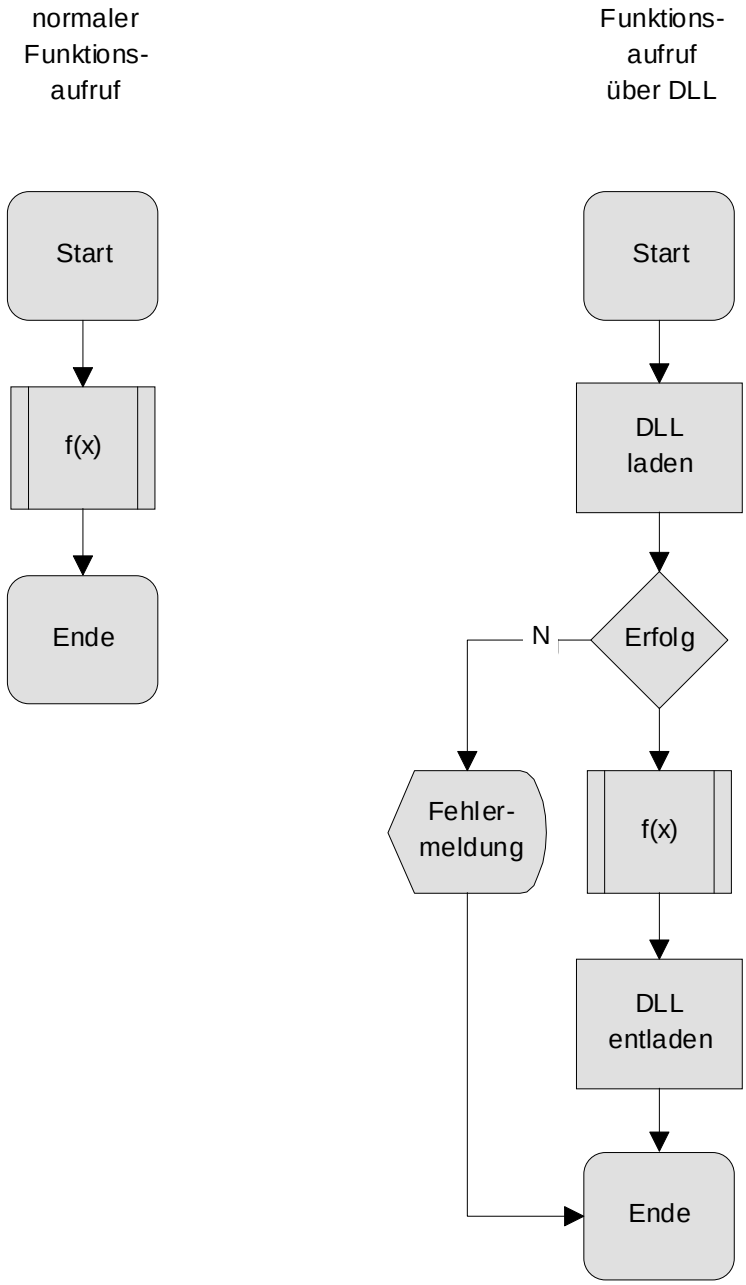
```
            "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE |
            WS_GROUP, 4, 12, 244, 8
}
```

5.3 Dynamic Link Libraries (DLLs)

Eine DLL ist eine Funktionssammlung, die nicht fest in ein Programm integriert ist. Statt dessen wird die Funktionssammlung erst bei Bedarf geöffnet. Dieses Konzept hat gleich zwei Vorteile:

- Die Belastung des Hauptspeichers geringer, da die Programmcodes erst dann geladen werden wenn sie gebraucht werden und nach Gebrauch sofort wieder entladen werden können.
- Die DLL kann, wenn Fehler entdeckt und entfernt wurden, ausgetauscht werden, ohne dass das aufrufende Hauptprogramm neu übersetzt werden muss.

Um mit DLL-Techniken zu arbeiten müssen natürlich trotzdem einige Spielregeln eingehalten werden. So kann z.B. eine DLL nur solange ausgetauscht werden, solange sich die Parameterleisten der darin enthaltenen Funktionen nicht ändern.
Zudem erfordert die Verwendung von DLLs natürlich auch mehr Sorgfalt bei der Programmierung, denn vor Nutzung einer Funktion die in einer DLL gespeichert ist, muss der Programmierer dafür sorgen, dass die entsprechende DLL geladen und später wieder entfernt wird (siehe Graphik).



5.4 API

API ist die Abkürzung für „Application Programmers Interface“. Ein API ist nichts weiter als eine Sammlung von Programmfunktionen (in C/C++ würde man von einer Funktions-Bibliothek sprechen). Das wichtigste API ist das Windows-API, welches alle Grundfunktionen von Windows zur Verfügung stellt. Das Windows-API umfasst das Windows-Kernel, sowie alle Erweiterungen (das sind üblicherweise die DLLs aus dem Windows-System-Verzeichnis).

6 Ein erstes Projekt

Im Folgenden soll schrittweise ein Programm zur Umrechnung von Maßeinheiten entwickelt werden.

6.1 Schritt 1 – Erzeugen des Hauptfensters

Schreiben Sie dazu zunächst ein Rumpfprogramm, das ein Hauptfenster (Client-Area) öffnet, wie in Abschnitt 3.7 beschrieben.
Das Ergebnis sollte dann etwa so aussehen:

Probieren Sie ruhig mal aus, was passiert, wenn Sie auf die Schalter links und rechts in der Titelzeile klicken, bzw. einige der Style-Parameter im Hauptprogramm ändern. Die hier einbezogenen Funktionalitäten sind Windows Standard und werden, sofern man sie nicht ausdrücklich entfernt, automatisch von hinzugefügt.

6.2 Schritt 2 – Ausgabe im Hauptfenster

Erweitern Sie das Rumpfprogramm um einen zentrierten Text „Umrechnung“ auf der Client-Area. Dazu benötigt man die Funktionen **GetClientRect**[∇] und **DrawText**[∇]. (Nummern aufgetretener Fehler dieser und vieler anderer Funktionen können über **GetLastError**[∇] ermittelt werden).
Diese sind im **WM_PAINT** Bereich einzufügen, da sie auf der Client-Area zeichnen.
Um später, wenn Dutzende von Messages in der **WndProc** verarbeitet werden müssen nicht den Überblick zu verlieren, macht es Sinn, nach jedem **case** der **switch** Anweisung über **nMsg** eine eigene Funktion aufzurufen, die man nach der Message benennt. Im Falle von **WM_Paint** also die Funktion **fnWMPaint (...):**

```
#pragma argsused
LRESULT fnWMPaint (HWND, WPARAM, LPARAM)
{
}
```

[∇] Siehe Syntaxlisting

Die Funktion **GetClientRect** liest die aktuelle Größe der Client-Area in eine bereitzustellende Rechteck-Struktur ein.
Da der Anwender die Fenstergröße nahezu beliebig ändern kann, ist dies notwendig um die gerade eingestellte Größe zu ermitteln. Nach jeder Größenänderung des Hauptfensters wird **WM_PAINT** automatisch neu aufgerufen.
Der Rechteckbereich der von **GetClientRect** zurückgegeben wird, ist der innere Fensterbereich (Ohne Titelzeile und Rand!). Das Ergebnis wird in die bereitzustellende Rechteck-Struktur eingetragen
Da die Funktion **GetClientRect** den Device-Context nicht benötigt, kann der Aufruf vor der **BeginPaint**$^{\nabla}$ Anweisung erfolgen, was den Zeitraum des exklusiven Bildschirmzugriffs reduziert.

```
#pragma argsused
LRESULT fnWMPaint (HWND, WPARAM, LPARAM)
{
   RECT aRect;
   GetClientRect (hWnd, &aRect);
}
```

Nach der Ermittlung des Rechteckes der Client-Area kann man in diesem Rechteck einen Text ausgeben:

```
DrawText (hDC, " Umrechnung ", -1, &aRect,
          DT_SINGLELINE | DT_CENTER | DT_VCENTER);
```

Allerdings benötigt die Funktion **DrawText** den Device-Context des Bildschirms (**hDC**). Dieser wird erst von der Funktion **BeginPaint** ermittelt, so dass der **DrawText** Aufruf erst nach diesem stehen kann, also in den Bereich des exklusiven Zugriffs gehört (was auch nur logisch erscheint, da **DrawText** auf den Bildschirm zeichnet).

```
hDC = BeginPaint (hWnd, &aPS);
DrawText (hDC, " Umrechnung ", -1, &aRect,
          DT_SINGLELINE | DT_CENTER | DT_VCENTER);
EndPaint (hWnd, &aPS);
```

Drawtext schreibt einen Text in das Fenster, welches mit **hDC** verbunden ist (hier die Client-Area).

Schalter	**Bedeutung**
DT_SINGLELINE	Einzeilige Ausgabe Zeilenvorschubzeichen im Text ("\n") werden nicht beachtet.
DT_BOTTOM	Text am unteren Randbereich des Rechtecks ausgeben. Nur zusammen mit DT_SINGLELINE möglich.
DT_CALCRECT	Berechnet des benötigten Platz bei mehrzeiligen Ausgaben selbstständig.
DT_CENTER	Zentriert den Text in der horizontalen Ebene.
DT_EXPANDTABS	Erweitert Tabulatorzeichen im Text ("\t") auf die Länge von acht Leerzeichen.
DT_EXTERNALLEADING	Berechnet typographische Unter- und Überlängen in den zur Ausgabe benötigten Platz mit ein.
DT_LEFT	Linksbündige Ausgabe im angegebenen Rechteck
DT_NOCLIP	Ausgabe ohne Berücksichtigung von verdeckten

Schalter	Bedeutung
	Bereichen. Ist schneller, macht aber ggf. Probleme mit überlagernden Unterfenstern (Child Windows).
DT_NOPREFIX	Normalerweise wird ein im Text befindliches "&"-Zeichen als Unterstreichung für das folgende Zeichen gewertet und ein "&&" als Ausgabe des &-Zeichens Diese Methodik kann über diesen Stil abgeschaltet werden.
DT_RIGHT	Rechtsbündige Ausgabe im angegebenen Rechteck.
DT_TABSTOP	Setzt einen Tabulatorstop.
DT_TOP	Am oberen Randbereich des Rechtecks ausgeben. Nur zusammen mit DT_SINGLELINE möglich.
DT_VCENTER	Zentriert den Text in der vertikalen Ebene. Nur zusammen mit DT_SINGLELINE möglich.
DT_WORDBREAK	Gibt an, dass bei mehrzeiliger Ausgabe nur an ganzen Worte umgebrochen werden soll.

EndPaint$^{\nabla}$ schließlich beendet den exklusiven Zugriff auf den Bildschirm. Als Gesamtfunktion ergibt sich nun:

```
#pragma argsused
LRESULT fnWMPaint (HWND hWnd, WPARAM wParam, LPARAM lParam)
{
   HDC         hDC ;
   PAINTSTRUCT aPS ;
   RECT        aRect ;

   GetClientRect (hWnd, &aRect);
   hDC = BeginPaint (hWnd, &aPS);
   DrawText (hDC, "Umrechnung", -1, &aRect,
             DT_SINGLELINE | DT_CENTER | DT_VCENTER);
   EndPaint (hWnd, &aPS);
   return 0;
}
```

In gleicher Weise wird mit der Message **WM_DESTROY** verfahren:

```
#pragma argsused
LRESULT fnWMDestroy (HWND hWnd, WPARAM wParam, LPARAM lParam)
{
   PostQuitMessage (0);
   return 0;
}
```

Die vereinfachte **WndProc** Funktion hat nun dieses Aussehen:

```
#pragma argsused
LRESULT CALLBACK WndProc (HWND hWnd, UINT nMsg, WPARAM wParam,
LPARAM lParam)
{
   switch (nMsg)
   {
     case WM_PAINT:   return fnWMPaint   (hWnd,wParam,lParam);
     case WM_DESTROY: return fnWMDestroy (hWnd,wParam,lParam);
   }
   return DefWindowProc (hWnd, nMsg, wParam, lParam);
}
```

Beachten Sie bitte, dass sich, sofern Sie die Größe des Fensters ändern, der Text immer wieder automatisch in der Fenstermitte zentriert.
Das Ergebnis sollte dann etwa so aussehen:

Nehmen Sie sich jetzt die Zeit mal auszuprobieren, was passiert, wenn Sie die Parameter in der DrawText-Funktion ändern und DT_xxx Schalter hinzufügen oder entfernen.

6.3 Schritt 3 – Zeitanzeige

Da eine Überschrift in der Mitte des Bildschirms wenig Sinn macht, verlagern Sie diese bitte an den oberen Rand.
Außerdem soll das Hauptfenster um eine kleine Serviceanzeige erweitert werden, indem eine Anzeige der aktuellen Uhrzeit am unteren Bildschirmrand hinzugefügt wird.

Um dies zu erreichen, muss Windows angewiesen werden, das Programm in regelmäßigen Abständen automatisch zu benachrichtigen.
Dazu verwendet man den Befehl **SetTimer**$^{\nabla}$, der bei Windows den Wunsch anmeldet, dass in bestimmten Zeitintervallen eine genau definierte Message geschickt werden soll. Dies ist ggf. sogar unabhängig davon, ob das Programm gerade den Eingabefokus hat oder nicht. Die von **SetTimer** benötigen Parameter sind im Syntax-Anhang aufgeführt.
Im Hauptprogramm **WinMain** ist dazu zu ergänzen:

```
if (!SetTimer (hWindow, ID_TIMER, 500, NULL))
{
   MessageBox (hWindow, "Zuviele Timer!", sWindowclassname,
               MB_ICONEXCLAMATION | MB_OK);
   return 0;
}
```

Die symbolische Konstante **ID_TIMER** wird benötigt, um mehrere Timer unterscheiden zu können. **ID_TIMER** kann einfach mit **#define** bestimmt werden.

```
#define ID_TIMER 10000
```

Weil die Anzahl der Timer im Windows beschränkt ist (es handelt sich um eine sogenannte *limited Resource*), sollte man Timer mit bedacht einsetzen.
Zudem werden Timer am Ende des Programms nicht automatisch wieder freigegeben, sondern müssen ausdrücklich beendet werden.
Der letztmögliche Zeitpunkt ist zudem der günstigste, dies ist genau dann, wenn **PostQuitMessage**$^{\nabla}$ aufgerufen wird, da danach das Programm ohnehin beendet wird.

```
#pragma argsused
LRESULT fnWMDestroy (HWND hWnd, WPARAM wParam, LPARAM lParam)
{
   PostQuitMessage (0);
   KillTimer (hWnd, ID_TIMER);
   return 0;
}
```

Nachdem nun dafür gesorgt ist, dass der Timer eingerichtet und wieder vernichtet wird, muss man noch programmieren, was zu tun ist, wenn ein Timer-Ereignis eintritt und Windows dem Programm eine Timer-Message schickt.
Genau wie bei die **WM_PAINT** und die **WM_DESTROY** Messages läuft auch die **WM_TIMER** Message in der mit der Client-Area verbundenen Handlerfunction **WndProc** auf. Entsprechend wird auch diese behandelt, indem man eine **fnWMTimer** Funktion erstellt und den **switch** und **WndProc** erweitert.

```
case WM_TIMER:   return fnWMTimer   (hWnd, wParam, lParam);
```

```
#pragma argsused
LRESULT fnWMTimer (HWND hWnd, WPARAM wParam, LPARAM lParam)
{
}
```

Entscheidend ist nun, dass aufgrund des Timers eine Textausgabe auf dem Bildschirm erzeugt werden soll. Die Textausgabe ist aber nur innerhalb des **WM_PAINT** Ereignisses möglich. Es muss also dafür gesorgt werden, dass Windows es als notwendig erachtet, die Client-Area neu zu zeichnen.
Natürlich könnte man an dieser Stelle die Größe der Client-Area ändern, um dies zu erreichen, das wäre aber relativ unpraktisch.
Statt dessen erklärt man kurzerhand den Teil der Client-Area, welcher sich ändern soll für ungültig (invalid) und zwingt Windows dergestalt, die eine **WM_PAINT** Message zu senden, um den ungültigen Bereich über einen Bildschirmupdate zu validieren.
Zur Invalidierung dient die Funktion **InvalidateRect**$^{\nabla}$.

```
InvalidateRect (hWnd, NULL, TRUE);
```

Um das ganze noch zu untermalen geben wir bei jedem **WM_TIMER** Ereignis noch einen **MessageBeep**$^{\nabla}$ mit aus, der uns akustisch davon unterrichtet, dass der Timer aufgerufen wurde:

```
#pragma argsused
LRESULT fnWMTimer (HWND hWnd, WPARAM wParam, LPARAM lParam)
{
   MessageBeep (0);
   InvalidateRect (hWnd, NULL, TRUE);
   return 0
}
```

Als letztes gilt es nun noch, in der Behandlungsfunktion des **WM_PAINT** Ereignisses die Uhrzeit zu ermitteln und mit **DrawText** auszugeben.
Dazu wird die Standard-C-Funktion **gettime** benötigt.

```
char sTime [10] = "";
struct time aTime;

gettime (&aTime);
sprintf (sTime, "%02d:%02d.%02d", aTime.ti_hour, aTime.ti_min,
         aTime.ti_sec);
```

Als zusätzliche Headerdateien benötigen Sie zudem **dos.h** (für **gettime** und **struct time**) und **stdio.h** (für **sprintf**).
Das Ergebnis sollte dann etwa so aussehen:

Probieren Sie ruhig mal aus, was passiert, wenn Sie InvalidateRect mit FALSE im dritten Parameter aufrufen und in der Ausgabeformatierung über sprintf keine Ausgabelängen definieren: "%d:%d.%d"

6.4 Schritt 4 – Ausgabe der Nutzdaten (I)

Um unsere Mitmenschen nicht an den Rand des Wahnsinns zu treiben, entfernen Sie bitte wieder den **MessageBeep** in der Funktion, die auf den Timer reagiert.
Als nächstes soll das Programm ein wenig sinnvolle Funktionalität bekommen. Da es ein Umrechnungsprogramm für Maßeinheiten

werden soll, muss auch eine Funktion bereitgestellt werden, die dieser Umrechnung durchführt.

Schreiben Sie eine Funktion, die beliebige Umrechnungen für Maßeinheiten durchführen kann. Je nach umzurechnender Maßeinheit müssen Sie mit einem Faktor multiplizieren oder durch diesen dividieren (z.B. Umrechnung von cm in Zoll). Gelegentlich kann es auch vorkommen, dass ein Offset hinzu- oder abgerechnet werden muss (z.B. Umrechnung von Celsius in Fahrenheit). Die Funktion sollte dies berücksichtigen.

Die Deklaration der Funktion sollte in etwa wie folgt aussehen, wobei wir die Verwendung von Defaultparametern dahingehend ausnutzen den Offset nicht jedes Mal mit angeben zu müssen:

```
double fnCalcValue (double fValue, double fFaktor=1.0,
                    bool bMultiply=true, double fOffset=0.0,
                    bool bOffset=false, bool bOffsetFirst=true);
```

Ebenso fehlt noch eine Funktion, welche die Umrechnungstabelle erzeugt, also in einer Schleife die Umrechnungsfunktion aufruft und als Text auf der Client-Area ausgibt:

```
void fnValueTable (HDC hDC);
```

Die bisher zur Textausgabe verwendete Funktion **DrawText** ist für diesen Zweck leider ein wenig umständlich, da man nun lauter Rechtecke berechnen müsste, um die Schrift darin zu platzieren.
Statt dessen ist es besser die etwas einfachere Funktion **TextOut**$^{\nabla}$ zu verwenden.
TextOut schreibt an eine mit **x-** und **y-Koordinate** vorgegebene Bildschirmposition. dazu benötigt die Funktion die Angabe der folgenden Parameter:

```
BOOL TextOut (HDC hDC, int x, int y, LPCTSTR sText, int nLen);
```

Die Funktion gibt einen Wert ungleich FALSE zurück, wenn alles korrekt verlaufen ist. Da **TextOut** den Device-Context benötigt, wird auch klar, warum dieser and die Funktion **fnValueTable** zu übergeben ist. In **nLen** ist die Länge des auszugebenden Textes anzugeben, dieser lässt sich üblicherweise leicht mit **strlen (sText)** ermitteln.
Ein erster Ansatz für die Tabellenausgabe (hier eine Umrechnung von Zoll in Zentimeter) sieht in etwa wie folgt aus:

```
void fnValueTable (HDC hDC)
{
   int i;
   double f;
   char sText [100];

   for (i=0, f=0.0; i<20; f+=0.5, i++)
   {
      sprintf (sText, "%10.2lf Zoll = %10.2lf cm",
```

```
                f, fnCalcValue(f, 2.56));
        TextOut (hDC, 10, i*20, sText, strlen (sText));
    }
}
```

Das Ergebnis sollte dann etwa so aussehen:

6.5 Schritt 5 – Berechnung der Schriftgrößen

Das bisherige Ergebnis ist funktional schon ganz in Ordnung, hat aber noch in der Darstellung einige zu behebende Mängel. So stehen die umgerechneten Werte noch in relativ weitem Abstand und beginnen zudem in der ersten Zeile, so dass bei mehrspaltiger Ausgabe die Überschrift überschrieben würde. Unschön, wenn auch nicht wirklich falsch ist die Tatsache, dass die Spalten nicht sauber rechtsbündig ausgerichtet sind.

Der Zeilenabstand ließe sich sicherlich durch ein wenig Ausprobieren ändern, dies hätte aber den Nachteil, dass davon ausgegangen wird, dass sich die Standardschriftart von Windows nicht ändert.
Statt dessen besteht der korrekt Weg darin, die Größe der Standardschriftart zur Programmlaufzeit zu ermitteln und alle anderen Gegebenheiten darauf anzupassen.
Hierfür bietet Windows die Funktion **GetTextMetrics**$^{\nabla}$, die alle wichtigen Informationen des eingestellten Standardschrifttyps eines Fensters liefert.

```
TEXTMETRIC   aTM;
GetTextMetrics (hDC, &aTM) ;
```

Wie man sieht, benötigt die Funktion den Device-Context des angeschlossenen Gerätes (in diesem Fall des Monitors), denn eine Textzeile auf dem Bildschirm hat logischerweise eine andere Auflösung als eine Textzeile auf einem Drucker.
Die einzige Stelle, an der das Programm bisher den Device-Context kennt, ist im Verlauf der **WM_PAINT** Ereignisbehandlung.
Da die Größe der Standardschriftart von Windows sich aber nur höchst selten ändern wird, ist diese Stelle denkbar ungeeignet, denn dann

würde die Größenberechnung für die Tabelle ja bei jedem Neuzeichnen erneut vorgenommen (zur Zeit – durch den Timer – jede Sekunde).

Weil die Änderung der Standardschriftart so selten ist, reicht es aus, die Schriftart einmal, am Beginn des Programms auswerten zu lassen. Dafür bietet sich in der Ereignisbehandlung von Windows die Message **WM_CREATE** an, die ausgelöst wird, nachdem das Fenster erzeugt wurde.
WM_CREATE wird genau wie z.B. **WM_PAINT** an die **WndProc** Funktion des Programms geschickt und kann dort (am Besten wiederum durch Aufruf einer eigenen Funktion) behandelt werden:

```
case WM_CREATE: return fnWMCreate (hWnd, wParam, lParam);
```

Die in **fnWMCreate** benötigte **TEXTMETRIC** Struktur legt man einfach global an, damit auf die darin enthaltenen Werte aus unterschiedlichen Funktionen zugegriffen werden kann.
Dazu muss zunächst der Device-Context mit der Funktion **GetDC**$^{\nabla}$ ermittelt werden. Dies ist problemlos möglich, da **GetDC** lediglich das Fensterhandle benötigt, welches nach der Erzeugung des Fensters vorliegt (daher ist **WM_CREATE** auch die früheste Stelle im Programm, an welcher die **TEXTMETRIC** Werte vom Betriebssystem abgegriffen werden können.
Danach lässt man die Werte ermitteln und gibt anschließend den Device-Context wieder frei.
Man darf auf keinen Fall vergessen, den Device-Context (DC) mit **ReleaseDC**$^{\nabla}$ wieder freigeben, denn der DC ist eine pro Gerät limitierte Resource (d.h. es kann nur einen pro Gerät, in diesem Fall also der Bildschirm, geben

```
LRESULT fnWMCreate (HWND hWnd, WPARAM wParam, LPARAM lParam)
{
   HDC hDC = GetDC (hWnd) ;
   GetTextMetrics (hDC, &aTM) ;
   ReleaseDC (hWnd, hDC) ;
   return 0;
}
```

Die **TEXTMETRIC** Struktur enthält eine ganze Reihe von interessanten Werten, so u.a. die Standardbreite eines Kleinbuchstaben in der Komponente **tmAveCharWidth** (dieser entspricht üblicherweise der Breite des Zeichens 'x').
Sie enthält auch Informationen darüber, ob es sich beim Standardfont um eine Proportionalschrift oder einen Font mit gleichbleibender Breite handelt. Um dies festzustellen, kann man testen, ob an der Strukturkomponente **tmPitchAndFamily** das Bit **TMPF_FIXED_PITCH** gesetzt ist. Wenn dies der Fall ist, handelt es sich um eine Schrift gleichbleibender Breite (Fixed Font). D.h. alle Zeichen sind gleich breit. Bei Proportionalfonts hingegen sind die Großbuchstaben in etwa 1,5 mal so breit wie ein einfacher Buchstabe.
Allgemeingültig kann die Breite eines großen Buchstabens wie folgt berechnet werden:

```
nBreite = (aTM.tmPitchAndFamily & TMPF_FIXED_PITCH ? 3 : 2) *
          aTM.tmAveCharWidth / 2;
```

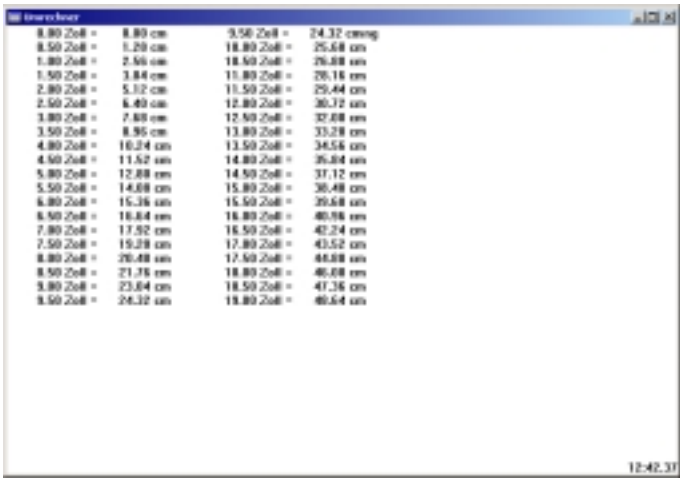
Die Komponente **tmHeight** enthält die Höhe eines Zeichens inklusive der Unterlängen und die Komponente **tmExternalLeading** beschreibt den Platz, der zwischen zwei Zeilen benötigt wird, damit die Buchstaben nicht aneinander "kleben".

```
nZeile = aTM.tmHeight + aTM.tmExternalLeading;
```

So gewappnet kann die Funktion **fnValueTable** dergestalt angepasst werden, dass sie den zur Verfügung stehenden Platz optimaler ausnutzt:

```
void fnValueTable (HDC hDC)
{
   int nLine, nColumn, nLeftSpace = 10;
   double f;
   char sText [100]    = "";
   int nSmallCharWidth = aTM.tmAveCharWidth;
   int nBigCharWidth   = (aTM.tmPitchAndFamily &
                          TMPF_FIXED_PITCH ? 3 : 2) *
                          aTM.tmAveCharWidth / 2;
   int nCharHeight     = aTM.tmHeight + aTM.tmExternalLeading;

   nLeftSpace = nLeftSpace + strlen (sText) * nSmallCharWidth;

   for (nLine=0, f=0.0; nLine<20; f+=0.5, nLine++)
   {
      sprintf (sText, "%10.2lf Zoll = %10.2lf cm",
               f, fnCalcValue(f, 2.56));
      TextOut (hDC, nLeftSpace, nLine*nCharHeight, sText,
               strlen (sText));
   }

   nLeftSpace = nLeftSpace + strlen (sText) * nSmallCharWidth;

   for (nLine=0, f=9.5; nLine<20; f+=0.5, nLine++)
   {
      sprintf (sText, "%10.2lf Zoll = %10.2lf cm",
               f, fnCalcValue(f, 2.56));
      TextOut (hDC, nLeftSpace, nLine*nCharHeight, sText,
               strlen (sText));
   }
}
```

Damit die Tabellenüberschrift nicht von den Werten überschrieben wird, muss die Ausgabe um zwei Zeilen nach unten versetzt werden:
Das Ergebnis sollte dann etwa so aussehen:



Probieren Sie aus, was passiert, wenn Sie das Fenster so weit verkleinern, dass die Tabelle nicht mehr vollständig angezeigt wird.

6.6 Schritt 6 – Platzoptimierung bei der Ausgabe

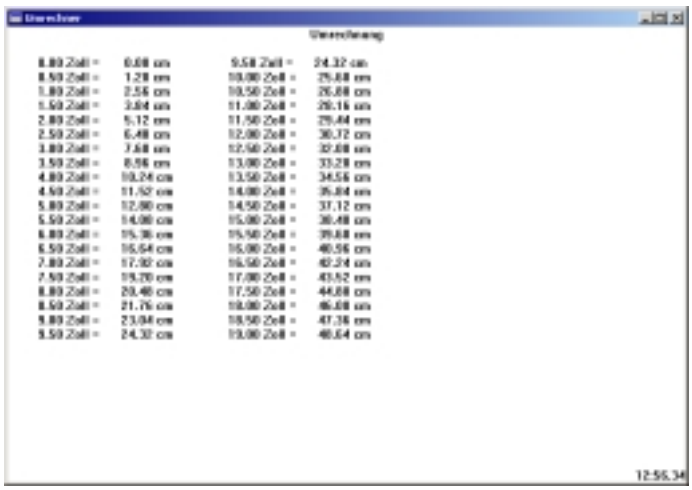
Diese bisher erstellte Programmversion hat immer noch einige Nachteile:

- Die Überschrift wird überschrieben
- Die Höhe wird noch nicht optimal genutzt
- Die Breite wird noch nicht genutzt.
- Die Ausgabe ist nicht rechtsbündig.

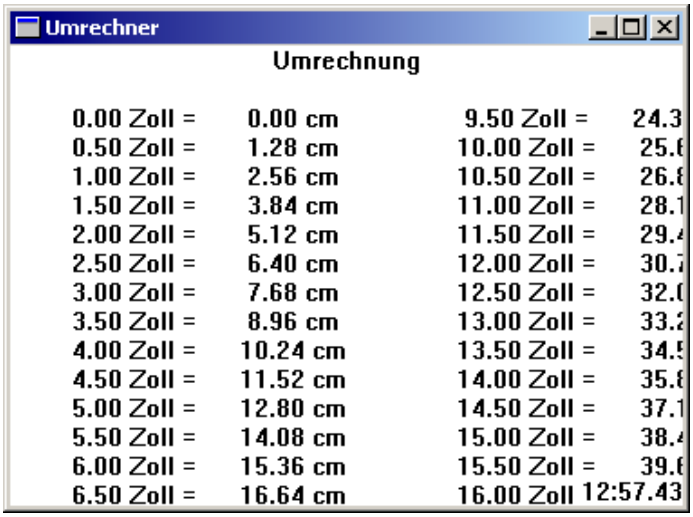
Die Anpassung in der Höhe ist einfach, indem man einfach auf die y-Koordinate zwei Zeilen hinzuaddiert:

```
TextOut (hDC, nLeftSpace, (nLine+2)*nCharHeight, sText,
                strlen (sText));
```

Damit wird immerhin verhindert, dass die Überschrift verdeckt wird:



Das Problem bei der Verkleinerung des Fensters wird dadurch natürlich noch nicht beseitigt:



Für eine optimale Platzausnutzung muss demnach wenig mehr Aufwand betrieben werden – man muss die Größe Ausgabefensters berücksichtigen, die dazu zu verwendende Funktion **GetClientRect** ist bereits bekannt.
Wir erhalten die Anzahl der darstellbaren Zeilen, indem wir die Höhe des Fensters durch die Zeilenhöhe teilen.

```
   RECT aRect;
   GetClientRect (hWnd, &aRect);
   nAnzahlZeilen = aRect.bottom / nCharHeight - 4;
```

Die Angabe von –4 zieht jeweils zwei Zeilen für die Überschrift und die Ausgabe der Uhrzeit im unteren Bereich ab.
Würde man diese Zeilen in die Funktion **fnValueTable** einfügen, so müsste man die Parameterleiste um das Handle **hWnd** erweitern.
Dies ist aber unnötig, denn so oft muss die Anzahl der Zeilen ja nicht berechnet werden, sie bleibt schließlich konstant, solange die Größe des Fensters nicht verändert wird. Es wäre daher programmtechnisch wesentlich effizienter, die Berechnung auch nur dann vorzunehmen, wenn sich die Ausdehnung des Fensters ändert.
Eine solche Änderung teilt Windows in Form einer **WM_SIZE** Message mit, auf die wir in bewährter Form reagieren:

```
case WM_SIZE: return fnWMSize (hWnd, wParam, lParam);
```

Die **WM_SIZE** Message enthält u.a. im **wParam** die Information, auf welche Weise die Größenänderung erfolgt ist (z.B. durch den Button „maximieren“ in der Titelzeile). Diese Information ist für das vorliegende Programm ohne Belang.
Im **lParam** hingegen sind sowohl die neue Höhe, wie auch die Breite des Fensters abgelegt. Die Höhe in den oberen 16 Bit (**HIWORD**)des Long-Wertes, die Breite in den unteren 16 Bit (**LOWORD**).
Zur Extraktion stellt Windows entsprechende Abruffunktionen bzw. Makros zur Verfügung:

```
fwSizeType = wParam;          // resizing flag
nWidth     = LOWORD(lParam);  // width of client area
```

```
nHeight    = HIWORD(lParam);  // height of client area
```

Um das ganze zu vereinfachen, speichern wir diese Werte in aus **fnWMSize** heraus in globalen Variablen.
Und da wir schon dabei sind, verlagern wir auch die aus der **TEXTMETRIC** berechneten Zeichengrößen in globale Variablen, die wir in **fnWMCreate** berechnen und zuweisen..

```
int nAnzZeichen, nAnzZeilen;
int nSmallCharWidth, nBigCharWidth, nCharHeight;
```

Da die TEXTMETRIC Struktur an anderer Stelle nicht benötigt wird, kann sie als Variable in die Funktion **fnWMCreate** verlagert werden.

```
LRESULT fnWMCreate (HWND hWnd, WPARAM wParam, LPARAM lParam)
{
   TEXTMETRIC aTM;

   HDC hDC = GetDC (hWnd) ;
   GetTextMetrics (hDC, &aTM) ;
   ReleaseDC (hWnd, hDC) ;

   nCharHeight     = aTM.tmHeight + aTM.tmExternalLeading;
   nSmallCharWidth = aTM.tmAveCharWidth;
   nBigCharWidth   = (aTM.tmPitchAndFamily & TMPF_FIXED_PITCH?
                      3 : 2) * aTM.tmAveCharWidth / 2;
   return 0;
}
```

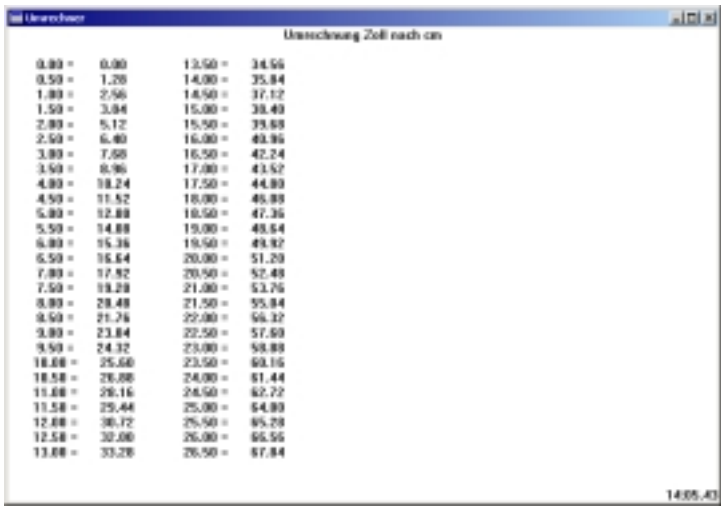
Dadurch, dass die Ermittlung der Zeilen und Zeichen in der Funktion **fnWMSize** erfolgt, kann man sich auch den Aufwand mit der **GetClientRect** Funktion sparen, da die zu ermittelnden Werte bereits im **lParam** überreicht werden:

```
LRESULT fnWMSize (HWND hWnd, WPARAM wParam, LPARAM lParam)
{
  nAnzZeichen = LOWORD(lParam) / nSmallCharWidth - 2;
  nAnzZeilen  = HIWORD(lParam) / nCharHeight - 4;

  return 0;
}
```

Die Bildschirmbreite wird durch **nSmallCharWidth** geteilt, da die Einheiten (derzeit Zoll und cm) eigentlich auch Großbuchstaben enthalten können, später (wenn es mehrere Umrechnungsarten gibt) ggf. sogar nur aus Großbuchstaben bestehen könnten, müsste durch **nBigCharWidth** geteilt werden.
Wir entledigen uns des Problems dadurch, dass wir die Angabe der Einheiten in die Überschrift ziehen und die Tabelle ohne die sich wiederholenden Einheiten angeben. Dies hat zudem den Vorteil, dass mehr Spalten auf den Bildschirm passen.

Nehmen Sie die entsprechenden Anpassungen in der Funktion fnValueTable vor, so dass der vorhandene Platz möglichst optimal zur Darstellung der Umrechnungstabelle genutzt wird

Das erzielte Ergebnis sollte sich in etwa wie folgt darstellen:



Dieses Fenster ist nun auch in der Lage auf Größenveränderungen in der Vertikalen einigermaßen korrekt zu reagieren:

allerdings macht es sich nicht gut, wenn man eine bestimmte Höhe unterschreitet.

Diesem Fall kann man vorbeugen, indem man eine Minimumgröße abfragt und in der **fnWMSize** Funktion die neue Höhe plausibilisiert. Wird die Minimumhöhe unterschritten, so ändert man die Fenstergröße automatisch ab. Dazu dient die Funktion **MoveWindow**$^{\nabla}$, sie erhält das Handle des zu änderndes Fensters, die relative Position des Fensters in übergeordneten Fenster (bei einer Client-Area ist dies der Desktop), die Breite und Höhe, sowie ein Flag, welches besagt, ob die Größenänderung eine **WM_PAINT** Message auslösen soll.

```
MoveWindow(hWnd, 0, 0, nNewWidth, nNewHeight, TRUE);
```

Bei der Größenänderung eines Fensters ist darauf zu achten, dass der Rand bei der **MoveWindow** in der Größe mit einzurechnen ist, die **WM_SIZE** Message aber die innere Größe der Client-Area liefert. Um hier nicht in eine Endlosschleife zu geraten, ist es also nötig, die Größe der Ränder, der ggf. vorhandenen Menüzeile und der Titelzeile mit zu berücksichtigen.
Diese Größen sind Systemweit einheitlich und können mit der Funktion **GetSystemMetrics** ermittelt werden:

```
nBreiteDesRahmens   = GetSystemMetrics (SM_CXFRAME);
nHoeheDesRahmens    = GetSystemMetrics (SM_CYFRAME);
nHoeheDesMenues     = GetSystemMetrics (SM_CYMENU);
nHoeheDerTitelzeile = GetSystemMetrics (SM_CYCAPTION);
```

Die wichtigsten von **GetSystemMetrics** ermittelten Werte sind:

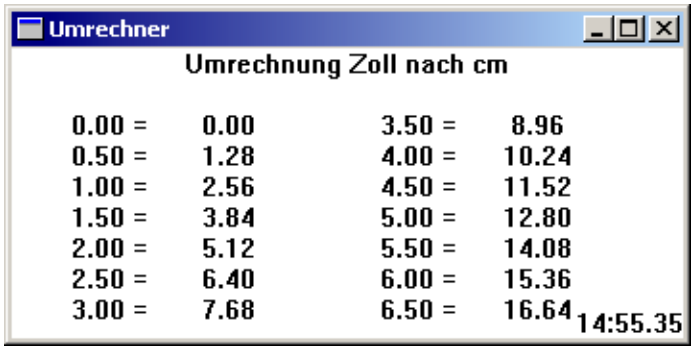
Konstante	Bedeutung
SM_CXSCREEN	Bildschirmbreite
SM_CYSCREEN	Bildschirmhöhe
SM_CXVSCROLL	Breite vertikaler Rollpfeil
SM_CYHSCROLL	Höhe horizontaler Rollpfeil
SM_CYCAPTION	Höhe der Titelleiste
SM_CXBORDER	Rahmenbreite
SM_CYBORDER	Rahmenhöhe
SM_CXDLGFRAME	Rahmenbreite von Dialogen
SM_CYDLGFRAME	Rahmenhöhe von Dialogen
SM_CXHTHUMB	Breite der Marke in Scrollbalken
SM_CYVTHUMB	Höhe der Marke in Scrollbalken
SM_CXICON	Breite von Piktogrammen
SM_CYICON	Höhe von Piktogrammen
SM_CXCURSOR	Breite des Cursors
SM_CYCURSOR	Höhe des Cursors
SM_CYMENU	Höhe der Menüleiste
SM_CXFULLSCREEN	Breite vergrößerter Fenster
SM_CYFULLSCREEN	Höhe vergrößerter Fenster
SM_MOUSEPRESENT	Flag „Maus vorhanden“
SM_CYVSCROLL	Höhe vertikaler Rollpfeil
SM_CXHSCROLL	Breite horizontaler Rollpfeil
SM_SWAPBUTTON	Flag „Mausknöpfe vertauscht“
SM_CXMIN	Minimale Fensterbreite
SM_CYMIN	Minimale Fensterhöhe
SM_CXSIZE	Breite der Zoom-Schaltflächen
SM_CYSIZE	Höhe der Zoom-Schaltflächen
SM_CXFRAME	Fensterrahmenbreite
SM_CYFRAME	Fensterrahmenhöhe
SM_CXMINTRACK	Minimalbreite für Verschieben
SM_CYMINTRACK	Minimalhöhe für Verschieben
SM_CXDOUBLECLK	Max. X-Bewegungstoleranz für Doppelklicks
SM_CYDOUBLECLK	Max. Y-Bewegungstoleranz für Doppelklicks
SM_CXICONSPACING	horizontaler Abstand von Piktogrammen
SM_CYICONSPACING	vertikaler Abstand von Piktogrammen

Insgesamt sieht die Funktion dann in etwa wie folgt aus:

```
LRESULT fnWMSize (HWND hWnd, WPARAM wParam, LPARAM lParam)
{
  int nActWidth = LOWORD(lParam);
  int nMinWidth = 50 * nSmallCharWidth;
  int nActHeight= HIWORD(lParam);
  int nMinHeight=  7 * nCharHeight;
  int nNewWidth = nActWidth<nMinWidth ? nMinWidth :nActWidth;
  int nNewHeight= nActHeight<nMinHeight?nMinHeight:nActHeight;

  if ((nActWidth < nNewWidth) || (nActHeight < nNewHeight))
  {
    nNewWidth  = nNewWidth +GetSystemMetrics (SM_CXFRAME) * 2;
    nNewHeight = nNewHeight+GetSystemMetrics (SM_CYFRAME) * 2
                           + GetSystemMetrics (SM_CYMENU)
```

```
                              + GetSystemMetrics (SM_CYCAPTION);
    MoveWindow(hWnd, 0, 0, nNewWidth, nNewHeight, TRUE);
  }

  //-------------------------------------------------------------
  // globale variablen setzten
  //-------------------------------------------------------------
  nAnzZeichen = nNewWidth  / nSmallCharWidth - 2;
  nAnzZeilen  = nNewHeight / nCharHeight     - 4;

  return 0;
}
```

Anschließend ist ein korrektes Verkleinern des Fensters, bis zu einer durch das Programm vorgegebenen, sinnvollen Größe möglich:



Die z.T. noch fehlende Rechtsbündigkeit der Ausgabe kann mit der Funktion **SetTextAlign**▽ erzielt werden, welche die Ausgabe mit **TextOut** entsprechend beeinflusst. Für ein korrektes Ergebnis muss die Ausgabe der Zahlen für cm und Zoll allerdings in zwei getrennten Blöcken erfolgen.

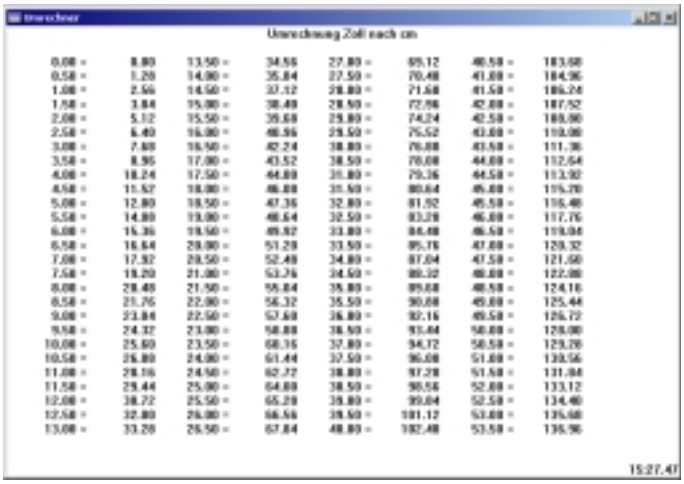
```
unsigned int nOld = SetTextAlign (hDC, TA_RIGHT | TA_TOP);
```

Die wichtigsten in **SetTextAlign** gültigen Schalter sind:

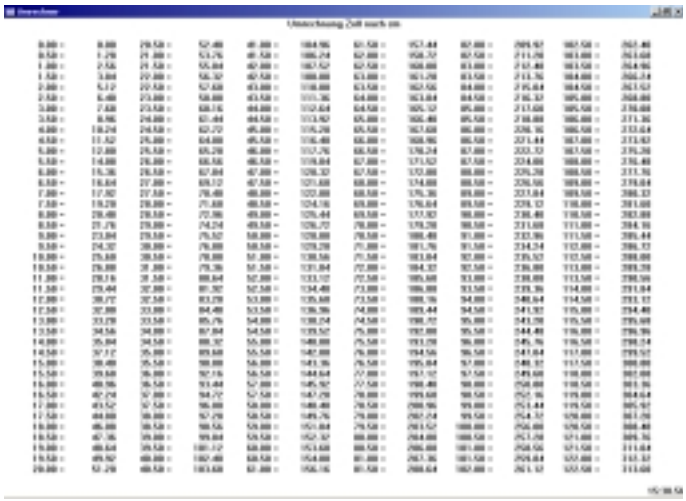
Konstante	Bedeutung
TA_RIGHT	Der angegebene Referenzpunkt (hier x-Koordinate der Funktion **TextOut**), soll als Bezugspunkt der rechten Kante für die Ausgabe betrachtet werden.
TA_LEFT	Der angegebene Referenzpunkt (hier x-Koordinate der Funktion **TextOut**), soll als Bezugspunkt der linken Kante für die Ausgabe betrachtet werden.
TA_TOP	Der angegebene Referenzpunkt (hier y-Koordinate der Funktion **TextOut**), soll als Bezugspunkt der oberen Kante für die Ausgabe betrachtet werden.
TA_BOTTOM	Der angegebene Referenzpunkt (hier y-Koordinate der Funktion **TextOut**), soll als Bezugspunkt der unteren Kante für die Ausgabe betrachtet werden.
TA_ BASELINE	Der angegebene Referenzpunkt (hier x-Koordinate der Funktion **TextOut**), soll als Bezugspunkt der Grundlinie für die Ausgabe betrachtet werden. Dies ist nur Wirksam bei Funktionen, die Texte mit TrueType-Fonts ausgeben, welche definiert abgesetzte Unterlängen beinhalten.
TA_CENTER	Der angegebene Referenzpunkt (hier x-Koordinate der Funktion **TextOut**), soll als Bezugspunkt des Mittelpunkts für die Ausgabe

	betrachtet werden.
TA_UPDATECP	Die aktuelle Position wird nach Aufruf der Ausgabe automatisch aktualisiert. Nur wirksam bei Funktionen, die sich nach der aktuellen Position richten, anstatt eigene Koordinaten als Parameter zu bekommen.
TA_RTLREADING	Formatierung für Schriften, die von Rechts nach Links gelesen werden, wie z.B. Hebräisch oder Arabisch.

Die Funktion gibt den vorher eingestellten Zustand zurück, dieser sollte gesichert und am Ende der Ausgabefunktion wiederhergestellt werden.
Da wir pro Berechnung 23 Zeichen ausgeben und in der Behandlungsfunktion zu **WM_SIZE** berechnet haben, wie viele Zeichen pro Zeile hineinpassen, kann die Anzahl der Spalten berechnet und ausgegeben werden. Der große Vorteil ist, dass sich das Programm bei jeder Größenänderung des Fensters nun automatisch anpasst.
Das nach diesen umfangreichen Umbauarbeiten erzielte Ergebnis hat – in der Standardgröße – etwa das folgende Aussehen:



Nach Klick auf den Button zur Fenstermaximierung:



6.7 Schritt 7 – Updateoptimierung und Verallgemeinerung

Nachdem die Ausgabe hinsichtlich der ausgegebenen Daten optimiert wurde, ist noch das Performance Verhalten des Programms zu verbessern.

Bisher wird jede Sekunde der gesamte Fensterbereich invalidiert und neu gezeichnet, obwohl sich nur die Zeitausgabe unten rechts ändert. Dies macht sich in Form eines gelegentlichen Flackern bemerkbar.
Windows bietet ein automatisches Clipping, welches es ermöglicht, auch nur kleinere Teile des Fensterinhaltes zu ändern, ohne dass sich dies in der Programmierung der **WM_PAINT** Funktion niederschlägt.
Vielmehr gibt man in der **WM_PAINT** Funktion auch weiterhin immer den vollen Fensterinhalt an, Windows selbst entscheidet, ob eine Zeichenoperation ausgeführt werden muss (weil sie den invalidierten Bereich tangiert) oder nicht.
Alles was der Anwendungsentwickler tun muss, ist die Fensterbereiche gezielter zu invalidieren. Im Falle der eingeblendeten Zeit genügt es in der **fnWMTimer** Funktion den rechteckigen Fensterausschnitt zu invalidieren, der von der Zeitanzeige eingenommen wird:

```
LRESULT fnWMTimer (HWND hWnd, WPARAM wParam, LPARAM lParam)
{
  RECT aRect;
  GetClientRect (hWnd, &aRect);
  aRect.left = aRect.right   - nSmallCharWidth * 10;
  aRect.top  = aRect.bottom - nCharHeight;
  InvalidateRect (hWnd, &aRect, TRUE);
  return 0;
}
```

Wenn mehr Umrechnungen realisiert werden können, muss auch die Überschrift flexibler sein. Da die Einheiten später dynamisch, je nach gewählter Umrechnungstabelle hinzugefügt werden sollen, ist es sinnvoll, den Text entsprechend aufzuteilen und erst direkt vor der Ausgabe zusammenzusetzen:

```
sprintf (sUeberschrift, "Umrechnung von %s nach %s",
         sEinheit1, sEinheit2);
aRectTitel = aClientRect;
aRectTitel.bottom = aRectTitel.top + 1.5 * nCharHeight;
```

Die Variablen **sEinheit1** und **sEinheit2** deklarieren und definieren wir der Einfachheit halber zunächst global.

Zudem sollten Überschrift und Zeitausgabe optisch ein wenig von der eigentlichen Ausgabetabelle abgesetzt sein, dazu setzen wir beide jeweils in ein mit hellgrauer Farbe gefülltes Rechteck, das wir vorher in den Ausmaßen definieren müssen:

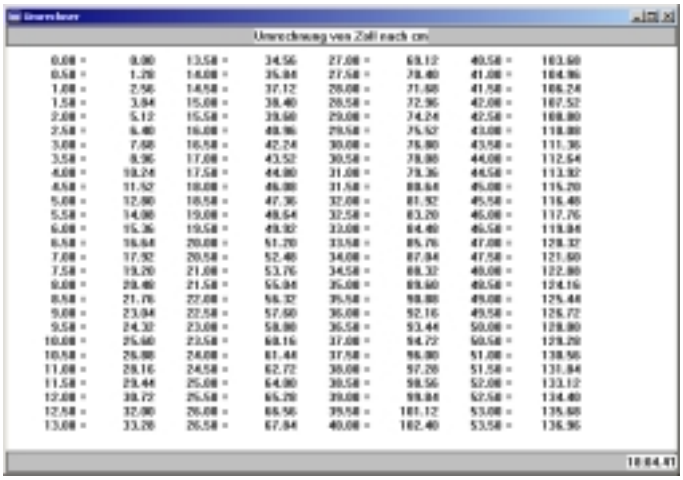
```
FillRect  (hDC, &aRectTitel,
           (HBRUSH)GetStockObject (LTGRAY_BRUSH));
FrameRect (hDC, &aRectTitel,
           (HBRUSH)GetStockObject (BLACK_BRUSH));
DrawText  (hDC, sUeberschrift, -1, &aRectTitel,
           DT_SINGLELINE|DT_CENTER|DT_VCENTER);
```

Die Funktion **FillRect**$^{\nabla}$ füllt das angegebene Rechteck mit einem **Brush** (Pinsel). An dieser Stelle wird ein vordefinierter Brush (Farbe Hellgrau) verwendet, der unter Windows jederzeit verfügbar ist. Der vordefinierte

Brush hat den Vorteil, dass man sich um die Resourcenverwaltung keinerlei Gedanken machen muss. Da **Brush** Objekte (wie Timer) limitierte Resourcen sind, müssen eigene **Brush** Objekte (wie später zu sehen sein wird) mit geringfügig mehr Aufwand verwaltet werden.
Die Funktion **FrameRect**$^{\nabla}$ zeichnet eine einfache Linie um das Rechteck, hier in Schwarz, da der vordefinierte, schwarze Pinsel verwendet wird.
Beide Funktionen verwenden **GetStockObject**$^{\nabla}$ um vordefinierte Elemente vom Windows Betriebssystem abzurufen.

Rufen Sie die Hilfe zur Funktion GetStockObject auf und informieren Sie sich darüber, welche weiteren vordefinierten Objekte es gibt.

Das Ergebnis ist zwar deutlich interessanter, aber nicht zufriedenstellend:



Die Funktion **TextOut** ist leider nicht in der Lage mit durchsichtigem Hintergrund zu zeichnen und überlagert daher sowohl in der Überschrift als auch in der Zeitausgabe die Hintergrundfarbe bzw. auch die Randlinie.

Nehmen Sie die notwendigen Anpassungen in der Funktion fnWMPaint vor, so dass die Überschrift und die Zeitausgabe ein eigenes Ausgaberechteck mit Hintergrundfarbe und Rand bekommen.

6.8 Schritt 8 – Truetype Fonts

Die beste Alternative für die Überschrift ist anstelle der einfachen **TextOut** Funktion einen TrueType Font zu verwenden, der erheblich mehr Gestaltungsspielraum lässt. Nebenbei soll auch gleich der Gebrauch limitierter Resourcen gezeigt werden, indem wir den Grauen Hintergrund der Überschrift durch einen blauen Hintergrund ersetzen.

Für den Hintergrund können wir keinen vordefinierten **Brush** verwenden, da es keinen im gewünschten Farbton gibt.
Mit Hilfe des Farbton-Makros **RGB** und der Funktion **CreateSolidBrush**$^{\nabla}$ ist ein Pinsel jedoch sehr schnell erzeugt.

Als **RGB**-Wert (Rot/Grün/Blau) verwenden wir hier ein Tripel von Zahlen, die vom Makro **RGB** zu einem Wert vom Typ **unsigned long** zusammengefasst werden. Jede der drei additiven Grundfarben kann einen Wert zwischen 0 und 255 annehmen:

```
HBRUSH hBlueBrush = CreateSolidBrush (RGB (100, 100, 255));
```

Die Funktion **CreateSolidBrush** liefert ein Handel auf den im Speicher erzeugten **Brush** zurück. Dieses Handle können wir statt des Aufrufes von **GetStockObject** in der **FillRect** Funktion einsetzen:

```
FillRect (hDC, &aRectTitel, hBlueBrush);
```

Die Funktion **CreateSolidBrush** liefert ein Handel auf das im, Speicher erzeugte Brush-Objekt zurück. Dieses Handle können wir statt des Aufrufes von **GetStockObject** in der **FillRect** Funktion einsetzen:
Abschließend, nach Beendigung des Zeichnens (also nach letzter Benutzung des **Brush** oder nach **EndPaint**) muss die Resource mit **DeleteObject**$^{\nabla}$ wieder freigegeben werden.

```
DeleteObject (hBlueBrush);
```

Auch wenn es auf den ersten Blick so scheinen mag, ist es keine besonders schlaue Idee, alle benötigten Resourcen zu Beginn des Programms anzulegen und bei Beendigung wieder freizugeben. Denn erstens belastet das Programm solange es im Speicher ist die limitierten Resourcen, so dass andere Programm ggf. die benötigten Resourcen nicht erhalten (und schlimmstenfalls abstürzen) und zweitens gehen alle belegten Resourcen verloren, wenn das selbstgeschriebene Programm durch einen Fehler (Absturz) inkorrekt beendet wird.

Geschieht dies nicht, entsteht ein sogenanntes Resource-Leak, es wird bei jedem neuen Aufruf des Ereignisses **WM_PAINT** ein weiterer Brush „verbraucht“. Dies ist einer der häufigsten Fehler in Windowsprogrammen.
Auch alle anderen limitierten Resourcen werden nach diesem Vorbild erzeugt, verwendet und wieder freigegeben.

Doch nun zurück zur Überschrift, die – wie kaum anders zu erwarten – nach gleichem Muster erstellt wird. Auch TrueType Fonts sind limitierte Resourcen, die bei Bedarf erstellt und anschließend wieder freigegeben werden.

```
HFONT hFont = CreateFont (30, 0, 0, 0, FW_THIN,
                          FALSE, FALSE, FALSE,
                          ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                          CLIP_DEFAULT_PRECIS,
                          DEFAULT_QUALITY,
                          DEFAULT_PITCH, "Braggadocio");
```

Bereits anhand der Parameterzahl von **CreateFont**∇ ist zu ermessen, dass die Funktion sehr mächtig und kompliziert ist.
Die Verwendung der Schrift erfolgt durch Erzeugen des Fonts (dies kann innerhalb oder außerhalb des Funktionspaares **BeginPaint** und **EndPaint** erfolgen), Auswahl und Verwendung des Fonts (Innerhalb) und Freigabe (innerhalb oder außerhalb).
Es kann immer nur ein Font zur Zeit gewählt sein. Die Anwahl eines erzeugten Fonts erfolgt über die Funktion **SelectObject**∇:

```
HFONT hOldFont = (HFONT)SelectObject (hDC, hFont);
```

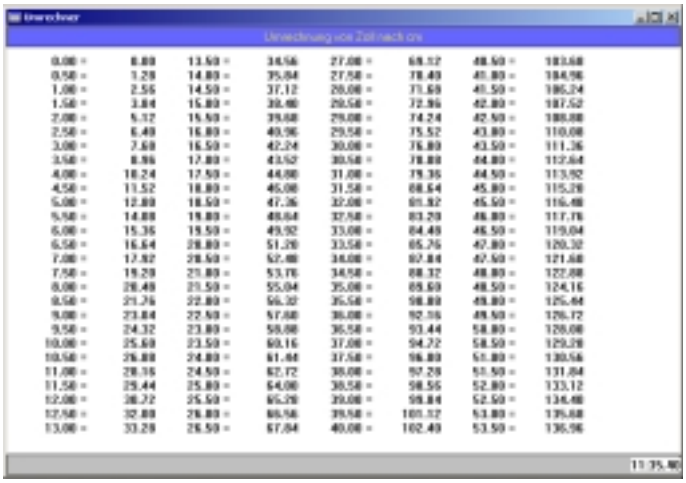
Die Funktion gibt das Handle auf den zuvor eingestellten Font zurück. Dieser sollte unbedingt zwischengespeichert und nach Beendigung der Ausgabe wieder selektiert werden, da das System sonst, nach Freigabe eines noch selektierten Fonts über **DeleteObject** in einem undefinierten Zustand ist.

```
SelectObject (hDC, hOldFont);
```

Eine durchsichtige Schriftausgabe erreicht man durch Setzen des Hintergrundmodus (**SetBKMode**∇) auf den Wert **TRANSPARENT**. Auch diese Funktion gibt den zuvor eingestellten Wert zurück, der wiederum zwischengespeichert und später wiederhergestellt wird. Der eingestellte Wert kann alternativ auch über die Funktion **GetBKMode**∇ ermittelt werden.
Und wenn man schon dabei ist, kann man auch gleich die Schriftfarbe mit **SetTextColor**∇ neu definieren. **SetTextColor** arbeitet über eine Farbreferenz (**RGB** Wert) statt über einen **Brush**, daher muss für **SetTextColor** kein Brush erzeugt werden:

```
hOldFont   = (HFONT)SelectObject (hDC, hFont);
nOldBk     = SetBkMode (hDC, TRANSPARENT);
nOldTxtCol = SetTextColor (hDC, RGB (255,255,0));
DrawText (hDC, sUeberschrift, -1, &aRectTitel,
          DT_SINGLELINE | DT_CENTER | DT_VCENTER);
SetTextColor (hDC, nOldTxtCol);
SetBkMode    (hDC, nOldBk);
SelectObject (hDC, hOldFont);
```

Das in der Titelzeile nun erzielte Ergebnis sieht schon recht annehmbar aus.



Passen Sie die Überschrift Ihrem persönlichen Geschmack an, indem Sie Hintergrundfarbe, Schriftfarbe und Schriftart Ihren Wünschen gemäß verändern.

6.9 Schritt 9 – vorgefertigte Controls

Die Zeitausgabe könnte man jetzt in gleicher Form anpassen, was aber einen relativ hohen Aufwand für eine Ausgabe darstellt, die von der Darstellung her eher einfach gehalten ist.
Hier ist es günstiger, auf bereits vordefinierte Unterfenster zuzugreifen.
Das einfachste Unterfenster oder auch Control ist die statische Textausgabe (static Control), die im Gegensatz zu den komplexeren Controls (Editfelder, Listboxen etc.) keine Eingabefunktionalität aufzuweisen hat.
Ein Control wird genau wie das Hauptfenster über die **CreateWindow**$^{\nabla}$ Funktion erzeugt. Dazu benötigt man zunächst eine eindeutige ID-Nummer und eine Variable für das Windowhandle, damit Windows und das Programm das Control bei Bedarf eindeutig ermitteln können. Die ID definiert man mittels einer symbolischen Konstanten, das Handle als globale Variable, da man es aus unterschiedlichen Funktionen ansprechen muss:

```
#define ID_STATIC_1 10001
HWND hStatic_1;
```

Zur Erzeugung des Unterfensters fügen wir in der **fnWMCreate** Funktion die folgende Anweisung hinzu:

```
hStatic_1 = CreateWindow ("static", "Uhrzeit",
                WS_CHILD | WS_VISIBLE | SS_RIGHT,
                0, 0, 0, 0, hWnd, (HMENU)ID_STATIC_1,
                (HINSTANCE)GetWindowLong (hWnd, GWL_HINSTANCE),
                NULL);
```

Die wichtigsten Styleflags für statische Textfelder:

Symbolische Konstante	Bedeutung
SS_BLACKFRAME	Zeichnet einen Rahmen mit der gleichen Farbe, die auch der

	Rahmen eines Fensters hat
SS_BLACKRECT	Zeichnet eine Fläche mit der gleichen Farbe, die auch der Rahmen eines Fensters hat
SS_CENTER	Zentriert den Text im Static-Control
SS_GRAYFRAME	Zeichnet einen Rahmen mit der gleichen Farbe, die auch der Hintergrund eines Dialogfensters (Desktop) hat: Standard ist Grau.
SS_GRAYRECT	Zeichnet eine Fläche mit der gleichen Farbe, die auch der Hintergrund eines Fensters (Desktop) hat: Standard ist Grau.
SS_ICON	Zeichnet ein angegebenes Piktogramm aus der Resourcedatei
SS_LEFT	Linksbündige Textausgabe im Static-Control
SS_RIGHT	Rechtssbündige Textausgabe im Static-Control
SS_WHITEFRAME	Zeichnet einen Rahmen mit der gleichen Farbe, die auch der Hintergrund eines Fensters hat: Standard ist Weiss.
SS_WHITERECT	Zeichnet eine Fläche mit der gleichen Farbe, die auch der Hintergrund eines Fensters hat: Standard ist Weiss.

Der erste Parameter ist der Fenstertyp. Während wir für die Client-Area unseres Programms einen eigenen Typ erzeugt und registriert haben, ist dies bei der Verwendung eines vordefinierten Fenstertyps natürlich nicht nötig, der Fenstertyp **static** ist bereits bekannt.
Als zweiter Parameter wird ein Fenstertitel angegeben. Bei static-Controls handelt es sich dabei um den im Control anzuzeigenden Text. Man kann ein static-Control also als Fenster betrachten, welches nur aus einer einfachen Titelzeile besteht.

Als dritter Parameter folgen die Window-Styles. Bei den Styleflags ist jetzt **WS_CHILD** mit angegeben. Dieses Flag besagt, dass es sich um ein abhängiges Unterfenster handelt. Dies ist wichtig für die Freigabe des belegten Speicherplatzes. Untergeordnete Fenster werden immer dann zerstört, wenn ihr übergeordnetes Fenster freigegeben (geschlossen) wird. Dadurch entfällt eine große Menge an Verwaltungsarbeit für den Entwickler.
Als nächstes folgen Position und Größe des Fensters. Während der Bearbeitung der **WM_CREATE** Message liegen die Werte der Fenstergröße aber noch nicht vor, so dass das Unterfenster zwar erzeugt, aber noch nicht an die korrekte Position geschoben werden kann. Für Koordinaten und Größe des Unterfensters wird daher einfach der Wert Null angegeben. Da wir ohnehin auf jede Größenänderung des Fensters reagieren müssen, schreiben wir den entsprechenden Code zur Positionierung in die **fnWMSize** Funktion.

Praktischerweise führt die Erzeugung eines Fensters ohnehin dazu, dass für dieses Fenster eine **WM_SIZE** Message generiert wird, sobald das Fenster sichtbar wird:
In der **fnWMSize** Funktion ist also zu ergänzen:

```
MoveWindow (hStatic_1, 0, nActHeight-nCharHeight, nActWidth,
            nCharHeight, TRUE);
```

Der achte Parameter ist das Handle des übergeordneten Fensters (Parentwindow). Für das Betriebssystem ist damit klar, bei welcher Freigabe das Unterfenster mit zu entfernen ist (es werden alle Fenster gelöscht, die den Style **WS_CHILD** und das Handle des zu löschenden Hauptfensters als Parent haben).

Der neunte Parameter verbindet das Unterfenster mit der ID. Dies ist besonders dann wichtig, falls ein Multi-Document-Interface (MDI) erzeugt wird, unser Programm also eventuell mehrere Fenster mit unterschiedlichen Tabellen hat. Jedes dieser Fenster hätte dann ein static-Control mit der gleichen ID und unterschiedlichem Parent-Handle, so dass sie immer noch eindeutig zu unterscheiden wären.

Für die ID wird an dieser Stelle der Eintrag für die Menü-Resource missbraucht. Da ein Unterfenster kein Menü haben kann, ist dieser Speicherplatz frei für eine andere Bedeutung, muss aber formal auf den Typ HMENU gecastet werden.
Der zehnte Parameter dient ebenfalls der eindeutigen Identifikation, in diesem Fall ist es das Instanzenhandle des Programms. Es wird benötigt, um die einzelnen Fenster zu unterscheiden, fall das gleiche Programm mehrfach gestartet wird. An dieser Stelle hätte man auch einfach eine globale Variable mit der Kopier von **hInstance** aus der **WinMain**$^{\nabla}$ Funktion benutzen können. Wir haben **hInstance** aber nicht gesichert – denn das ist auch nicht nötig, da wir eine Kopie von **hInstance** im **CreateWindow** der Client-Area eingebaut haben. Dementsprechend kann man diese Variable auch über das Handle der Client-Area (hier **hWnd**) unter zu Hilfenahme der Funktion **GetWindowLong**$^{\nabla}$ wieder abrufen.
Der Letzte Parameter ist wieder die Angabe von Extrabytes für eigene Zwecke, die wir hier jedoch nicht benötigen.

Nun da wir ein Unterfenster zur Anzeige der Uhrzeit haben, können wir das Schreiben der Uhrzeit über **DrawText** aus der Funktion **fnWMPaint** herausnehmen. Durch die Verwendung des Unterfensters ist unsere **WM_PAINT** Funktion dafür nicht mehr zuständig, sondern die **WM_PAINT** Routine des Fenstertyps **static** muss dies leisten. Die **WndProc** Funktion des Fenstertyps **static** ist in Windows bereits enthalten. Sie kann nicht viel mehr als sich einen Text merken und in der eigenen Client-Area (also dem Unterfenster) formatiert ausgeben.

Um den Text des Unterfensters zu setzen nutzen wir die **SetWindowText**$^{\nabla}$ Funktion, mit welcher der Titel eines Fensters gesetzt werden kann (es ist also in der tat so, dass ein **static** Control ein Fenster nur bestehend aus dem Titelbereich ist):

```
SetWindowText (hStatic_1, sTime);
```

Da wir nun die **WM_PAINT** Funktion zur Ausgabe der Uhrzeit nicht mehr benötigen, kann alles was mit der Ermittlung und Ausgabe der zeit in Zusammenhang steht problemlos in die **fnWMTimer** Funktion überführt werden:

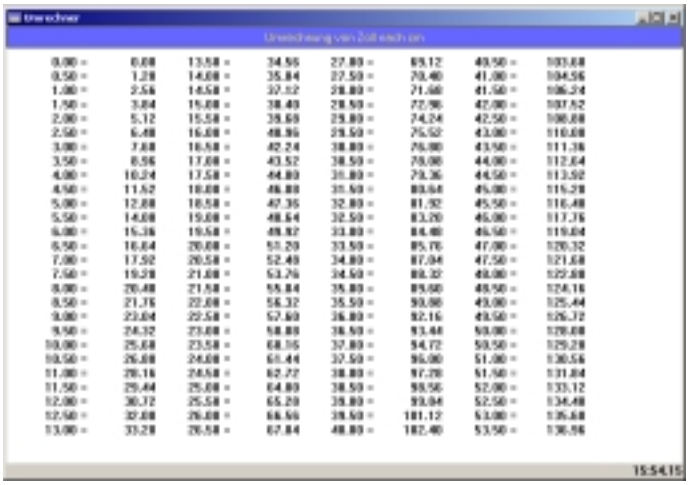
```
LRESULT fnWMTimer (HWND hWnd, WPARAM wParam, LPARAM lParam)
{
   char        sTime [10] = "";
   struct time aTime;
   gettime(&aTime);
   sprintf (sTime, "%02d:%02d.%02d",
            aTime.ti_hour, aTime.ti_min, aTime.ti_sec);
```

```
   SetWindowText (hStatic_1, sTime);
   return 0;
}
```

Auch die **fnWMPaint** Funktion wird durch Entfernen der Programmanweisungen für den Timer deutlich übersichtlicher:

```
LRESULT fnWMPaint (HWND hWnd, WPARAM wParam, LPARAM lParam)
{
   HDC         hDC ;
   PAINTSTRUCT aPS ;
   RECT        aRectTitel;
   char        sUeberschrift [2*EINHEITTEXTLEN + 50];
   HBRUSH      hBlueBrush;
   HFONT       hFont, hOldFont;
   int         nOldBk;
   COLORREF    nOldTxtCol;

   //-------------------------------------------------------
   // Überschrift
   //-------------------------------------------------------
   GetClientRect (hWnd, &aRectTitel);
   aRectTitel.bottom = aRectTitel.top + 1.5 * nCharHeight;
   hBlueBrush = CreateSolidBrush (RGB (100, 100, 255));
   hFont      = CreateFont (nCharHeight, 0, 0, 0, FW_THIN,
                FALSE, FALSE, FALSE,
                ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                DEFAULT_PITCH, "Braggadocio");
   sprintf (sUeberschrift, "Umrechnung von %s nach %s",
            sEinheit1, sEinheit2);
   //-------------------------------------------------------
   hDC = BeginPaint (hWnd, &aPS);
   FillRect (hDC, &aRectTitel, hBlueBrush);
   FrameRect(hDC, &aRectTitel,
             (HBRUSH)GetStockObject (BLACK_BRUSH));
   hOldFont   = (HFONT)SelectObject (hDC, hFont);
   nOldBk     = SetBkMode (hDC, TRANSPARENT);
   nOldTxtCol = SetTextColor (hDC, RGB (255,255,0));
   DrawText (hDC, sUeberschrift, -1, &aRectTitel,
             DT_SINGLELINE|DT_CENTER|DT_VCENTER);
   SetTextColor (hDC, nOldTxtCol);
   SetBkMode    (hDC, nOldBk);
   SelectObject (hDC, hOldFont);

   fnValueTable(hDC);
   EndPaint (hWnd, &aPS);
   //-------------------------------------------------------
   DeleteObject (hBlueBrush);
   DeleteObject (hFont);
   return 0;
}
```

Nach Erzeugung des Unterfensters und Erweiterung von **fnWMSize**, **fnWMTimer** und dem Umbau von **fnWMPaint** sieht das Programm schon deutlich besser aus:



Wer mag kann wieder ein Rechteck um die Zeitausgabe legen, indem als zusätzlicher Window Style des Unterfensters **WS_BORDER** angegeben wird.

Die Titelzeile kann leider nicht nach gleichem Verfahren angepasst werden, da das static Control weder mit unterschiedlichen Farben noch mit TrueType Fonts arbeiten kann.

Ändern Sie das Programm so ab, dass die Uhrzeit in einem static-Control ausgegeben wird.

6.10 Schritt 10 – Linien

Als nächstes wollen wir die Ausgabespalten optisch noch ein wenig trennen. Dazu erhöhen wir zunächst den Abstand der Spalten in der Funktion **fnValueTable** von 23 auf 25 Zeichen und fügen am Ende der **while** Schleife die folgenden Zeilen hinzu:

```
MoveToEx (hDC, j*25*nSmallCharWidth, 2*nCharHeight, NULL);
LineTo   (hDC, j*25*nSmallCharWidth, (nLine+2)*nCharHeight);
```

Die Funktion **MoveToEx**$^{\nabla}$ bewegt einen virtuellen Stift (**Pen**) an die Anfangsposition, zeichnet aber nicht. Die Funktion **LineTo**$^{\nabla}$ hingegen nimmt nur die Endposition des Linienzugs entgegen und bewegt den virtuellen Stift anschließend auf diese Position, so dass mit mehreren aufeinanderfolgenden **LineTo** Befehlen ein geschlossener Linienzug erzeugt wird.
Der Linienbefehl benutzt den aktuell im System eingestellten **Pen**. Ein **Pen** ist eine limitierte Resource, die einem **Brush** ähnelt und im Prinzip genauso verwendet wird (**SelectObject**, **DeleteObject**). Zur Erzeugung eines eigenen Stiftes wird die Funktion **CreatePen**$^{\nabla}$ verwendet:

```
HPEN hPen = CreatePen (PS_SOLID, 1, RGB (0, 0, 0));
```

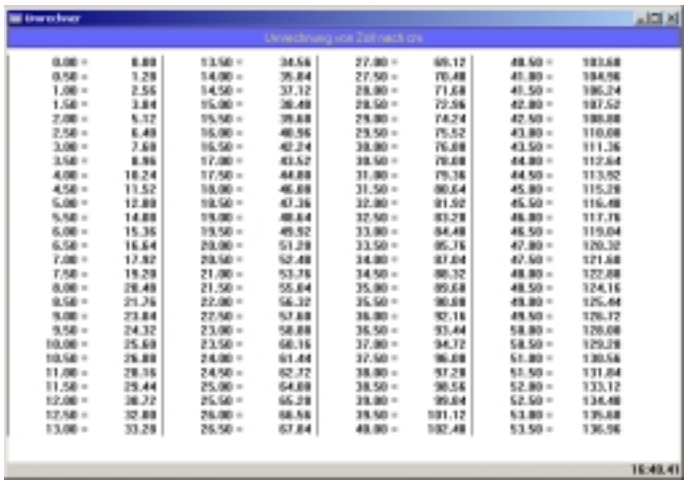
CreatePen erhält als Parameter einen Pen-Style, eine Stiftbreite und eine Farbreferenz. Die wichtigsten Pen-Styles sind:

Symbolische Konstante	Bedeutung

PS_SOLID	Durchgehende Linie
PS_DASH	gestrichelte Linie
PS_DOT	gepunktete Linie
PS_DASHDOT	Strich-Punkt Linie
PS_DASHDOTDOT	Strich-Punkt-Punkt Linie
PS_NULL	unsichtbare Linie

Ändern Sie das Programm so ab, dass die Umrechnungsblöcke durch Linien voneinander getrennt werden.

Das Ergebnis sollte in etwa wie folgt aussehen:



6.11 Schritt 11 – Menüs

Nachdem die Oberfläche jetzt durchaus passabel aussieht, wollen wir die Umrechnungen erweitern. Dazu benötigen wir ein Menü, welches es uns erlaubt, zwischen einzelnen Umrechnungsarten umzuschalten.

Dazu legen wir per Hand oder mit Hilfe eines Resource-Editors zwei neue Dateien an, eine *.rc (Resource) Datei und eine zugehörige Headerdatei *.rh (Resource-Header):

```
// Resource Header
#ifndef _WIN_11_RH_
#define _WIN_11_RH_

#define IDM_MENU1              20000
#define IDM_ZOLLCM             20001
#define IDM_CELSIUSFAHRENHEIT 20002

#endif
```

Resource-Datei:

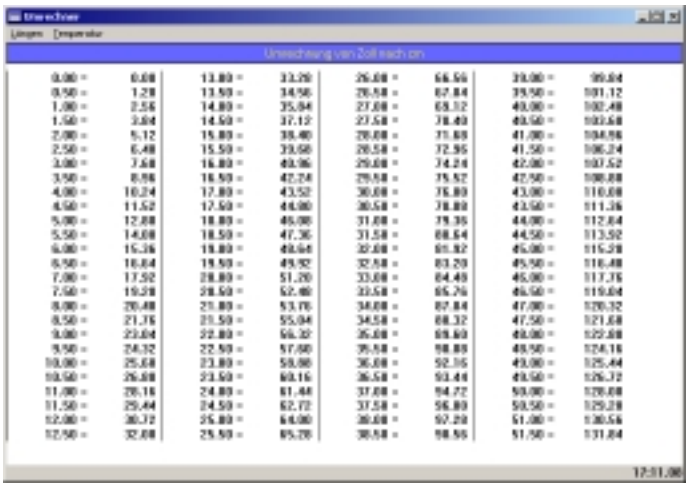
```
#include "Win_11.rh"

IDM_MENU1 MENU
{
 POPUP "&Längen"
 {
  MENUITEM "&Zoll in cm", IDM_ZOLLCM
 }
```

```
 POPUP "&Temperatur"
 {
  MENUITEM "&Celsius in Fahrenheit", IDM_CELSIUSFAHRENHEIT
 }
}
```

Im Hauptprogramm ist der Resource-Header mit einzubinden **(#include**) und die Resource-Datei muss in das Projekt mit aufgenommen werden.

```
aWndclass.lpszMenuName  = MAKEINTRESOURCE(IDM_MENU1);
```

Somit ist das Menü aus der Resource mit der Client-Area verbunden:



Immer wenn einer der Menüpunkte gewählt wird, so erhält die Funktion **WndProc** eine **WM_COMMAND** Message über das Ereignis. Das **WM_COMMAND** Ereignis behandeln wir nach bewährtem Muster mit einer **fnWMCommand** Funktion:

```
case WM_COMMAND: return fnWMCommand (hWnd, wParam, lParam);
```

WM_COMMAND Messages werden von allen Formen von Controls ausgelöst, sie sind die universellsten Messages in Windows und die Bedeutung der Parameter **wParam** und **lParam** ist immer auch vom Typ des Controls und dem Ereignis abhängig.

```
LRESULT fnWMCommand (HWND hWnd, WPARAM wParam, LPARAM lParam)
{
  WORD wID = LOWORD(wParam);
  switch (wID)
  {
    case  IDM_ZOLLCM:
          strcpy (sEinheit1, "Zoll");
          strcpy (sEinheit2, "cm");
          InvalidateRect (hWnd, NULL, TRUE);
          return 0;

    case  IDM_CELSIUSFAHRENHEIT:
          strcpy (sEinheit1, "°Celsius");
          strcpy (sEinheit2, "°Fahrenheit");
```

```
            InvalidateRect (hWnd, NULL, TRUE);
            return 0;
    }
    return DefWindowProc (hWnd, WM_COMMAND, wParam, lParam);
}
```

Je nach gewähltem Menüpunkt wird jetzt schon einmal der Text der Titelzeile geändert. Der Aufruf von **InvalidateRect** lässt die Änderungen wirksam werden, da dadurch die Tabelle neu gezeichnet wird.
Bisher ist nur die Titelanzeige verallgemeinert. Um auch die Umrechnung umstellen zu können, müssen auch die Berechnungsvariablen verallgemeinert werden, d.h. in globalen Variablen stehen:

```
//============================================================
// globale Variablen
//============================================================
int nAnzZeichen, nAnzZeilen;
int nSmallCharWidth, nBigCharWidth, nCharHeight;
char sEinheit1 [EINHEITTEXTLEN] = "Zoll";
char sEinheit2 [EINHEITTEXTLEN] = "cm";
double fFaktor      = 2.56;
bool   bMultiply    = true;
double fOffset      = 0.0;
bool   bAddOffset   = false;
bool   bOffsetFirst = false;
```

Der Aufruf in **fnValueTable** ist entsprechend anzupassen und in **fnWMCommand** sind natürlich alle diese Variablen korrekt zu setzen.

```
    case  IDM_ZOLLCM:
          strcpy (sEinheit1, "Zoll");
          strcpy (sEinheit2, "cm");
          fFaktor      = 2.56;
          bMultiply    = true;
          fOffset      = 0.0;
          bAddOffset   = false;
          bOffsetFirst = false;
          InvalidateRect (hWnd, NULL, TRUE);
          return 0;

    case  IDM_CELSIUSFAHRENHEIT:
          strcpy (sEinheit1, "°Celsius");
          strcpy (sEinheit2, "°Fahrenheit");
          fFaktor      = 5.0/9.0;
          bMultiply    = false;
          fOffset      = 32.0;
          bAddOffset   = true;
          bOffsetFirst = false;
          InvalidateRect (hWnd, NULL, TRUE);
          return 0;
```

Ändern Sie das Programm so ab, dass die Berechnung die globalen Variablen berücksichtigt und verallgemeinern Sie auch die Schrittweite und Startwert der Berechnung sowie die Anzeigenformatierung der Tabelle. Fügen Sie zudem Umrechnungen von Fahrenheit in Celsius und cm in Zoll hinzu.

6.12 Schritt 12 – mehr Komfort

Als nächstes wollen wir ein wenig mehr Komfort in das Programm hineinbringen. So soll ist es z.B. wünschenswert, den Startwert, die Schrittweite sowie die Vor- und Nachkommastellen gemäß Anwenderwunsch gestalten zu können.
Dazu führen wir zunächst die benötigten Informationen als globale Variablen ein:

```
double fSchrittweite = 1.0;
double fStartwert    = 0.0;
int    nStellenVK    =  10;
int    nStellenNK    =   2;
bool   bStellenNK    = true;
```

Die logische Variable zum Ein- und Ausschalten der Nachkommastellen ist eigentlich nicht nötig, da man in der Variablen **nStellenNK** auch den Wert Null angeben könnte, er ist aber – wie noch zu sehen sein wird – hier zu Demonstrationszwecken ganz praktisch.

Zum Erzeugen der Eingabefelder bedienen wir uns der gleichen Technik, wie bei der Ausgabe der Uhrzeit, indem wir Unterfenster (Controls) anlegen. Für den Startwert und die Schrittweite soll jeweils ein editierbares Feld Verwendung finden (Editfeld), für den logischen Schalter zum Ausblenden der Nachkommastellen eine Checkbox und für die Anzahl der Vor- und Nachkommastellen ein Popup.
Wie beim static-Control benötigen wir dazu eindeutige ID Nummern für die Unterfenster:

Die Benennung folgt hier wiederum einem Schema ähnlich der Ungarischen Notation, indem hier der Typ des Controls als Buchstabenkombination in den Namen aufgenommen wird. Da es sich aber um **#define** Werte handelt und solche in C/C++ üblicherweise in Großbuchstaben geschrieben werden, trennt man den Typ durch einen Unterstrich ab. Die folgende, alphabetisch sortierte Tabelle zeigt ein mögliches Abkürzungsschema:

Präfix	Bedeutung
ACC	Accelerator / Hotkey
B	Button
BMP	Bitmap
CB	Combobox / Popup
CK	Checkbox
DLG	Dialog
E	Editfeld
F	Frame / Linie
ICO	Icon
L	Label / Static Control
LB	Listbox
M	Menü
PB	Forschrittsanzeige (Progressbar)
RB	Radiobutton
RE	Rich-Editfeld
S	Stringresource
SB	Scrollbar / Schieberegler / Drag Control

ST	Statusbar
T	Timer
TAB	Tab-Control
TB	Toolbar
TBL	Tabelle
TC	Tree Control
TK	Ticker / Wippregler / Up-Down-Control

Da wir schon dabei sind, ändern wir die symbolischen Konstanten für den Timer und das static-Control und den Namen der Handle-Variablen des static-Controls auch gleich entsprechend ab. Außerdem benötigen wir für die neuen Controls natürlich auch neue Handles:

```
#define TIM_ZEIT        10000
#define L_ZEIT          10001
#define E_START         10002
#define E_SCHRITT       10003
#define CK_NACHKOMMA    10004
#define CB_NACHKOMMA    10005
#define CB_VORKOMMA     10006

HWND       hLZeit; // vorher: hStatic_1
HWND       hEStart;
HWND       hESchritt;
HWND       hCKNachkomma;
HWND       hCBNachkomma;
HWND       hCBVorkomma;
```

Anschließend erzeugen wir in der Funktion **fnWMCreate** die zusätzlichen Controls.

```
hEStart = CreateWindow ("edit", "",
        ES_RIGHT | WS_CHILD | WS_VISIBLE | WS_BORDER |
        WS_TABSTOP,
        0, 0, 0, 0, hWnd, (HMENU)E_START,
        (HINSTANCE)GetWindowLong (hWnd, GWL_HINSTANCE), NULL);
hESchritt = CreateWindow ("edit", "",
        ES_RIGHT | WS_CHILD | WS_VISIBLE | WS_BORDER |
        WS_TABSTOP,
        0, 0, 0, 0, hWnd, (HMENU)E_SCHRITT,
        (HINSTANCE)GetWindowLong (hWnd, GWL_HINSTANCE), NULL);
hCBVorkomma = CreateWindow ("combobox", NULL,
        CBS_DROPDOWNLIST | WS_CHILD | WS_VISIBLE | WS_TABSTOP,
        0, 0, 0, 0, hWnd, (HMENU)CB_VORKOMMA,
        (HINSTANCE)GetWindowLong (hWnd, GWL_HINSTANCE), NULL);
hCBNachkomma = CreateWindow ("combobox", NULL,
        CBS_DROPDOWNLIST | WS_CHILD | WS_VISIBLE | WS_TABSTOP,
        0, 0, 0, 0, hWnd, (HMENU)CB_NACHKOMMA,
        (HINSTANCE)GetWindowLong (hWnd, GWL_HINSTANCE), NULL);
hCKNachkomma = CreateWindow ("button", "Nachkomma",
        BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP,
        0, 0, 0, 0, hWnd, (HMENU)CK_NACHKOMMA,
        (HINSTANCE)GetWindowLong (hWnd, GWL_HINSTANCE), NULL);
```

Die wichtigsten Styleflags für Editfelder:

Symbolische Konstante	**Bedeutung**
ES_AUTOHSCROLL	Scrollt automatisch in der horizontalen, wenn nötig.

ES_AUTOVSCROLL	Scrollt automatisch in der vertikalen, wenn nötig.
ES_CENTER	Zentriert den Text im Editfeld
ES_LEFT	Linksbündige Ausgabe im Editfeld
ES_LOWERCASE	Wandelt automatisch alle Buchstaben in Kleinbuchstaben um
ES_MULTILINE	Mehrzeiliges Editfeld
ES_PASSWORD	Passworteingabe, eingegebene Zeichen werden optisch als Sterne dargestellt.
ES_RIGHT	Rechtsbündige Ausgabe im Editfeld
ES_UPPERCASE	Wandelt automatisch alle Buchstaben in Großbuchstaben um

Die wichtigsten Styleflags für Buttons:

Symbolische Konstante	**Bedeutung**
BS_AUTOCHECKBOX	Button ist Checkbox, Markierung wird automatisch vorgenommen
BS_AUTORADIOBUTTON	Button ist Radiobutton, Markierung wird automatisch vorgenommen
BS_AUTO3STATE	Button ist Checkbox mit drei Zuständen, Markierung wird automatisch vorgenommen
BS_CHECKBOX	Button ist Checkbox
BS_DEFPUSHBUTTON	Pushbutton, der wird mit Enter automatisch ausgelöst wird
BS_LEFTTEXT	Text ist linksbündig angeordnet
BS_PUSHBUTTON	Standard Pushbutton
BS_RADIOBUTTON	Button ist Radiobutton
BS_3STATE	Button ist Checkbox mit drei Zuständen

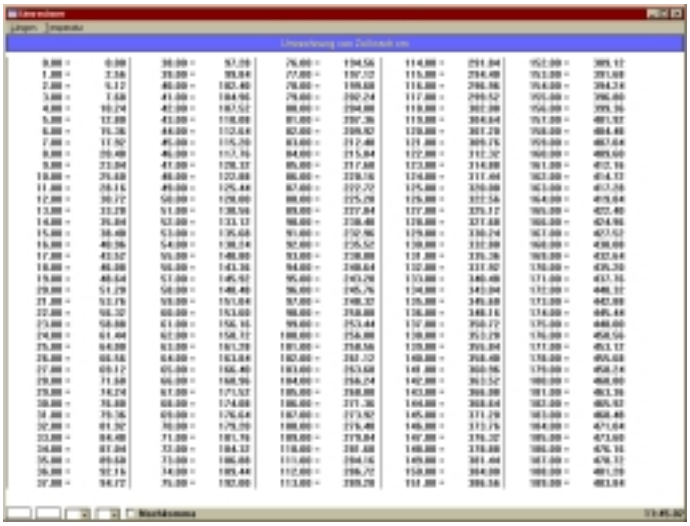
Die wichtigsten Styleflags für Comboboxen:

Symbolische Konstante	**Bedeutung**
CBS_AUTOHSCROLL	Scrollt automatisch horizontal bei der Eingabe im Editfeld, wenn nötig
CBS_DROPDOWN	Einfache Combobox, Listboxteil wird erst angezeigt, wenn auf den Button geklickt wird
CBS_DROPDOWNLIST	Combobox ohne Editierbares Feld
CBS_SIMPLE	Listboxteil wird immer angezeigt
CBS_SORT	Box wird automatisch sortiert

Auch diese Unterfenster müssen natürlich bei einer WM_SIZE Message mittels der **MoveWindow** Funktion an die jeweils richtige Position geschoben werden. Die einzig "passende" Stelle ist die unterste Zeile, die bisher allein der Zeitanzeige gehört. Hier fügen wir die neuen Controls an der linken Seite ein und verkleinern des static-Control für die Zeitausgabe entsprechend.
Bei den Comboboxen ist zu beachten, dass sie in der Vertikalen größer angegeben werden müssen, als sie zunächst erscheinen. mit einzurechnen ist die Dropdownliste, die erst bei Betätigung des Aufklapp-Buttons sichtbar wird.

Ändern Sie das Programm so ab, dass die neuen Controls in der untersten Zeile angezeigt werden.

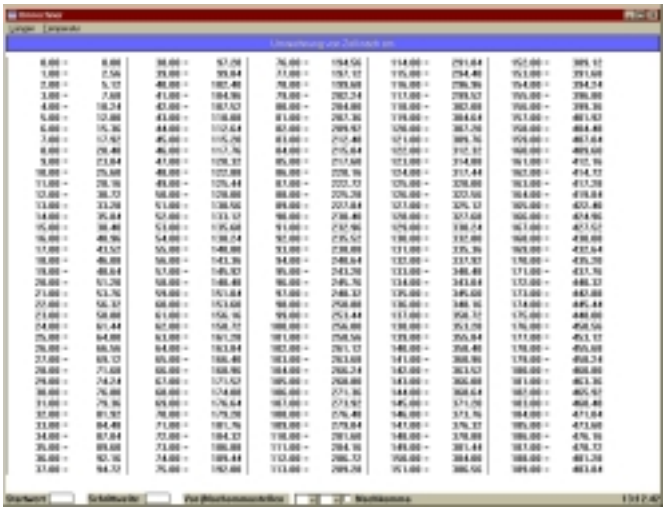
Das Ergebnis sollte dann in etwa wie folgt aussehen:



Damit der Anwender auch weiß, was die einzelnen Felder bedeuten, müssen weitere static-Controls mit Beschriftungen vor den Edit- und den Popup-Feldern eingefügt werden.

Fügen Sie die Beschriftungen als static-Controls hinzu.

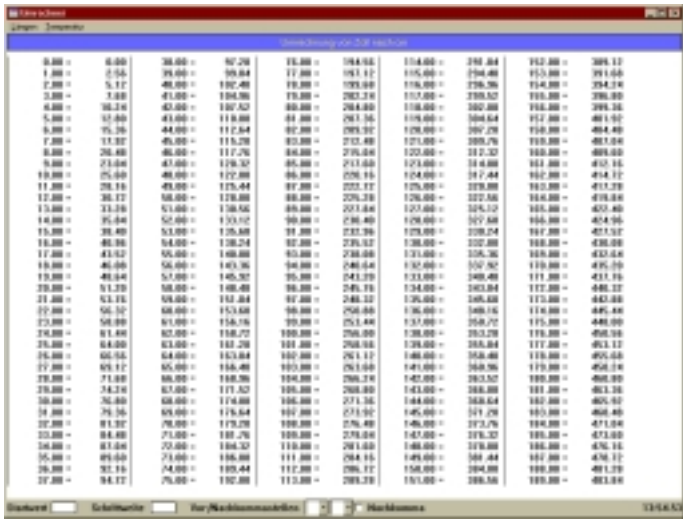
Als Ergebnis erhalten wir in etwa folgendes Bild:



Jetzt stören nur noch die entstandenen Lücken und die Tatsache, dass die Popup-Boxen etwas höher sind, als die Fläche des bisher grau gezeichneten unteren Rands.
Um dies zu ändern fügen wir einerseits ein zusätzliches static-Control ohne Beschriftung ein, welches als Hintergrund fungiert. Außerdem machen wir dieses Control ein wenig höher (8 Pixel) und setzen die y-Koordinate der Editfelder um 6 Pixel, der beiden Comboboxen um 8 Pixel und aller übrigen Controls um 4 Pixel herunter (dadurch sind die Beschriftungen wieder mittig).

Beachten Sie bitte, dass der **MoveWindow** Befehl für das Hintergrundfeld als letztes ausgeführt werden muss, da die Reihenfolge

in der die Bildbereiche invalidiert werden aufgrund der sich überlagernden Controls eine wichtige Rolle spielt.
Danach sieht das Ergebnis aus wie gewünscht:



Abschliessend sind die Vorbelegungen noch in die neuen Controls einzutragen. Die Editfelder müssen mit den Werten aus den globalen Variablen gefüllt werden, die Checkbox entsprechend ist gemäß der Nachkommstellen **bool**-Variablen zu markieren und die Comboboxen sollten Werte zur Auswahl anbieten.

Für die Editfelder kann dies wieder mit der Funktion **SetWindowText** erledigt werden. Analog zum static-Controls gilt auch für die Editfelder, dass es sich um Fenster handelt, die eigentlich nur aus einer Titelzeile bestehen (im Gegensatz zum static-Control ist der Titel des Editfeldes halt editierbar).
Die Vorbelegung kann problemlos in der Behandlungsfunktion zu **WM_CREATE** mit erledigt werden:

```
char sTemp [255] = "";
sprintf (sTemp, "%lf", fStartwert);
SetWindowText (hEStart, sTemp);
```

Für die Checkbox benötigen wir die Funktion **SendMessage**$^{\nabla}$:

```
SendMessage (hCKNachkomma, BM_SETCHECK,
             (WPARAM)(bStellenNK ? TRUE : FALSE), (LPARAM)0);
```

Über den Nachrichtenmechanismus von Windows erhält die Checkbox nun die Anweisung, den Haken gemäß der Variablen **bStellenNK** zu setzen. Um die Umsetzung müssen wir uns nicht selbst kümmern, denn das wird von der internen **WndProc** des Fenstertyps "button" automatisch übernommen.
Es gibt für das Markieren einer Checkbox leider keinen einfacheren Befehl. Auch der oben verwendete Befehl **SetWindowText** ist letztlich nur die Kurzform für:

```
SendMessage (hEStart, WM_SETTEXT, (WPARAM)0,
```

```
                (LPARAM)(LPCTSTR)sTemp);
```

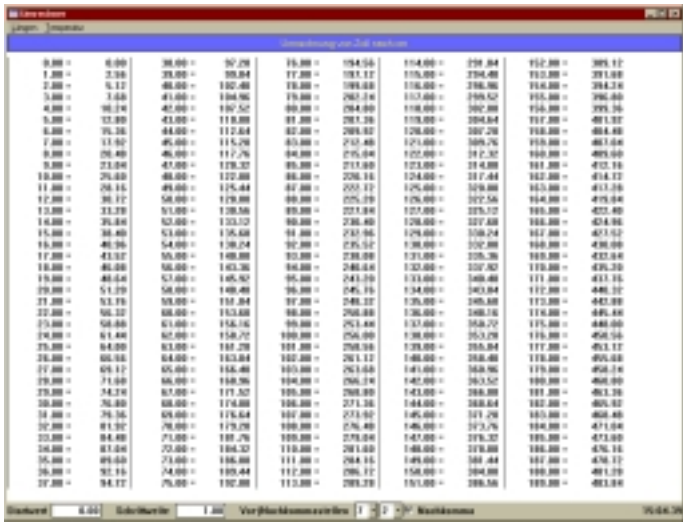
Das Einfügen von Einträgen in eine Combobox oder ein Popup ist einfach, lediglich die Auswahl als Vorbelegung ist ein klein wenig komplizierter:

```
SendMessage (hCBVorkomma, CB_ADDSTRING, (WPARAM)0,
             (LPARAM)(LPCTSTR)"1");
//...
SendMessage (hCBVorkomma, CB_ADDSTRING, (WPARAM)0,
             (LPARAM)(LPCTSTR)"10");
```

Die Auswahl wird zusätzlich ein wenig schwieriger, als dass der zur Formatierung verwendete **sprintf** Befehl nur die Gesamtzahl der Stellen (und davon die Nachkommastellen) kennt. Wir müssen also noch ein wenig rechnen:

```
nTemp = nStellenVK - nStellenNK - 1;
nTemp = nTemp > 1 ? nTemp : 1;
sprintf (sTemp, "%d", nTemp);
nIndex = SendMessage (hCBVorkomma, CB_FINDSTRING, (WPARAM)-1,
                      (LPARAM)(LPCSTR)sTemp);
SendMessage (hCBVorkomma, CB_SETCURSEL, (WPARAM)nIndex,
             (LPARAM)0);
```

Das Setzen und die Auswahl der Nachkommastellen erfolgt analog.



Zum Abschluss ändern Sie bitte die Funktion fnWMSize so ab, dass die Controls immer sichtbar bleiben (Achtung: auch hier gibt es ein Reihenfolgen-Problem !!).

6.13 Schritt 13 – Reaktion auf Eingaben

Nachdem die neuen Controls eingefügt und mit Startwerten versehen wurden, muss noch auf Änderungen an diesen Controls reagiert werden. Dazu sind zunächst die Control ID Nummern in der **fnWMCommand** Funktion einzutragen (die static-Texte kann man sich natürlich sparen).

```
LRESULT fnWMCommand (HWND hWnd, WPARAM wParam, LPARAM lParam)
{
  WORD wID = LOWORD(wParam);
  switch (wID)
  {
    case  E_START:      return 0;
    case  E_SCHRITT:    return 0;
    case  CK_NACHKOMMA: return 0;
    case  CB_NACHKOMMA: return 0;
    case  CB_VORKOMMA:  return 0;
    // …
```

Anders als bei den Menüeinträgen, die ein Ereignis nur dann an die **WndProc** Funktion melden, wenn sie ausgelöst wurden, muss man hier jedoch genau unterscheiden, auf welche Ereignisse man reagieren möchte und auf welche nicht.
So schickt z.B. ein Editfeld bei jeder Eingabe eines Buchstabens gleich drei Messages (sogenannte Notification Messages) an die **WndProc** Funktion (**WM_KEYDOWN**, **WM_CHAR** und **WM_KEYUP**). Diese Botschaften sind aber für unser Programm zumeist uninteressant, denn der Neuaufbau der Tabelle soll ja nicht dreimal pro Tastenanschlag durchgeführt werden, sondern nur dann wenn der Anwender die Eingabe abgeschlossen hat.
Notification Messages sind im **HIWORD** des **wParam** der **WM_COMMAND** Messages verpackt (damit wissen wir auch, wozu die zweite Hälfte des **wParam** dient, der **lParam** des **WM_COMMAND** enthält praktischerweise noch einmal das Handle des Controls) und informieren das Anwendungsprogramm darüber, das ein bestimmtes Ereignis eingetreten ist.
Das für unseren Zweck geeignetste Ereignis ist die Nachricht, dass das Eingabefeld den Eingabefokus verliegt, der Anwender also das Feld mit der Tabulatortaste verlassen oder mit der Maus außerhalb des Eingabefeldes geklickt hat. Die korrespondierende Message zu diesem Ereignis ist **EN_KILLFOCUS** (EN = Editfield Notification):

```
case  E_START:
      if (wNotification == EN_KILLFOCUS)
      {
         char sTemp [101] = "";
         GetWindowText ((HWND)lParam, sTemp, 100);
         fStartwert = atof (sTemp);
         InvalidateRect (hWnd, NULL, TRUE);
         return 0;
      }
      break;
```

Beachten Sie bitte, dass die Message nur im Fall einer Bearbeitung mit **return** abgeschlossen wird. Das ist wichtig, weil bei allen anderen Messages (wie z.B. das Einfügen eines Zeichens), natürlich das Standardverhalten zum tragen kommen muss. Der Returnwert Null hingegen signalisiert dem System, dass wir alle notwendigen arbeiten erledigt haben und nichts mehr zur tun ist. Im Falle des **EN_KILLFOCUS** ist das völlig in Ordnung, denn das Standardverhalten des Systems ist für diese Botschaft: tue nichts.

Wenn Sie jetzt im Feld für den Startwert eine Änderung vornehmen, sollte das Programm sofort darauf reagieren.

Um den Wert (Haken gesetzt oder nicht) der Checkbox zu ermitteln müssen Sie der Checkbox eine Nachricht senden, welche mit einem Status (hier **BST_CHECKED** oder **BST_UNCHECKED**) beantwortet wird:

```
case  CK_NACHKOMMA:
      if (wNotification == BN_CLICKED)
      {
         LRESULT nCheck;
         nCheck = SendMessage ((HWND)lParam, BM_GETCHECK,
                               (WPARAM)0, (LPARAM)0);
         bStellenNK = nCheck == BST_CHECKED;
         InvalidateRect (hWnd, NULL, TRUE);
         return 0;
      }
      break;
```

Für die Combobox / Popup ist es wiedereinmal etwas aufwendiger, weil zunächst der Index des Eintrags festgestellt werden muss:

```
case  CB_NACHKOMMA:
      if (wNotification == CBN_SELCHANGE)
      {
         LRESULT nIndex;
         char sTemp [101] = "";
         nIndex = SendMessage ((HWND)lParam, CB_GETCURSEL,
                               (WPARAM)0, (LPARAM)0);
         SendMessage ((HWND)lParam, CB_GETLBTEXT,
                      (WPARAM)nIndex, (LPARAM)(LPCSTR)sTemp);
         nStellenNK = atoi (sTemp);
         InvalidateRect (hWnd, NULL, TRUE);
         return 0;
      }
      break;
```

Fügen Sie die noch fehlenden Programmreaktionen ein. Die Notification Messages auf die Sie reagieren müssen sind EN_KILLFOCUS für die Editfelder, CBN_SELCHANGE für die Popups und BN_CLICKED für die Checkbox.

6.14 Schritt 14 – Tastatursteuerung

Was nun noch fehlt, ist die Möglichkeit mit der Tabulatorteste zwischen den Eingabe-Controls zu wechseln. Diese Funktionalität ist in sogenannten Dialogen (siehe unten) automatisch gegeben, der Client-Area hingegen fehlt sie.
Um sie nachzuahmen ist es notwendig, den entsprechenden Code selbst hinzuzufügen. Glücklicherweise gibt es die Funktion **GetNextDlgTabItem**[▽], die selbst das nächste anzuspringende Control feststellt und den Eingabefokus dorthin verlagert.
Diese Funktion ist auszuführen, sowie die Tabulatortaste losgelassen wird. war zugleich die Umschalt-Taste (Shift) gedrückt, so soll das vorangegangene Control angesprungen werden, sonst das nächste.

Dazu benötigen wir neben dem Handle des Hauptfensters auch das Handle des Controls, auf dem die Tabulatortaste gedrückt wurde. Windows teilt dies in einer WM_KEYUP Message an das entsprechende Control mit. Somit verfügen wir über das globale Handle (hier **hWindow**) und das Handle des Controls (das **hWnd** in der Message-Struktur).
Die Verarbeitung kann somit allgemeingültig erfolgen sollen müssen und lassen sich sehr bequem in der **while**-Schleife des Hauptprogramms erledigen. Später geht dies nicht, denn die Nachricht, dass die Tabulatortaste gedrückt worden ist landet ja nicht in der von uns geschriebenen **WndProc** sondern in den Standard **WndProc** Funktionen der unterschiedlichen Controls, alles was normalerweise geschieht, ist dass diese alle Botschaften die sie nicht selbst verarbeitet haben, an das **WndProc** des übergeordneten Fensters weiterreichen (das ist dann das **WndProc** unserer Anwendung) – genauso, wie unsere Funktion **DefWindowProc**$^{\nabla}$ aufruft.
Alle Teile die allgemeingültig abgefangen werden sollen, bevor sie an die eigentlichen Bearbeitungsfunktionen weitergereicht werden, fasst man in einer Funktion zusammen (außer dem Aufruf einer allgemeinen Hilfe, sind es zumeist ohnehin nicht viele), für die sich der Name **MessagePreprocess** eingebürgert hat:

```
BOOL MessagePreprocess (HWND hWnd, MSG &aMsg)
{
  BOOL bRC = FALSE;
  static bool bShiftDown = false;

  if (((WM_KEYDOWN == aMsg.message)
        || (WM_KEYUP == aMsg.message))
      && (VK_SHIFT == (int)aMsg.wParam))
  {
      bShiftDown = WM_KEYDOWN == aMsg.message;
  }

  if ((WM_KEYUP == aMsg.message)
       && (VK_TAB == (int)aMsg.wParam))
  {
     SetFocus(GetNextDlgTabItem(hWnd, aMsg.hwnd, bShiftDown));
     bRC = TRUE;
  }
  return bRC;
}
```

Weil die Umschalttaste (Shift) und die Tabulatortaste in getrennten **WM_KEYUP** und **WM_KEYDOWN** Messages geliefert werden, muss man den Zustand der Shift-Taste leider zwischenspeichern.

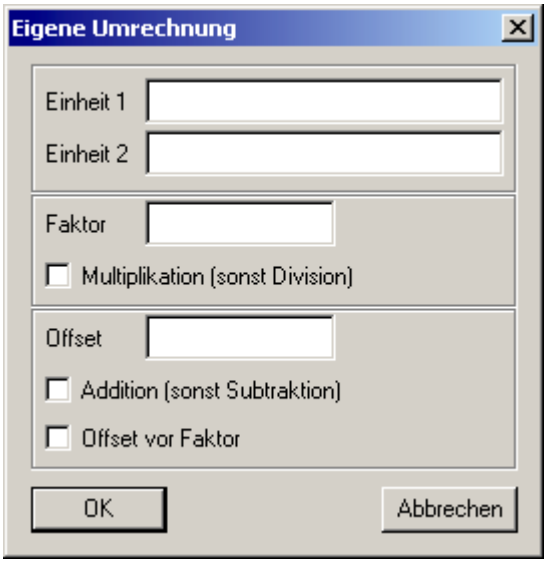
6.15 Schritt 15 – Dialogbox

Für den nächsten Schritt erweitern wir zunächst unser Menü um einen weiteren Menüeintrag („Eigene Umrechnungen“ mit dem Menüpunkt „Hinzufügen“) und ändern dabei zugleich die bisherigen Einträge auf den Stil der oben angegebenen Ungarischen Notation:

```
IDM_MENU1 MENU
{
```

```
 POPUP "&Längen"
 {
  MENUITEM "&Zoll in cm", M_ZOLLCM
  MENUITEM "&cm in Zoll", M_CMZOLL
 }

 POPUP "&Temperatur"
 {
  MENUITEM "&Celsius in Fahrenheit", M_CELSIUSFAHRENHEIT
  MENUITEM "&Fahrenheit in Celsius", M_FAHRENHEITCELSIUS
 }
 POPUP "&Eigene Umrechnungen"
 {
  MENUITEM "&Hinzufügen", M_HINZUFUEGEN
 }
}
```

Der neue Menüpunkt soll ein eigenes Dialogfenster öffnen, in welchem der Anwender alle Daten, die für eine eigene Umrechnung benötigt werden, eingeben kann.
Um uns die Entwicklung erheblich zu vereinfachen, benutzen wir zur Erstellung des Dialoges einen Resourceneditor, dessen Ergebnis in etwa wie folgt aussehen sollte:



Wenn alles korrekt verlaufen ist und die Ungarische Notation eingehalten wurde, sieht der Dialog in der Resourcendatei (*.rc) wie folgt aus:

```
DLG_HINZUFUEGEN DIALOG 0, 0, 176, 155
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Eigene Umrechnung"
FONT 8, "MS Sans Serif"
{
CONTROL "", -1, "static", WS_CHILD | WS_VISIBLE |
        SS_ETCHEDFRAME, 7, 5, 162, 40
CONTROL "Einheit 1", -1, "static", WS_CHILD | WS_VISIBLE |
        SS_LEFT, 12, 12, 28, 8
CONTROL "", E_EINHEIT_1, "edit", WS_CHILD | WS_VISIBLE |
        WS_BORDER | WS_TABSTOP | ES_LEFT, 45, 9, 119, 14
CONTROL "Einheit 2", -1, "static", WS_CHILD | WS_VISIBLE |
        SS_LEFT, 12, 28, 28, 8
```

```
CONTROL "", E_EINHEIT_2, "edit", WS_CHILD | WS_VISIBLE |
        WS_BORDER | WS_TABSTOP | ES_LEFT, 45, 25, 119, 14
CONTROL "", -1, "static", WS_CHILD | WS_VISIBLE |
        SS_ETCHEDFRAME, 7, 45, 162, 35
CONTROL "Faktor", -1, "static", WS_CHILD | WS_VISIBLE |
         SS_LEFT, 12, 50, 28, 8
CONTROL "", E_FAKTOR, "edit", WS_CHILD | WS_VISIBLE |
        WS_BORDER | WS_TABSTOP | ES_LEFT, 45, 47, 63, 14
CONTROL "Multiplikation (sonst Division)", CK_MULT,
        "button", WS_CHILD | WS_VISIBLE | WS_TABSTOP |
        BS_AUTOCHECKBOX, 12, 65, 150, 10
CONTROL "", -1, "static", WS_CHILD | WS_VISIBLE |
        SS_ETCHEDFRAME, 7, 80, 162, 50
CONTROL "Offset", -1, "static", WS_CHILD | WS_VISIBLE |
        SS_LEFT, 12, 85, 28, 8
CONTROL "", E_OFFSET, "edit", WS_CHILD | WS_VISIBLE |
        WS_BORDER | WS_TABSTOP | ES_LEFT, 45, 82, 63, 14
CONTROL "Addition (sonst Subtraktion)", CK_ADD,
        "button", WS_CHILD | WS_VISIBLE | WS_TABSTOP |
        BS_AUTOCHECKBOX, 12, 100, 150, 10
CONTROL "Offset vor Faktor", CK_FIRST, "button", WS_CHILD |
        WS_VISIBLE | WS_TABSTOP | BS_AUTOCHECKBOX,
        12, 115, 150, 10
CONTROL "OK", IDOK, "button", WS_CHILD | WS_VISIBLE |
        WS_TABSTOP | BS_DEFPUSHBUTTON | BS_CENTER,
        7, 135, 45, 14
CONTROL "Abbrechen", IDCANCEL, "button", WS_CHILD |
        WS_VISIBLE | WS_TABSTOP | BS_PUSHBUTTON | BS_CENTER,
        124, 135, 45, 14
}
```

Im zugehörigen Headerfile wurde automatisch ergänzt:

```
#define DLG_HINZUFUEGEN     30000
#define E_EINHEIT_1         30001
#define E_EINHEIT_2         30002
#define E_FAKTOR            30003
#define CK_MULT             30004
#define E_OFFSET            30005
#define CK_ADD              30006
#define CK_FIRST            30007
```

Ein Dialog ist nichts anderes als eine Sonderform einer Client-Area. Das Besondere ist natürlich, dass ein Dialog ein übergeordnetes Fenster hat (die Client-Area), das Hauptfenster hingegen nicht. Da ein Dialog zudem eine benutzerdefinierte Bedeutung hat und daher kein standardisiertes Verhalten aufweisen kann (anders als ein Control), muss der Entwickler selbst für eine passende **WndProc** sorgen. Diese benennen wir nach dem Dialog (**DlgHinzufuegenProc**), so dass sofort erkennbar ist, zu welchem Dialog die Behandlungsfunktion gehört:

```
case  M_HINZUFUEGEN:
      {
         BOOL bResult;
         bResult = DialogBox((HINSTANCE)GetWindowLong(hWnd,
                   GWL_HINSTANCE),
                   MAKEINTRESOURCE(DLG_HINZUFUEGEN), hWnd,
                   (DLGPROC)DlgHinzufuegenProc);
         if (bResult)
         {
```

```
            InvalidateRect (hWnd, NULL, TRUE);
         }
      }
```

Der Aufruf der Funktion **DialogBox**$^{\nabla}$ erzeugt automatisch das entsprechende Fenster und zeigt es an. Die zur Dialogbox gehörende Behandlungsfunktion hat im Prinzip denselben Aufbau wie die Behandlungsfunktion des Hauptprogramms (**WndProc**). Einige Ausnahmen gibt es jedoch, so gibt es z.B. keine **WM_CREATE** Message, da diese von Windows bereits intern abgehandelt wird, um die mit dem Resourceneditor vorgegebenen und platzierten Controls zu erzeugen. Statt dessen erhält man nach der Erzeugung des Controls eine **WM_INITDIALOG** Message, in der man alle notwendigen Einstellungen (z.B. Vorbelegungen) vornehmen kann.
Minimal hat die Dialog Behandlungsfunktion das folgende Aussehen, wenn je ein „OK“ und ein „Abbrechen“ Knopf vorhanden sind:

```
BOOL WINAPI DlgHinzufuegenProc (HWND hDlg, UINT message,
                                WPARAM wParam, LPARAM lParam)
{
   switch (message)
   {
      case WM_INITDIALOG:
           break;

      case WM_COMMAND:
           switch (wParam)
           {
              case IDOK:
                   EndDialog (hDlg, TRUE);
                   break;

              case IDCANCEL:
                   EndDialog (hDlg, FALSE);
                   break;
           }
           break;
   }
   return FALSE;
}
```

Die beiden ID Nummern **IDOK** und **IDCANCEL** sind in Windows für genau diesen Zweck vordefiniert . Man sollte sie verwenden, auch wenn sie nicht der Ungarischen Notation entsprechen. Der Dialog kann nur beendet werden, indem abhängig von einem Ereignis die Funktion **EndDialog**$^{\nabla}$ aufgerufen wird.

Um die Eingaben zu speichern (und zu verwenden) müssen die Edit- und Checkbox-Controls beim Auslösen des OK-Buttons ausgewertet und auf die globalen Variablen kopiert werden. Durch den Aufruf von **InvalidateRect** für den Fall das TRUE zurückgegeben wird, werden die neuen Werte sofort benutzt.

Hierbei stellt sich aber ein Problem, um die Controls abzufragen, benötigen wir deren Handles. Diese besitzen wir aber nicht, da die Controls automatisch erzeugt wurden. das einzige Handle welches wir

in der Dialog Behandlungsfunktion kennen ist das Handle des Dialogen (**hDlg**), da es uns mit jedem Ereignis mitgeliefert wird.
Um diese Problem zu umgehen können wir Windows das Handle eines Controls in einem gegebenen Fenster oder Dialog anhand der ID-Nummer suchen lassen. Dazu dient die Funktion **GetDlgItem**[∇]:

```
HWND hEFaktor = GetDlgItem (hDlg, E_FAKTOR);
```

Diese Funktion kann bei Bedarf direkt in einem **SendMessage** Aufruf ausgeführt werden:

```
GetWindowText (GetDlgItem (hDlg, E_FAKTOR), sTemp, 100);
fFaktor = atof (sTemp);
```

Um uns die Sache etwas einfacher zu machen, brauchen uns die einzelnen Ereignisse der Controls aus unserem Dialog nicht zu interessieren. Die Inhalte sind nur schließlich nur von Belang, wenn der Button „OK“ gedrückt wurde.

Werten sie die Controls aus und sichern Sie die Werte in den globalen Variablen.

6.16 Schritt 16 – Laden der Umrechnungen aus einer INI-Datei

Jetzt, nachdem wir eigene Umrechnungen erstellen können, wäre es natürlich schön, wenn man diese nicht als festen Bestandteil in das Programm hineinschreiben müsste (wie wir es bisher in der **fnWMCommand** Funktion getan haben). Statt dessen wäre es von Vorteil, die Umrechnungen aus einer Datei zu lesen, denn dann könnten Erweiterungen in Form neuer Umrechnungen und Fehlerkorrekturen durch Konfiguration vorgenommen werden.
Der einfachste Mechanismus hierfür sind Windows INI-Dateien. Zwar sind INI-Dateien „offiziell“ von Microsoft durch die Registry abgelöst worden, für viele einfache Datenhaltungen sind INI-Dateien aber nach wie vor unproblematischer als Einträge in der Registrierungsdatenbank. So ist unser Umrechnungsprogramm z.B. bisher völlig unabhängig von irgendwelchen Systemeinträgen, ein einfaches Löschen des Anwendungsverzeichnisses mit allem Inhalt verbannt das Programm aus dem System, ohne dabei irgendwelche Spuren oder Reste zu hinterlassen. Würden wir die nachstehenden Werte statt in eine INI-Datei in die Registry schreiben, so würden sie dort verbleiben, oder wir müssten ein Installations- und Deinstallationsprogramm schreiben (bei dem aber auch keine Garantie besteht, dass der Anwender es auch benutzt).
Als erstes sollten wir uns darum kümmern, wo die INI-Datei steht. Gibt man in den Zugriffsfunktionen (**GetPrivateProfileString**[∇], **GetPrivateProfileInt**[∇] und **WritePrivateProfileString**[∇]) auf INI-Dateien nur einen Dateinamen an, so wird die INI-Datei automatisch im Windows Systemverzeichnis gesucht und ggf. geschrieben. Viel schöner ist es jedoch, wenn die INI-Datei im gleichen Verzeichnis steht wie das Anwendungsprogramm, dann könnte man bei Bedarf einfach

das gesamte Verzeichnis löschen, wie oben beschrieben. Dazu müssen wir aber zunächst feststellen, wo unsere Anwendung steht.
Glücklicherweise gibt es seit der Zeit der ersten C-Version die Konvention, dass jedes ausführbare Programm beim Start als Parameter seinen eigenen Namen inklusive vollständigem Pfad in der Variablen **argv[0]** übergeben bekommt. Diese Übergabe erfolgt an die **main** Funktion, an die wir in Windowsprogrammen aber nicht herankommen. Das in Windowsprogrammen verwendete **main** sichert die Parameter aber in einer globalen Struktur **_argv**, die wir statt dessen verwenden können. Alles was wir benötigen ist eine globale Variable zum Aufbau des Dateinamens:

```
#include <dos.h>
char sIni [MAXPATH];
```

Anschließend können wir in der **WinMain** Funktion die folgenden Zeilen hinzufügen :

```
strcpy (sIni, _argv[0]);
char *psHelp = strrchr (sIni, '\\');
if (psHelp)
{
   psHelp++;
   *psHelp = '\0';
   strcat (sIni, "Umrechnung.ini");
}
```

Anschließend legen wir die Datei „Umrechnung.INI“ mit dem folgenden Inhalt an:

```
[Umrechnungen]
Anzahl=2
1=Meter - Zentimeter
2=Dezimeter - Zentimeter

[Kategorien]
Anzahl=2
1=Längen
2=Temperaturen

[Meter - Zentimeter]
Kategorie=1
Einheit1=Meter
Einheit2=Zentimeter
Faktor=100.0
Multiplikation=1
Offset=0.0
Addition=0
OffsetZuerst=0

[Dezimeter - Zentimeter]
Kategorie=2
Einheit1=Dezimeter
Einheit2=Zentimeter
Faktor=10.0
Multiplikation=1
Offset=0.0
Addition=0
```

```
OffsetZuerst=0
```

Die Zeichenfolgen in eckigen Klammern werden „Sections“ genannt und gliedern die Inhalte in Form von Datensätzen. Alle Einträge (Attribute) in der INI-Datei bestehen aus Key-Value-Paaren, die durch das Gleichheitszeichen getrennt werden.
Auf einen Eintrag wird mittels der Funktionen **GetPrivateProfileString** und **GetPrivateProfileInt** lesend und mittels **WritePrivateProfileString** schreibend zugegriffen.
In der Funktion **fnWMCreate** können wir die Menüeinträge nun dynamisch zusammenstellen, statt die vorhandenen Umrechnungen in der Resourcedatei einzutragen. Dazu ändern wie den Resourceheader wie folgt ab:

```
#define M_FIRST             20000
#define M_LAST              20999
#define IDM_MENU1           20000
#define M_HINZUFUEGEN       22001
```

Beachten Sie bitte, dass **M_HINZUFUEGEN** eine neue Nummer außerhalb des für Menüeinträge reservierten Nummernkreises bekommen hat. Die ID Nummern für die bisher verwendeten Menüereignisse (**M_CMZOLL** etc.) entfallen.
Auch im Menüeintrag der Resourcedatei **(*.rc**) können wir erheblich vereinfachen:

```
IDM_MENU1 MENU
{
 POPUP "Eigene Umrechnungen"
 {
  MENUITEM "Hinzufügen", M_HINZUFUEGEN
 }
}
```

Die Menüeinträge werden über das Auslesen der INI-Datei ermittelt und dynamisch an das Menü angefügt. Als ID Nummer für die Menüeinträge verwenden wir die laufende Nummer innerhalb der INI-Datei plus des im Resource Header festgelegten Offsets **M_FIRST**.
Die in der INI-Datei festgelegten Kategorien bilden dabei die Überschriften im Menü. die einzelnen Umrechnungen werden anschließend anhand ihrer Kategorienummer an diese Einträge angehängt:

```
char sNum [20]  = "";
int nMaxEntries = GetPrivateProfileInt("Kategorien", "Anzahl",
                                       0, sIni);
HMENU hMenu;
for (int i=1; i<=nMaxEntries; i++)
{
   sprintf (sNum, "%d", i);
   GetPrivateProfileString("Kategorien", sNum, "unbekannt",
                           sTemp, 100, sIni);
   hMenu = CreateMenu ();
   AppendMenu((HMENU)GetWindowLong(hWnd, GWL_ID), MF_POPUP,
              (UINT)hMenu, sTemp);
```

```
}

int nKat;
nMaxEntries = GetPrivateProfileInt("Umrechnungen", "Anzahl",
                                   0, sIni);
for (int i=1; i<=nMaxEntries; i++)
{
   sprintf (sNum, "%d", i);
   GetPrivateProfileString("Umrechnungen", sNum, "unbekannt",
                           sTemp, 100, sIni);
   nKat = GetPrivateProfileInt(sTemp, "Kategorie", 1, sIni);
   hMenu = GetSubMenu((HMENU)GetWindowLong(hWnd,GWL_ID),nKat);
   AppendMenu (hMenu, 0, M_FIRST+i, sTemp);
}
```

Da dem Programm jetzt keine festen Konstanten für die einzelnen Menüeinträge mehr bekannt sind, ist die bisherige Form der Ereignisbehandlung nicht mehr haltbar. Statt dessen müssen wir in der **fnWMCommand** Funktion zunächst feststellen, ob ein Ereignis aus dem reservierten Menü Nummernkreis **M_FIRST** bis **M_LAST** stammt. Danach benötigen wir einen allgemeingültigen Mechanismus, um die Berechnungsparameter festzulegen. Dazu verwenden wir einen einfachen aber wirkungsvollen Trick, den wir in der INI-Datei bereits vorbereitet haben, indem wir die Section-Einträge mit den Berechnungsparametern genauso benannt haben wie die Texte für die Menü-Einträge. Für diesen Trick ermitteln wir aus der ID-Nummer des ausgelösten Menü-Eintrages den dazugehörigen Menütext und benutzen diesen, um die Berechnungsparameter aus der INI-Datei nachzuladen:

```
LRESULT fnWMCommand (HWND hWnd, WPARAM wParam, LPARAM lParam)
{
  WORD wID           = LOWORD(wParam);
  WORD wNotification = HIWORD(wParam);

  if ((wID >= M_FIRST) && (wID <= M_LAST))
  {
    char sTemp [101] = "";
    GetMenuString ((HMENU)GetWindowLong(hWnd, GWL_ID), wID,
                   sTemp, 100, MF_BYCOMMAND);
    LoadUmrechnung (sTemp);
    InvalidateRect (hWnd, NULL, TRUE);
  }
//...
```

Das nachladen übernimmt die Funktion **LoadUmrechnung**:

```
void LoadUmrechnung (char *sUmrechnung)
{
  char sTemp [100] = "";

  GetPrivateProfileString(sUmrechnung, "Einheit1", "???",
                          sTemp, 100, sIni);
  strcpy (sEinheit1, sTemp);

  GetPrivateProfileString(sUmrechnung, "Einheit2", "???",
                          sTemp, 100, sIni);
  strcpy (sEinheit2, sTemp);
```

```
  GetPrivateProfileString(sUmrechnung, "Faktor", "0.0", sTemp,
                          100, sIni);
  fFaktor = atof (sTemp);

  GetPrivateProfileString(sUmrechnung, "Offset", "0.0", sTemp,
                          100, sIni);
  fOffset = atof (sTemp);

  GetPrivateProfileString(sUmrechnung, "Multiplikation", "1",
                          sTemp, 100, sIni);
  bMultiply = atoi (sTemp) == 1;

  GetPrivateProfileString(sUmrechnung, "Addition", "1", sTemp,
                          100, sIni);
  bAddOffset = atoi (sTemp) == 1;

  GetPrivateProfileString(sUmrechnung, "OffsetZuerst", "1",
                          sTemp, 100, sIni);
  bOffsetFirst = atoi (sTemp) == 1;
}
```

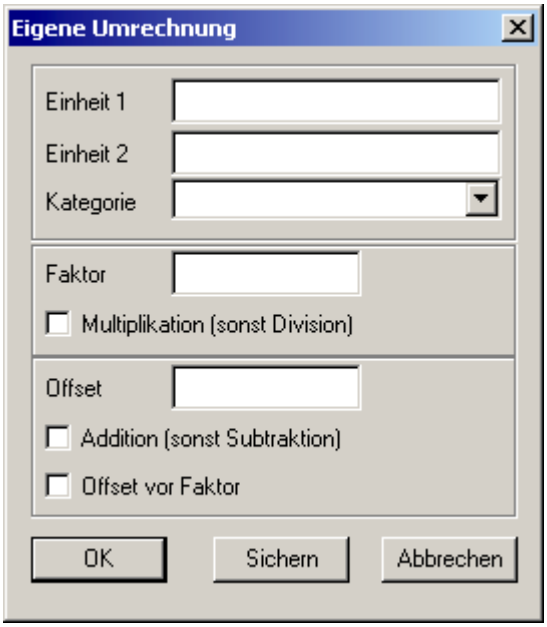
Nun gilt es noch die bisher in **fnWMCommand** fest vorgegebenen Umrechnungen in die INI-Datei zu verlagern und die Behandlungen der nicht mehr definierten Ereignisse **M_ZOLLCM**, **M_CMZOLL**, **M_CELSIUSFAHRENHEIT** und **M_FAHRENHEITCELSIUS** zu löschen.

Stellen Sie das Programm so um, dass alle Umrechnungen nur noch aus der INI-Datei gelesen werden..

6.17 Schritt 17 – Sichern neuer Umrechnungen

Nun, da wir die Umrechnungen aus einer Datei laden können, liegt es nah, neue Umrechnungen auch aus dem Programm heraus speichern zu können, so dass diesen beim nächsten Aufruf des Programms automatisch zur Verfügung stehen.
Dazu benötigen wir das Gegenstück zur **GetPrivateProfileString** Funktion, also die Funktion, die eine Einstellung in die INI-Datei schreibt. Diese Funktion trägt den Namen **WritePrivateProfileString** (eine **WritePrivateProfileInt** Funktion gibt es nicht).

Dazu müssen wir als erstes unseren Einstellungsdialog ein wenig erweitern, denn er muss alle Parameter schreiben können, die in der INI-Datei für einen Eintrag gebraucht werden. Dazu fehlt uns jetzt noch die Angabe der Kategorie und wir benötigen natürlich einen „Sichern“ Button, um die Speicherung auszulösen (es empfiehlt sich nicht, das Sichern automatisch mit dem „OK“ Button zu verknüpfen, denn ggf. möchte der Anwender zunächst die Parameter ausprobieren).
Der neue Dialog sieht dann wie folgt aus:



Das Popup mit den Kategorien kann man problemlos im WM_INITDIALOG Ereignis aus der INI-Datei befüllen und den ersten Eintrag auch gleich selektieren:

```
case WM_INITDIALOG:
{
   char sKat    [101] = "";
   char sNum    [20]  = "";
   int  nAnzahl;
   LRESULT nResult;

   nAnzahl = GetPrivateProfileInt ("Kategorien", "Anzahl", 0,
                                   sIni);
   for (int i=1; i<=nAnzahl; i++)
   {
      sprintf (sNum, "%d", i);
      GetPrivateProfileString("Kategorien", sNum, "unbekannt",
                              sKat, 100, sIni);
      nResult = SendMessage (GetDlgItem(hDlg, CB_KATEGORIE),
                             CB_ADDSTRING, (WPARAM)0,
                             (LPARAM)(LPCTSTR)sKat);
      SendMessage (GetDlgItem(hDlg, CB_KATEGORIE),
                   CB_SETITEMDATA, (WPARAM)nResult,
                   (LPARAM)(DWORD)i);
   }
   SendMessage (GetDlgItem(hDlg, CB_KATEGORIE), CB_SETCURSEL,
                (WPARAM)0, (LPARAM)0);
}
break;
```

Führen Sie einen zusätzlichen Save-Button ein, der die im Hinzufügen-Dialog gemachten Einstellungen in der INI-Datei sichert und den Dialog dann beendet.

6.18 Schritt 18 – Drucken

Nun will der Anwender die Ergebnisse meistens nicht nur auf dem Bildschirm sehen, sondern auch ausdrucken können.

Grundsätzlich ist dies in Windows nicht besonders schwierig zu realisieren, da auf dem Drucker genauso gezeichnet wird, wie auf dem Bildschirm. Einen guten Teil der Arbeit haben wir also schon getätigt.
Was wir zunächst benötigen, ist der Device-Context des Druckers, denn wenn die dargestellten Pixel 1:1 auf dem Drucker ausgegeben werden, ist das Ergebnis so klein, dass man eine Lupe benötigt. Ursache ist die wesentlich höhere Auflösung der Drucker gegenüber den Monitoren.
Um den Standarddrucker auszulesen und dessen Device-Context zurückzugeben, ist nur eine kurze Funktion nötig:

```
HDC fnGetPrinterDC (void)
{
   HDC hResult = 0;
   char szPrinter [80];
   char *szDevice, *szDriver, *szOutput;

   GetProfileString("windows","device", ",,,", szPrinter,80);
   if (NULL != (szDevice = strtok (szPrinter, "," )) &&
       NULL != (szDriver = strtok (NULL,      ", )) &&
       NULL != (szOutput = strtok (NULL,      ", )))
   {
      hResult = CreateDC (szDriver, szDevice, szOutput, NULL);
   }
   return hResult;
}
```

Außerdem benötigen wir noch eine zweite Funktion **fnPrintPage**, die etwas später erklärt wird.
Um die Druckausgabe anzustoßen fügen wir dem Menüpunkt „Eigene Umrechnungen“ einen weiteren Eintrag „Drucken“ hinzu:

```
IDM_MENU1 MENU
{
 POPUP "Eigene Umrechnungen"
 {
  MENUITEM "Hinzufügen", M_HINZUFUEGEN
  MENUITEM "Drucken",    M_DRUCKEN
 }
}
```

Diesen neuen Menüeintrag behandeln wir – wie gehabt – in der Funktion **fnWMCommand** ab. Die Reaktion auf die Auslösung des Menüpunktes ist recht einfach:

```
    case  M_DRUCKEN:
          if (fnPrintPage (hWnd))
          {
            MessageBox (NULL,
                        "Fehler beim Drucken der Seite!",
                        "Umrechnungen",
                        MB_OK | MB_ICONEXCLAMATION);
          }
          return 0;
```

Nun zur eigentlichen Druckfunktion **fnPrintPage**. Im Prinzip geschieht in dieser exakt das Gleiche wie in der Funktion **fnWMPaint**. Es sind

einige Vor- und Nachbereitungen zu treffen (in **fnWMPaint** reduzierten sich die Vor- und Nachbereitungen auf den Aufruf der Funktionen **BeginPaint** und **EndPaint**), die ein klein wenig aufwendiger als bei der Bildschirmausgabe sind. Dies liegt prinzipiell an der Tatsache, dass der Drucker seitenorientiert arbeitet, wohingegen der Bildschirm (spätestens beim Einsatz von Fenstern mit Scrollbalken) eine virtuell beliebige Größe hat. Außerdem muss auch die Druckwarteschlage (Drucker-Queue) sauber bedient werden.
Für die Funktion **fnPrintPage** stellt es sich so dar, dass zunächst der Drucker-Device-Context geholt wird. Die Funktion **fnGetPrinterDC** ist ausgelagert, da sie in dieser Form leicht in anderen Projekten wiederverwendet werden kann.
Dann wird eine **DocInfo**-Struktur angelegt und ausgefüllt, welche die Kommunikation mit der Drucker-Queue regelt. Die Komponente **lpszDocName** der **DocInfo**-Struktur enthält den Namen der in der Drucker-Queue angezeigt wird (hier könnte man etwas ausführlicher sein und z.B. die umgerechneten Einheiten mit angeben).
der Aufruf von **StartDoc**$^{\nabla}$ übergibt die Struktur an die Drucker-Queue, sofort im Anschluss daran starten wir eine neue Seite (Aufruf **StartPage**$^{\nabla}$).
Da wir am Device-Context des Druckers nun einige Änderungen vornehmen müssen, sichern wir den Inhalt des Device-Context ab. Dazu gibt es die Funktion **SaveDC**$^{\nabla}$, die Windowsintern dafür sorgt, dass alle zum DC gehörigen Daten kopiert werden. Dadurch kann der Aufbau eines Gerätespezifischen DC durch einen Gerätetreiber geändert werden, ohne dass das Programm angepasst werden muss.

Als nächstes legen wir fest, welcher Ausschnitt der zu malenden Seite wir ausgeben wollen. Dies ist meistens alles (wie hier dargestellt). Bei komplexeren Anwendungen wie z.B. CAD Programmen ist es aber häufig so, dass nur Teile gedruckt werden sollen. Die Bereiche können dann mit den Funktionen **SetWindowExtEx**$^{\nabla}$ (Größe des virtuellen Ausgabefensters auf dem Drucker setzen), **SetViewportExtEx**$^{\nabla}$ (Größe des darzustellenden Ausschnitts auf dem Drucker setzen) und **SetViewportOrgEx**$^{\nabla}$ (Ursprungskoordinaten auf dem Drucker setzen) festgelegt werden. Auf Details dieser komplexen Materie kann hier nicht näher eingegangen werden, bei Bedarf ziehen Sie bitte weiterführende Literatur zu Rate[1].
nach diesen Vorbereitungen wird fast genauso ausgegeben, wie in der Funktion **fnWMPaint**. Wir benötigen lediglich noch die ausgefüllte **TEXTMETRIC**-Struktur für den Drucker-Context. Die davon abhängigen Werte schreiben wir auf die globalen Variablen, die in **fnValueTable** benutzt werden. Da wir die globalen Werte aber für die Bildschirmausgabe anschließend wieder brauchen, müssen wir die alten Werte zwischenspeichern. Abschließend wird mit **RestoreDC**$^{\nabla}$ der gesicherte DC Zustand zurückgeholt

```
BOOL fnPrintPage (HWND hWnd)
```

[1] z.B. die Bücher von Charles Petzold über Windows Programmierung, die bei Microsoft Press erschienen sind.

```
{
   HDC     hdcPrn;
   DOCINFO aDocInfo;
   int     xPage, yPage;

   //---------------------------------------------------------
   // Device-Context des Druckers holen, wenn keiner verfügbar
   // ist, dann Fehler
   //---------------------------------------------------------
   hdcPrn = fnGetPrinterDC ();
   if (NULL == hdcPrn) return TRUE ;

   //---------------------------------------------------------
   // DocInfo-Struktur für die Drucker-Queue ausfüllen
   //---------------------------------------------------------
   ZeroMemory (&aDocInfo, sizeof (DOCINFO));
   aDocInfo.cbSize = sizeof (DOCINFO);
   aDocInfo.lpszDocName = "Umrechnungen";

   //---------------------------------------------------------
   // Druckvorgang mit neuer Seite starten
   //---------------------------------------------------------
   StartDoc (hdcPrn, & aDocInfo);
   StartPage (hdcPrn);

   SaveDC (hdcPrn);
   SetMapMode (hdcPrn, MM_ISOTROPIC);
   SetWindowExtEx   (hdcPrn, xPage, yPage, NULL);
   SetViewportExtEx (hdcPrn, xPage, yPage, NULL);
   SetViewportOrgEx (hdcPrn, 0, 0, NULL);

   //---------------------------------------------------------
   // Hier mach wir das gleiche, wie auf dem Bildschirm...
   //---------------------------------------------------------
   RECT aRectTitel;
   char sUeberschrift[255];

   int nOldSmallCharWidth = nSmallCharWidth;
   int nOldBigCharWidth   = nBigCharWidth;
   int nOldCharHeight     = nCharHeight;

   TEXTMETRIC aTM;
   GetTextMetrics (hdcPrn, &aTM) ;

   nCharHeight     = aTM.tmHeight + aTM.tmExternalLeading;
   nSmallCharWidth = aTM.tmAveCharWidth;
   nBigCharWidth   = (aTM.tmPitchAndFamily & TMPF_FIXED_PITCH
                      ? 3 : 2) * aTM.tmAveCharWidth / 2;

   GetClientRect (hWnd, &aRectTitel);
   aRectTitel.bottom = aRectTitel.top + 1.5 * nCharHeight;
   HBRUSH hBlueBrush = CreateSolidBrush (RGB (100, 100, 255));
   HFONT hFont      = CreateFont (nCharHeight, 0, 0, 0,
                            FW_THIN, FALSE, FALSE, FALSE,
                            ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                            CLIP_DEFAULT_PRECIS,
                            DEFAULT_QUALITY,
                            DEFAULT_PITCH, "Braggadocio");
   sprintf (sUeberschrift, "Umrechnung von %s nach %s",
            sEinheit1, sEinheit2);

   FillRect (hdcPrn, &aRectTitel, hBlueBrush);
   FrameRect(hdcPrn, &aRectTitel, (HBRUSH)GetStockObject
             (BLACK_BRUSH));
```

```
   HFONT hOldFont = (HFONT)SelectObject (hdcPrn, hFont);
   int nOldBk     = SetBkMode (hdcPrn, TRANSPARENT);
   COLORREF nOldTxtCol = SetTextColor (hdcPrn, RGB
                         (255,255,0));
   DrawText (hdcPrn, sUeberschrift, -1, &aRectTitel,
             DT_SINGLELINE|DT_CENTER|DT_VCENTER);
   SetTextColor (hdcPrn, nOldTxtCol);
   SetBkMode    (hdcPrn, nOldBk);
   SelectObject (hdcPrn, hOldFont);

   fnValueTable(hdcPrn);

   //-------------------------------------------------------
   // Limitierte Resourcen wieder freigeben
   //-------------------------------------------------------
   DeleteObject (hBlueBrush);
   DeleteObject (hFont);

   //-------------------------------------------------------
   // Seite und Druckvorgang beenden, Device-Context wieder
   // freigeben
   //-------------------------------------------------------
   RestoreDC (hdcPrn, -1);

   EndPage (hdcPrn);
   EndDoc  (hdcPrn);
   DeleteDC(hdcPrn);

   nSmallCharWidth = nOldSmallCharWidth;
   nBigCharWidth   = nOldBigCharWidth;
   nCharHeight     = nOldCharHeight;

   return 0;
}
```

Die nun erzielte Druckausgabe hat einige Schönheitsfehler:

- Die Ausgabe auf dem Drucker ist ein fast exaktes Abbild der Ausgabe auf dem Bildschirm, was leider auch zur Folge hat, dass das Papier nicht ausgenutzt wird und die Ausgabe auf dem Drucker abhängig von der Größe des Bildschirmfensters ist. Wir haben derzeit also ein Ergebnis, welches eher einem Screenshot als einem Druck ähnelt.
- Wir haben die Ausgabebefehle doppelt im Programm und damit doppelten Aufwand.
- Wir haben eine unangenehme Abhängigkeit von den globalen Variablen, die kurzzeitig überschrieben werden.

Diese Probleme zu lösen ist der nächste Schritt.

6.19 Schritt 19 – Druckausgabeoptimierung

Um die Druckausgabe elegant zu lösen ist es zunächst nötig, die Funktion **fnValueTable** komplett aus ihrer Abhängigkeit von solchen globalen Variablen zu lösen, deren Bezugsgröße der Bildschirm ist.

Alle graphischen Bildschirmausgaben sollten immer in Form von voll parametrisierten und wiederverwendbaren Funktionen erfolgen.

Dazu löschen wir die Deklarationen der Variablen **nAnzZeichen**, **nAnzZeilen**, **nSmallCharWidth**, **nBigCharWidth** und **nCharHeight**, da diese aus dem Bildschirm-Context errechnet werden. Statt dessen fügen wir die folgende Zeile wieder ein:

```
TEXTMETRIC aTM;
```

In der Funktion **fnWMCreate** entfernen wir die lokale **TEXTMETRIC** Struktur, da wir nun eine globale verwenden wollen. Außerdem entfernen wir die Berechnung der nicht mehr vorhanden Variablen **nCharHeight**, **nSmallCharWidth** und **nBigCharWidth**. Der geänderte Anfang der Funktion hat somit das folgende Erscheinungsbild:

```
LRESULT fnWMCreate (HWND hWnd, WPARAM wParam, LPARAM lParam)
{
  char sTemp [255] = "";
  int nTemp;
  LRESULT nIndex;
  HDC hDC = GetDC (hWnd) ;

  GetTextMetrics (hDC, &aTM) ;
  ReleaseDC (hWnd, hDC) ;

  hLHintergrund = CreateWindow ("static", "",
                  WS_CHILD | WS_VISIBLE,
                  0, 0, 0, 0, hWnd, (HMENU)L_HINTERGRUND,
                 (HINSTANCE)GetWindowLong (hWnd, GWL_HINSTANCE),
                  NULL);
//...
```

In der Funktion **fnWMSize** fügen wir die Variablen **nCharHeight** und **nSmallCharWidth** lokal hinzu und berechnen sie aus der globalen **TEXTMETRIC** Struktur.

```
int nCharHeight     = aTM.tmHeight + aTM.tmExternalLeading;
int nSmallCharWidth = aTM.tmAveCharWidth;
```

Da die Variablen nAnzZeichen und nAnzZeilen nicht mehr global existieren, nehmen wir ihre Berechnung hier heraus. An den Stellen, an denen wir diese Informationen benötigen, berechnen wir sie bei Bedarf. Hier der geänderte Teil der Funktion **fnWMSize**:

```
LRESULT fnWMSize (HWND hWnd, WPARAM wParam, LPARAM lParam)
{
  int nCharHeight     = aTM.tmHeight + aTM.tmExternalLeading;
  int nSmallCharWidth = aTM.tmAveCharWidth;
  int nActWidth = LOWORD(lParam);
  int nMinWidth = 105 * nSmallCharWidth;
  int nActHeight= HIWORD(lParam);
  int nMinHeight=  7 * nCharHeight;
  int nNewWidth = nActWidth<nMinWidth ? nMinWidth : nActWidth;
  int nNewHeight= nActHeight<nMinHeight?nMinHeight:nActHeight;

  //-------------------------------------------------------
  // abhänge Unterfenster skalieren
  //-------------------------------------------------------
```

```
//...
```

Nun die eigentliche Verallgemeinerung. Wir verlagern alle Ausgabe (und die zugehörigen lokalen Variablen) aus **fnWMPaint** in die Funktion **fnValueTable**, wobei wir zudem die Größe des Ausgabebereiches (**GetClientRect** für den Bildschirm) und die zu verwendende **TEXTMETRIC** Struktur für die Fontgrößen als Parameter übergeben.

```
void fnValueTable (HDC hDC, RECT& aRect,
                   TEXTMETRIC &aTextMetric)
{
   int      nLine, nPos, j = 0;
   double   f = fStartwert;
   char     sText [100]    = "";
   UINT     nOldSettings = SetTextAlign(hDC,TA_RIGHT|TA_TOP);
   HPEN     hPen, hOldPen;
   int      nStellen = 2*nStellenVK+5;
   RECT     aRectTitel = aRect;
   HBRUSH   hBlueBrush;
   HFONT    hFont, hOldFont;
   char     sUeberschrift [2*EINHEITTEXTLEN + 50];
   int      nCharHeight = aTextMetric.tmHeight +
                          aTextMetric.tmExternalLeading;
   int      nSmallCharWidth = aTextMetric.tmAveCharWidth;
   int      nAnzZeichen = aRect.right  / nSmallCharWidth - 2;
   int      nAnzZeilen  = aRect.bottom / nCharHeight     - 4;
   int      nOldBk;
   COLORREF nOldTxtCol;

   //-------------------------------------------------------
   // Ausgabe der Titelzeile
   //-------------------------------------------------------
   aRectTitel.bottom = aRectTitel.top + 1.5 * nCharHeight;
   hBlueBrush = CreateSolidBrush (RGB (100, 100, 255));
   hFont      = CreateFont (nCharHeight, 0, 0, 0, FW_THIN,
                            FALSE, FALSE, FALSE,
                            ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                            CLIP_DEFAULT_PRECIS,
                            DEFAULT_QUALITY,
                            DEFAULT_PITCH, "Braggadocio");
   sprintf (sUeberschrift, "Umrechnung von %s nach %s",
            sEinheit1, sEinheit2);

   FillRect (hDC, &aRectTitel, hBlueBrush);
   FrameRect(hDC, &aRectTitel,
             (HBRUSH)GetStockObject (BLACK_BRUSH));

   hOldFont   = (HFONT)SelectObject (hDC, hFont);
   nOldBk     = SetBkMode (hDC, TRANSPARENT);
   nOldTxtCol = SetTextColor (hDC, RGB (255,255,0));
   DrawText (hDC, sUeberschrift, -1, &aRectTitel,
             DT_SINGLELINE|DT_CENTER|DT_VCENTER);
   SetTextColor (hDC, nOldTxtCol);
   SetBkMode    (hDC, nOldBk);
   SelectObject (hDC, hOldFont);

   DeleteObject (hBlueBrush);
   DeleteObject (hFont);

   //-------------------------------------------------------
   hPen    = CreatePen (PS_SOLID, 1, RGB (0, 0, 0));
   hOldPen = (HPEN)SelectObject (hDC, hPen);
```

```
   while (((j+1)*nStellen) < nAnzZeichen)
   {
      nPos = nSmallCharWidth + j * nStellen * nSmallCharWidth;
      for (nLine=0; nLine<nAnzZeilen; f+=fSchrittweite,
           nLine++)
      {
         sprintf (sText, "%*.*lf =",
                  nStellenVK, bStellenNK ? nStellenNK : 0, f);
         TextOut (hDC, nPos+(nStellenVK+2)*nSmallCharWidth,
                  (nLine+2)*nCharHeight, sText,
                  strlen (sText));

         sprintf (sText, "%*.*lf", nStellenVK,
                  bStellenNK ? nStellenNK : 0,
                  fnCalcValue(f, fFaktor, bMultiply, fOffset,
                              bAddOffset, bOffsetFirst));
         TextOut (hDC, nPos+(nStellen-2)*nSmallCharWidth,
                  (nLine+2)*nCharHeight, sText, strlen
                  (sText));
      }

      MoveToEx (hDC, j*nStellen*nSmallCharWidth,
                2*nCharHeight, NULL);
      LineTo (hDC, j*nStellen*nSmallCharWidth,
              (nLine+2)*nCharHeight);
      j++;
   }
   SelectObject (hDC, hOldPen);
   SetTextAlign (hDC, nOldSettings);
   DeleteObject (hPen);
}
```

Den Prototyp für **fnValueTable** ändern wir entsprechend ab.

```
void fnValueTable (HDC hDC, RECT& aRect,
                   TEXTMETRIC &aTextMetric) ;
```

Die Funktion **fnWMPaint** wird dadurch wesentlich übersichtlicher und allein auf die notwendigen Vor- und Nacharbeiten der Bildschirmausgabe reduziert:

```
LRESULT fnWMPaint (HWND hWnd, WPARAM wParam, LPARAM lParam)
{
   HDC         hDC ;
   PAINTSTRUCT aPS ;
   RECT        aRectTitel;

   GetClientRect (hWnd, &aRectTitel);

   hDC = BeginPaint (hWnd, &aPS);
   fnValueTable(hDC, aRectTitel, aTM);
   EndPaint (hWnd, &aPS);
   return 0;
}
```

Jetzt können wir auch die **fnPrintPage** Funktion um die nach **fnValueTable** verlagerten Ausgaben reduzieren. Für die zu übergebende **RECT** Struktur benötigen wir aber noch die Größe des Druckblattes aus dem Device-Context des Druckers:

```
   xPage = GetDeviceCaps (hdcPrn, HORZRES) ;
   yPage = GetDeviceCaps (hdcPrn, VERTRES) ;
```

Insgesamt sieht die Funktion jetzt so aus:

```
BOOL fnPrintPage (HWND hWnd)
{
   HDC     hdcPrn;
   DOCINFO aDocInfo;
   int     xPage, yPage;
   RECT    aRect = {0, 0, 0, 0};
   TEXTMETRIC aPrintTM;

   //-------------------------------------------------------
   // Device-Context des Druckers holen, wenn keiner verfügbar
   // ist, dann Fehler
   //-------------------------------------------------------
   hdcPrn = fnGetPrinterDC ();
   if (NULL == hdcPrn) return TRUE ;

   //-------------------------------------------------------
   // Auflösung des Druckers ermitteln
   //-------------------------------------------------------
   xPage = GetDeviceCaps (hdcPrn, HORZRES) ;
   yPage = GetDeviceCaps (hdcPrn, VERTRES) ;

   //-------------------------------------------------------
   // DocInfo-Struktur für die Drucker-Queue ausfüllen
   //-------------------------------------------------------
   ZeroMemory (&aDocInfo, sizeof (DOCINFO));
   aDocInfo.cbSize = sizeof (DOCINFO);
   aDocInfo.lpszDocName = "Umrechnungen";

   //-------------------------------------------------------
   // Druckvorgang mit neuer Seite starten
   //-------------------------------------------------------
   StartDoc (hdcPrn, &aDocInfo);
   StartPage (hdcPrn);

   SaveDC (hdcPrn);
   SetMapMode (hdcPrn, MM_ISOTROPIC);
   SetWindowExtEx   (hdcPrn, xPage, yPage, NULL);
   SetViewportExtEx (hdcPrn, xPage, yPage, NULL);
   SetViewportOrgEx (hdcPrn, 0, 0, NULL);

   aRect.bottom = yPage;
   aRect.right = xPage;

   GetTextMetrics (hdcPrn, &aPrintTM) ;
   fnValueTable(hdcPrn, aRect, aPrintTM);

   //-------------------------------------------------------
   // Seite und Druckvorgang beenden, DC wieder freigeben
   //-------------------------------------------------------
   RestoreDC (hdcPrn, -1);

   EndPage (hdcPrn);
   EndDoc  (hdcPrn);
   DeleteDC(hdcPrn);

   return 0;
}
```

6.20 Schritt 20 – Zwischenablage

Als letztes wollen wir die Zwischenablage von Windows mit einem der umgerechneten Werte bedienen. Am sinnvollsten ist es, den umgerechneten Wert in die Zwischenablage zu kopieren, der sich unter dem Mauszeiger befindet.
Als Auslöser wählen wir das loslassen der rechten Maustaste. Dieses wird von Windows in Form einer **WM_RBUTTONUP** Message signalisiert. Die Message behandeln wir in der gewohnten Form durch eine eigene Funktion:

```
case WM_RBUTTONUP: return fnWMRButtonUp(hWnd, wParam, lParam);
```

Für uns in geradezu ideal, enthält die Message im **LPARAM** die relative Position des Mauszeigers zum linken oberen Rand des Fensters.
Somit können wir die Mausposition (in Pixeln) leicht ermitteln:

```
LRESULT fnWMRButtonUp(HWND hWnd, WPARAM wParam, LPARAM lParam)
{
  int     xPos = LOWORD(lParam);  // horizontale Position
  int     yPos = HIWORD(lParam);  // vertikale Position
  //...
```

Anschließend erfolgt einiges an Rechnerei, denn aus der Pixelposition muss wieder auf den umzurechnenden Wert geschlossen werden, dies geschieht, indem wir berechnen in welcher Spalte und Zeile sich der Mauszeiger befindet. Die meisten der nötigen Rechenoperationen kennen wir bereits aus der **fnValueTable** Funktion.
Sowie ermittelt wurde, welcher umzurechnende Wert sich unter dem Mauszeiger befinden muss, können wir für diesen Wert die Umrechnungsfunktion aufrufen und erhalten den umgerechneten Wert.
Diesen Wert wandeln wir mit **sprintf** in eine formatierte Zeichenkette um (unter Berücksichtigung der angegebenen Nachkommastellen) und hängen noch den Inhalt von **sEinheit2** an:

```
sprintf (sText, "%.*lf %s", bStellenNK ? nStellenNK : 0,
         fResult, sEinheit2);
```

Diese Zeichenkette kann nun in die Zwischenablage kopiert werden.
Da die Zwischenablage allen Programmen gemeinsam zugänglich ist, gehört sie nicht zu einem Programm, sondern wird von Windows in einem eigenen Datensegment verwaltet. Ein Segment der benötigten Größe (hier 100 Zeichen) versuchen wir zunächst über die Funktion **GlobalAlloc**$^{\nabla}$ von Windows zu erhalten. Da wir die Konstante **GMEM_MOVEABLE** angeben, darf Windows bei Bedarf den Speicherblock mit unserem Textinhalt im Speicher verschieben (dies ist bei unseren kleinen Texten zwar unwahrscheinlich, sollte aber immer angegeben werden, insbesondere wenn mit großen Speicherblöcken, wie z.B. Bildern gearbeitet wird).
Bevor wir mit dem Zwischenablage-Speicher arbeiten können, muss man das Handle in einen Pointer umwandeln. Wenn wir aber mit einem Zeiger arbeiten, müssen wir verhindern, dass Windows zwischendurch

den Inhalt der Zwischenablage verschiebt, denn dann würde der Zeiger auf einen ungültigen Bereich zeigen. Daher vermerkem wir den Speicherblock bis zum Abschluss unserer Arbeit über **GlobalLock**$^{\nabla}$ als gesperrt.
Da wir den Zeiger auf den Speicherbereich von **GlobalLock** erhalten haben, können wir jetzt den Text hineinschreiben.
Jetzt öffnen wir das Clipboard mit **OpenClipboard**$^{\nabla}$. Der Vorgang macht den Clipboard-Zugriff exklusiv, dadurch ist es für alle anderen Anwendungen solange nicht mehr zugänglich, bis wir es wieder geschlossen haben.
Als erstes lassen wir den bisherigen Inhalt durch **EmptyClipboard**$^{\nabla}$ löschen und geben damit auch den dafür reservierten Speicher wieder frei. Dafür schreiben wir unseren Text durch Übergabe des globalen Handles in das Clipboard hinein. Auf gar keinen Fall darf man vergessen das CLIPBOARD wieder zu schließen, da es sonst erst nach Beendigung unseres Programms wieder zugänglich wäre und andere Programme ggf. blockiert.
Die komplette Clipboard Programmsequenz sieht wie folgt aus:

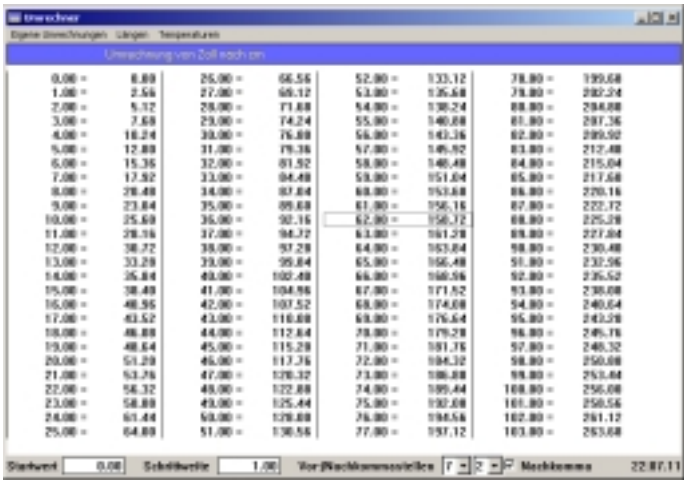
```
hData = GlobalAlloc (GMEM_MOVEABLE, 100L);
if (hData)
{
   pData = (LPSTR)GlobalLock (hData);
   if (!pData)
   {
      GlobalFree (hData);
   }
   else
   {
      GlobalUnlock (hData);
      strcpy (pData, sText);
      if (OpenClipboard (hWnd))
      {
         EmptyClipboard ();
         SetClipboardData (CF_TEXT, hData);
         CloseClipboard ();
      }
      else
      {
         GlobalFree(hData);
      }
   }
}
```

Um zu zeigen, dass etwas geschehen ist, zeichnen wir ein graues Rechteck um den kopierten Wert. Das Rechteck **aChosen** definieren wir global. Die folgenden Zeilen werden am besten hinter **CloseClipboard**$^{\nabla}$ eingefügt:

```
InvalidateRect (hWnd, &aChosen, TRUE);
aChosen.left  = ((nAktSpalte-1)*
                 (nAusgabeZeichen+2)+1)*nSmallCharWidth;
aChosen.right = ((nAktSpalte)  *
                 (nAusgabeZeichen+2))  *nSmallCharWidth;
aChosen.top   = (nAktZeile+2)* nCharHeight;
aChosen.bottom= (nAktZeile+3)* nCharHeight;
InvalidateRect (hWnd, &aChosen, TRUE);
```

Der **InvalidateRect** Befehl wird zweimal ausgeführt, einmal mit den alten und dann mit den neuen Einstellungen. Der erste Aufruf führt dazu, dass das die vorherige Rechteckmarkierung wieder zurückgenommen wird.
Das eigentliche Zeichnen des Rechtecks lassen wir in einer eigenen Funktion ausführen, die wir nach der **fnValueTable** Funktion in **fnWMPaint** ausführen. In **fnPrintPage** hingegen hat dieser Aufruf nichts zu suchen, schließlich ist der Wert, der in der Zwischenablage gespeichert ist, auf einem Ausdruck nicht von Interesse. Das ist auch der Grund die Anweisungen von fnFrameChosen getrennt zu halten und nicht – wie all die anderen Anweisungen zum Zeichnen der Tabelle – in **fnValueTable** durchzuführen.

```
void fnFrameChosen (HDC hDC)
{
   FrameRect(hDC, &aChosen,
   (HBRUSH)GetStockObject (GRAY_BRUSH));
}
```



7 Syntax A-Z

Nachfolgend die Syntaxbeschreibungen der im Programm verwendeten Befehle. Die Syntaxbeschreibungen sind alphabetisch sortiert.

7.1 Windowsfunktionen – A

```
BOOL AppendMenu (HMenu hMenu, UINT uFlags,
                 UINT_PTR uIDNewItem,
                 LPCTSTR lpNewItem);
```

Parameter:

- hMenu: Handle des Menüs an das ein neuer Menüpunkt angehängt werden soll.
- uFlags: Flags, die das Verhalten des Menüeintrags bestimmen (u.a. MF_CHECKED, MF_UNCHECKED, MF_DISABLED, MF_ENABLED und MF_GRAYED).
- uIDNewItem: ID des neuen Menü-Eintrags
- lpNewItem: Adresse des Textes des neuen Menü-Eintrags.

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

7.2 Windowsfunktionen – B

```
HDC BeginPaint (HWND hWnd, LPPAINTSTRUCT lpPaint);
```

Parameter:

- hWnd: Handle des Fensters, auf dem gezeichnet werden soll.
- lpPaint: Adresse einer Paint-Struktur, in der Informationen zum Vorgang des Zeichnens abgelegt werden.

Returnwert:
Device-Context des Gerätes, auf dem gezeichnet wird. Dieser Wert wird in den meisten Ausgabefunktionen benötigt.

7.3 Windowsfunktionen – C

```
BOOL CloseClipboard (VOID);
```

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

```
HDC CreateDC (LPCTSTR sDriver, LPCTSTR sDevice,
              LPCTSTR sOutput, CONST DEVMODE *lpInitData);
```

Parameter:

- sDriver: Name des zu verwendenden Gerätetreibers.
- sDevice: Name des zu verwendenden Gerätes.
- sOutput: NICHT VERWENDET IMMER NULL;
- lpInitData: Zeiger auf optionale Druckerdaten.

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war (Device-Context). Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

```
HFONT CreateFont (int nHeight, int nWidth, int nEscapement,
                  int nOrientation, int nWeight,
                  DWORD dwItalic, DWORD dwUnderline,
                  DWORD dwStrikeOut, DWORD dwCharset,
                  DWORD dwOutputPrecision,
                  DWORD dwClipPrecision, DWORD dwQuality,
                  DWORD dwPitchAndFamily, LPCTSTR lpszFace);
```

Parameter:

- nHeight: Schriftgröße in logischen Einheiten (em-Höhe)
- nWidth: Schriftbreite in logischen Einheiten. Wird Null übergeben, so wird automatisch eine zur Höhe passende Breite gewählt
- nEscapement: Winkel der Basislinie gegenüber der Horizontalen in zehntel Grad
- nOrientation: Winkel der Basislinie eines Zeichens gegenüber der Horizontalen des Device in zehntel Grad
- nWeight: Flag des typographischen Font-Gewichts (Namen in Klammern sind alternative Namen mit gleicher Bedeutung): FW_DONTCARE, FW_THIN, FW_EXTRALIGHT, FW_ULTRALIGHT, FW_LIGHT, FW_NORMAL (FW_REGULAR), FW_MEDIUM, FW_SEMIBOLD (FW_DEMIBOLD), FW_BOLD, FW_EXTRABOLD (FW_ULTRABOLD), FW_HEAVY (FW_BLACK)
- dwItalic: Flag (TRUE/FALSE) ob Schrift Kursiv dargestellt werden soll
- dwUnderline: Flag (TRUE/FALSE) ob Schrift unterstrichen dargestellt werden soll
- dwStrikeout: Flag (TRUE/FALSE) ob Schrift durchgestrichen dargestellt werden soll
- dwCharSet: Flag des zu verwendenden Zeichensatzes (hier ANSI_CHARSET), nur von Interesse, wenn Schriftsätze mit im ANSI-Satz nicht enthaltenen Zeichen benötigt werden (z.B. Russisch, Thai, Arabisch)
- dwOutputPrecision: Flagfeld für Einstellungen, wie aus ggf. mehreren vorhandenen Fonts gewählt werden soll. OUT_DEFAULT_PRECIS ist bis auf Ausnahmefälle korrekt.
- dwClipPrecision: Flagfeld des Clipping-Verhaltens, z.Zt. nur interessant wenn Dokumente mit eingeschlossenen (embedded) Fonts verarbeitet werden, sonst CLIP_DEFAULT_PRECIS
- dwQuality: Flagfeld der Berechnungsqualität des Fonts. Für Bildschirm- und Druckausgabe ist DEFAULT_QUALITY völlig ausreichend. Aternative Werte sind: ANTIALIASED_QUALITY, NONANTIALIASED_QUALITY, DEFAULT_QUALITY, DRAFT_QUALITY, PROOF_QUALITY
- dwPitchAndFamily: Dieses Flagfeld zerfällt in zwei Bereiche. Die beiden low-order Bits werden verwendet um festzulegen ob es sich um einen Font fixer oder variabler Länge handelt: DEFAULT_PITCH, FIXED_PITCH, VARIABLE_PITCH wird hier DEFAULT_PITCH angegeben, so entnimmt Windows die entsprechende Information aus dem TrueType Font. Die oberen vier Bits werden verwendet, um generelle Informationen über den Font zu geben, dies erleichtert Windows die Auswahl einer alternativen Schriftart, wenn der angegebene Font auf dem System nicht installiert ist:
  FF_DECORATIVE (Schmuckschriftart, wie z.B. Old English)
  FF_DONTCARE (unwichtig oder unbekannt), FF_MODERN (Schriftart mit fixer Breite und ohne Serifen wie z.B. Courier)
  FF_ROMAN (Fonts mit variabler Länge und Serifen wie z.B. Times Roman)
  FF_SCRIPT (handschriftartige Fonts), FF_SWISS (Schriftart mit variabler Breite und ohne Serifen wie z.B. SansSerif)

- lpszFace: Name der Schriftart, wie sie in den Font-Auswahlfeldern des Systems angeboten wird (nicht der Name auf der Festplatte).

Returnwert:
Handle auf den Font, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

HMENU CreateMenu (VOID);

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war (Menü-Handle). Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

HPEN CreatePen (int nPenStyle, int nWidth, COLORREF crColor);

Parameter:
- nPenStyle: Flags die das Aussehen der gezogenen Linie bestimmen (u.a. PS_SOLID, PS_DASH etc.)
- nWidth: Breite der Linie, die mit dem Pen gezogen wird.
- crColor: Farbe der Linie, die mit dem Pen gezogen wird

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war (Pen-Handle). Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

HBRUSH CreateSolidBrush (COLORREF crColor);

Parameter:
- crColor: Farbe des Brushes

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war (Brush-Handle). Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

```
HWND CreateWindow (LPCTSTR sClassname, LPCTSTR sWindowname,
                   DWORD dwStyle, int x, int y, int nWidth,
                   int nHeight, HWND hWndParent, HMENU hMenu,
                   HINSTANCE hInst, LPVOID lpParam);
```

Parameter:
- sClassname: Name einer registrierten Fensterklasse.
- sWindowname: Fenstertitel
- dwStyle: Styleflags des Fensters. Die gültigen Flags sind abhängig von der Fensterklasse des zu erzeugenden Fensters. Allgemeingültige Styleflags beginnen mit WS_.
- x: horizontale Position des Fensters.
- y: vertikale Position des Fensters.
- nWidth: Breite des Fensters.
- nHeight: Höhe des Fensters.
- hWndParent: Handle des übergeordneten Fensters.
- hMenu: Handle des zum Fenster gehörigen Menüs bzw. ID-Nummer, wenn es sich beim zu erzeugenden Fensters um ein Control handelt.
- hInst: Handle der Programminstanz.
- lpParam: Zeiger auf ggf. vorhandene Fensterzusatzdaten.

Rückgabewert:

Ungleich Null, wenn die Funktion erfolgreich war (Window-Handle). Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

7.4 Windowsfunktionen – D

LRESULT DefWindowProc (HWND hWnd, UINT nMsg, WPARAM wParam, LPARAM lParam);

Parameter:
- hWnd: Handle des Fensters, für welches die Botschaft empfangen wurde.
- nMsg: ID der empfangenen Message
- wParam: erster Parameter. Typ und Parameter sind abhängig von der Message-ID.
- lParam: zweiter Parameter. Typ und Parameter sind abhängig von der Message-ID.

Rückgabewert:
Ist abhängig von der verarbeiteten Message.

BOOL DeleteDC (HDC hDC);

Parameter:
- hDC: Device-Context der zu löschen ist.

Returnwert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

BOOL DeleteObject (HGDIOBJ hObject);

Parameter:
- hObject: Zeichenobjekt (limitierte Resource) das zu löschen ist.

Returnwert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

int DialogBox (HINSTANCE hInst, LPCTSTR lpTemplate, HWND hParent, DLGPROC lpDlgFunc) ;

Parameter:
- hInst: Das Instanzenhandle des Programms (wird hier aus dem übergeordneten Fenster entnommen
- lpTemplate: Die Dialog Resource (Makroaufruf)
- hParent: Das Handle des übergeordneten Fensters
- lpDlgFunc: Die Behandlungsfunktion für die Verarbeitung von Ereignissen.

Returnwert:
Der Wert, der als Rückgabewert in **EndDialog** spezifiziert wird.

int DrawText (HDC hDC, LPCTSTR lpString, int nCount, LPRECT lpRect, UINT uFormat);

Parameter:
- hDC: Device-Context des Gerätes auf dem der Text auszugeben ist.

- lpString: Adresse der Zeichenkette die auszugeben ist.
- nCount: Länge der auszugebenden Zeichenkette.
- lpRect: Adresse einer Rechteck-Struktur (RECT). Dies ist der Ausschnitt auf dem die Ausgabe erfolgt.
- uFormat: Style-Flags der Ausgabe (siehe Tabelle).

Returnwert:
Ungleich Null, wenn die Funktion erfolgreich war (dann Höhe der Ausgabe). Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

7.5 Windowsfunktionen – E

BOOL EmptyClipboard (VOID);

Returnwert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

BOOL EndDialog (HWND hDlg, int nResult);

Parameter:
- hDlg: Fensterhandle des zu schließenden Dialoges
- nResult: Ergebniswert, der an das übergeordnete Fenster zurückgeliefert wird.

Returnwert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

int EndDoc (HDC hDC);

Parameter:
- hDC: Device-Context des Druckers, dessen Dokumentausgabe beendet werden soll.

Rückgabewert:
Größer Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null oder kleiner Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

int EndPage (HDC hDC);

Parameter:
- hDC: Device-Context des Druckers, dessen Datenübertragung für den Druck hiermit beendet wird.

Rückgabewert:
Größer Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null oder kleiner Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

BOOL EndPaint (HWND hWnd, LPPAINTSTRUCT lpPaint);

Parameter:
- hWnd: Handle des Fensters, auf dem gezeichnet wurde.
- lpPaint: Adresse einer Paint-Struktur, in der Informationen zum Vorgang des Zeichnens abgelegt werden.

Returnwert:

Immer Null.

7.6 Windowsfunktionen – F

int FillRect (HDC hDC, CONST RECT *lpRect, HBRUSH hBR);

Parameter:
- hDC: Device-Context des Gerätes auf dem gezeichnet werden soll.
- lpRect: Das mit Farbe zu füllende Rechteck. Hier kann ein von Programmierer berechnete Rechteck in Form einer **RECT** Struktur angegeben werden.
- hBR: Brush (bestimmt Füllfarbe und Füllmuster)

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

int FrameRect (HDC hDC, CONST RECT *lpRect, HBRUSH hBR);

Parameter:
- hDC: Device-Context des Gerätes auf dem gezeichnet werden soll.
- lpRect: Das zu zeichnende Rechteck. Hier kann ein von Programmierer berechnete Rechteck in Form einer **RECT** Struktur angegeben werden.
- hBR: Brush (bestimmt Füllfarbe und Füllmuster)

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

7.7 Windowsfunktionen – G

int GetBkMode (HDC hDC);

Parameter:
- hDC: Device-Context, dessen Hintergrundmodus ausgewertet werden soll.

Rückgabewert:
Ungleich Null (OPAQUE oder TRANSPARENT), wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

BOOL GetClientRect (HWND hWnd, LPRECT lpRect);

Parameter:
- hWnd: Handle des Fensters, dessen Ausmaße zu ermitteln sind.
- lpRect: Adresse einer Rechteck-Struktur (RECT) in welche die Ausmaße eingetragen werden

Returnwert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

HDC GetDC (HWND hWnd);

Parameter:
- hWnd: Handle des Fensters, für welches der Device-Context geholt werden soll.

Rückgabewert:
Null wenn ein Fehler auftrat. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

int GetDeviceCaps (HDC hDC, int nIndex);

Parameter:
- hDC: Device-Context, dessen Informationen ausgewertet werden sollen.
- nIndex: Codenummer der abzufragenden Information. Die abfragbaren Informationen sind abhängig vom Gerätetyp.

Rückgabewert:
Abhängig vom abgefragten Wert.

HWND GetDlgItem (HWND hDlg, int nIDDlgItem);

Parameter:
- hDlg: Handle des Fensters, welches das zu ermittelnde Control enthält.
- nIDDlgItem: ID-Nummer des zu ermittelnden Controls.

Rückgabewert:
Ungleich Null (Handle des Controls), wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

DWORD GetLastError(VOID);

Returnwert:
Fehlernummer der letzten, fehlerhaften Operation. Wird von vielen Windowsfunktionen gesetzt.

HWND GetNextDlgTabItem (HWND hDlg, HWND hCtl, BOOL bPrevious);

Parameter:
- hDlg: Handle des Fensters, welches das Control enthält.
- hCtl: Handle des Controls, dessen Nachfolger oder Vorgänger (mit gesetztem WS_TABSTOP Flag) gesucht wird.
- bPrevious: entscheidet ob Vorgänger oder Nachfolger gesucht wird.

Rückgabewert:
Ungleich Null (Handle des gesuchten Controls), wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

```
UINT GetPrivateProfileInt (LPCTSTR sSectionName,
                           LPCTSTR sKeyName,
                           INT     nDefaultErgebnis,
                           LPCTSTR sIniDateiName);
```

Parameter:
- sSectionName: Name einer Section (Text in eckigen Klammern) in der INI-Datei

- sKeyName: Name eines Schlüsselwertes in der Section (Text vor dem Zeichen =)
- nDefaultErgebnis: Wert der zurückgegeben wird, wenn der Schlüsselwert nicht gefunden wurde.
- sIniDateiName: Pfad und Dateiname der INI-Datei.

Rückgabewert:
Der gelesene Wert.

```
DWORD GetPrivateProfileString (LPCTSTR sSectionName,
                               LPCTSTR sKeyName,
                               LPCTSTR sDefaultErgebnis,
                               LPTSTR  sErgebnisPuffer,
                               DWORD   nErgebnisPufferGroesse,
                               LPCTSTR sIniDateiName);
```

Parameter:
- sSectionName: Name einer Section (Text in eckigen Klammern) in der INI-Datei
- sKeyName: Name eines Schlüsselwertes in der Section (Text vor dem Zeichen =)
- sDefaultErgebnis: Text der zurückgegeben wird, wenn der Schlüsselwert nicht gefunden wurde.
- sErgebnisPuffer: Zeichenkette in die der gelesene Text kopiert wird (Text hinter dem Zeichen =)
- nErgebnisPufferGroesse: maximale Anzahl von Zeichen die gelesen werden kann (maximale Länge von sErgebnisPuffer)
- sIniDateiName: Pfad und Dateiname der INI-Datei.

Rückgabewert:
Anzahl der gelesenen Zeichen (entspricht der Stringlänge von sErgebnisPuffer).

```
DWORD GetProfileString (LPCTSTR sSectionName,
                        LPCTSTR sKeyName,
                        LPCTSTR sDefaultErgebnis,
                        LPTSTR  sErgebnisPuffer,
                        DWORD   nErgebnisPufferGroesse);
```

Parameter:
- sSectionName: Name einer Section (Text in eckigen Klammern) in der WININI-Datei
- sKeyName: Name eines Schlüsselwertes in der Section (Text vor dem Zeichen =)
- sDefaultErgebnis: Text der zurückgegeben wird, wenn der Schlüsselwert nicht gefunden wurde.
- sErgebnisPuffer: Zeichenkette in die der gelesene Text kopiert wird (Text hinter dem Zeichen =)
- nErgebnisPufferGroesse: maximale Anzahl von Zeichen die gelesen werden kann (maximale Länge von sErgebnisPuffer)

Rückgabewert:
Anzahl der gelesenen Zeichen (entspricht der Stringlänge von sErgebnisPuffer).

```
HGDIOBJ GetStockObject(int nObject);
```

Parameter:
- nObject: ID des vorgefertigten Objektes, dessen Handle zu ermitteln ist (u.a. BLACK_BRUSH, DKGRAY_BRUSH, GRAY_BRUSH, LTGRAY_BRUSH, SYSTEM_FONT, SYSTEM_FIXED_FONT, WHITE_BRUSH, BLACK_PEN, WHITE_PEN).

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

HMENU GetSubMenu (HMENU hMenu, int nPos);

Parameter:
- hMenu: Handle des Menüs, dessen Untermenü ermittelt werden soll.
- nPos: Position des Menüs, dessen Handle ermittelt werden soll.

Rückgabewert:
Ungleich Null (Handle zum gesuchten Untermenü), wenn die Funktion erfolgreich war, sonst Null.

BOOL GetTextMetrics (HDC hDC, LPTEXTMETRIC lpTM);

Parameter:
- hDC: Device-Context des Gerätes für das die Größen ermittelt werden sollen.
- lpTM: Zeiger auf eine TEXTMETRIC Struktur, in welche die ermittelten Werte eingetragen werden.

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

LONG GetWindowLong (HWND hWnd, int nIndex);

Parameter:
- hWnd: Handle des Fensters, auf dessen Create-Daten zugegriffen werden soll.
- nIndex: Index der Daten, auf die zugegriffen werden soll. Dies können eigene Daten in den Zusatzbytes sein, die bei CreateWindow angegeben werden (positive Indices) oder die Inhalte z.B. folgender Konstanten:
  GWL_EXSTYLE, GWL_STYLE, GWL_HWNDPARENT, GWL_ID, GWL_WNDPROC, GWL_HINSTANCE, GWL_USERDATA, DWL_DLGPROC

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

int GetWindowText (HWND hWnd, LPCTSTR sTitel, int nMaxCount);,

Parameter:
- hWnd: Handle des Controls, dessen Text (= Fenstertitel) ermittelt werden soll.
- sTitel: Zeichenkette in die der gelesene Text (Titel) kopiert wird
- nMaxCount: maximale Anzahl von Zeichen die gelesen werden kann (maximale Länge von sTitel)

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war = Anzahl der gelesenen Zeichen (entspricht der Stringlänge von sTitel). Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

HGLOBAL GlobalAlloc (UINT uFlags, DWORD dwBytes);,

Parameter:
- uFlags: Styleflags, die besagen, wie mit dem reservierten Speicher umgegangen werden darf (u.a. GMEM_FIXED, GMEM_MOVEABLE, GMEM_DISCARDABLE, GMEM_ZEROINIT, GMEM_SHARE).
- dwBytes: Anzahl der Bytes im global reservierten Speicherbereich.

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war (Handle auf den reservierten Speicher). Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

HGLOBAL GlobalFree (HGLOBAL hMem);,

Parameter:
- hMem: Handle des freizugebenden globalen Speichers.

Rückgabewert:
Null, wenn die Funktion erfolgreich war (Handle auf den reservierten Speicher). Ist der Rückgabewert ungleich Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

LPVOID GlobalLock (HGLOBAL hMem);,

Parameter:
- hMem: Handle des globalen Speichers, der gesperrt werden soll (= keine Verschiebung im virtuellen oder physikalischen Speicher erlaubt) und dessen Zeiger zu ermitteln ist.

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war (Adresse des reservierten Speichers). Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

7.8 Windowsfunktionen – I

BOOL InvalidateRect (HWND hWnd, CONST RECT *lpRect,
BOOL bErase);

Parameter:
- hWnd: Das Handle des Fensters, welches zu invalidieren ist.
- lpRect: Das zu invalidieren Rechteck. Hier kann ein von Programmierer berechnete Rechteck in Form einer **RECT** Struktur angegeben werden. Wird statt dessen **NULL** angegeben so wertet Windows dies als Anweisung das komplette mit **hWnd** verbundenen Fenster zu invalidieren.
- bErase: Ein **BOOL** Flag, welches besagt, ob das angegebene Rechteck vor Neuzeichnung zu löschen ist. Wird hier **TRUE** angegeben, so wird das Rechteck bei Aufruf der Funktion **BeginPaint** automatisch gelöscht.

Returnwert :
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

7.9 Windowsfunktionen – L

BOOL LineTo (HDC hDC, int nXEnd, int nYEnd);

Parameter:
- hDC: Device-Context des Gerätes auf dem die Linie gezeichnet werden soll.
- nXEnd: x-Koordinate des Endpunktes, bis zu dem von der aktuellen x-Koordinate gezeichnet werden soll.
- nYEnd: y-Koordinate des Endpunktes, bis zu dem von der aktuellen y-Koordinate gezeichnet werden soll.

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

7.10 Windowsfunktionen – M

BOOL MessageBeep (UINT uType);

Parameter:
- uType: Soundtyp, eine der folgenden Konstanten, die den Sounds entsprechend, die in der Systemsteuerung zugeordnet sind. 0xFFFFFFFF (Standard), MB_OK, MB_ICONASTERISK, MB_ICONHAND, MB_ICONEXCLAMATION, MB_ICONQUESTION

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

int MessageBox (HWND hWnd, LPCTSTR sText, LPCTSTR sTitel,
UINT uType);

Parameter:
- hWnd: Handle des übergeordneten Fensters, zu dem die MessageBox gehören soll. Wird NULL übergeben, so ist automatisch das Desktop-Fenster gemeint.
- sText: Anzuzeigender Messagetext.
- sTitel: Titel des Messagefensters.
- uType: Eigenschaften des Messagefensters und enthaltene Buttons (u.a. MB_ABORTRETRYIGNORE, MB_CANCELTRYCONTINUE, MB_OK, MB_OKCANCEL, MB_RETRYCANCEL, MB_YESNOCANCEL, MB_TOPMOST, MB_TASKMODAL, MB_ICONQUESTION, MB_ICONWARNING, MB_ICONEXCLAMATION, MB_ICONINFORMATION, MB_ICONASTERISK, MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND, MB_APPLMODAL, MB_SYSTEMMODAL, MB_YESNO)

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war (= ausgewählter Button: IDABORT, IDCANCEL, IDCONTINUE, IDIGNORE, IDNO, IDOK, IDRETRY, IDTRYAGAIN, IDYES). Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

BOOL MoveToEx (HDC hDC, int x, int y, LPPOINT lpPoint);

Parameter:
- hDC: Device-Context des Gerätes für den die aktuelle Koordinate für Zeichenoperationen gesetzt werden soll.

- x: x-Koordinate des Punktes, zu dem der aktuelle Ansatzpunkt verschoben werden soll.
- y: y-Koordinate des Punktes, zu dem der aktuelle Ansatzpunkt verschoben werden soll.
- lpPoint: Struktur (POINT), in welche die Koordinaten die vor der Verschiebung aktuell waren gesichert werden. Dieser Parameter darf NULL sein.

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

```
BOOL MoveWindow (HWND hWnd, int x, int y, int nWidth,
                 int nHeight, BOOL bRepaint);
```

Parameter:
- hWnd: Handle des Fensters, das bewegt und/oder in der Größe verändert werden soll.
- x: neue horizontale Position des Fensters.
- y: neue vertikale Position des Fensters.
- nWidth: neue Breite des Fensters.
- nHeight: neue Höhe des Fensters.
- bRepaint: Wenn **TRUE** wird automatisch eine WM_PAINT für das mit dem Handle (hWnd) verbundene Fenster ausgelöst.

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

7.11 Windowsfunktionen – O

```
BOOL OpenClipboard (HWND hWndOwner);
```

Parameter :
- hWndOwner: Handle des Fensters, welches nun Zugriff auf das Clipboard hat.

Returnwert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

7.12 Windowsfunktionen – P

```
VOID PostQuitMessage (int nExitCode);
```

Parameter:
- nExitCode: Rückgabewert an Windows (bzw. Batchprogramme).

7.13 Windowsfunktionen – R

```
int ReleaseDC (HWND hWnd, HDC hDC);
```

Parameter:
- hWnd: Handle des Fensters, für welches der Device-Context freigegeben werden soll.
- hDC: Der freizugebende Device-Context.

Rückgabewert:
Eins wenn der DC freigegeben wurde, sonst Null.

BOOL RestoreDC (HDC hDC, int nSavedDC);

Parameter:
- hDC: Device-Context des Gerätes der wieder auf einen gesicherten Zustand zu bringen ist.
- nSavedDC: Index des gesicherten Zustandes (Rückgabewert von SaveDC). –1 beschreibt immer den zuletzt gesicherten Zustand.

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

7.14 Windowsfunktionen – S

int SaveDC (HDC hDC);

Parameter:
- hDC: Device-Context des Gerätes, dessen aktueller Zustand zu sichern ist.

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war (Index des intern gesicherten Zustandes). Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

HGDIOBJ SelectObject (HDC hDC, HGDIOBJ hGDIObj);

Parameter:
- hDC: Device-Context des Gerätes, für das eine limitierte Resource zu wählen ist.
- hGDIObj: Die neu auszuwählende und fortan bei Zeichenoperationen zu verwendende limitierte Resource. Dies kann eine Resource des folgenden Typs sein, von denen immer nur eine für die Zeichenoperation gültig sein kann: Bitmap, Brush, Font, Pen, Region.

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war (Handle der ausgetauschten Resource). Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

LRESULT SendMessage (HWND hWnd, UINT nMsg, WPARAM wParam, LPARAM lParam);

Parameter:
- hWnd: Handle des Fensters, an das die Message gesendet werden soll.
- nMsg: ID der zu sendenden Message
- wParam: erster Parameter. Typ und Parameter sind abhängig von der Message-ID.
- lParam: zweiter Parameter. Typ und Parameter sind abhängig von der Message-ID.

Rückgabewert:
Ist abhängig von der Message.

int SetBkMode (HDC hDC, int nBkMode);

Parameter:
- hDC: Device-Context, dessen Hintergrundmodus geändert werden soll.
- nBkMode: neu einzustellender Backgroundmodus (OPAQUE oder TRANSPARENT).

Rückgabewert:
Ungleich Null (vorherige Einstellung von OPAQUE oder TRANSPARENT), wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

int SetMapMode (HDC hDC, int nMapMode);

Parameter:
- hDC: Device-Context, dessen mapMode geändert werden soll.
- nMapMode: neuer MapModus.

Rückgabewert:
Ungleich Null (vorheriger Map-Mode), wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

UINT SetTextAlign (HDC hDC, UINT nMode);

Parameter:
- hDC: Device-Context, dessen Text-Ausgabeflags geändert werden soll.
- nMode: neu einzustellende Text-Ausgabeflags (u.a TA_TOP, TA_BASELINE, TA_BOTTOM, TA_CENTER, TA_LEFT, TA_RIGHT).

Rückgabewert:
Ungleich Null (vorherige Einstellung), wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

COLORREF SetTextColor (HDC hDC, COLORREF crColor);

Parameter:
- hDC: Device-Context, dessen Textfarbe geändert werden soll.
- crColor: Referenz (RGB-Wert) der neu zu verwendenden Textfarbe.

Rückgabewert:
Ungleich Null (vorherige Farb-Einstellung), wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

UINT SetTimer (HWND hWnd, UINT nIDEvent, UINT uElapse,
TIMERPROC lpTimerProc);

Parameter :
- hWnd: Handle des Fensters, an dessen Behandlungsfunktion (WndProc) die Timer-Messages gesendet werden sollen.
- nIDEvent: Nummer die der Timer als ID haben soll, denn es ist auch möglich einem Fenster mehrere, unterschiedliche Timer mit unterschiedlichen Zeitabständen zuzuweisen.
- uElapse: Zeitabstand in welchem die Timer-Nachrichten geschickt werden sollen (in Millisekunden)

- lpTimerProc: Callback-Adresse. Normalerweise wird als Adresse NULL angegeben, die Message kommt wie gehabt über die Message-Queue (also nicht, wenn das Programm den Focus nicht hat), für andere Maßnahmen kann hier die Adresse einer CALLBACK Funktion angegeben werden, die dann direkt aufgerufen wird.

Returnwert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

BOOL SetViewPortExtEx (HDC hDC, int nXExt, int nYExt, LPSIZE lpSize);

Parameter:
- hDC: Device-Context, dessen Viewport-Ausdehnung (sichtbarer Ausschnitt) geändert werden soll.
- nXExt: neuer horizontaler Ausschnitt.
- nYExt: neuer vertikaler Ausschnitt.
- lpSize: Zeiger auf Size-Struktur, welche anschließend die vorher eingestellte Größe beinhaltet. Darf NULL sein.

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

BOOL SetViewPortOrgEx (HDC hDC, int x, int y, LPSIZE lpSize);

Parameter:
- hDC: Device-Context, dessen Viewport-Startpunkt (linke odere Ecke des sichtbaren Ausschnitts) geändert werden soll.
- nXExt: neuer horizontaler Startpunkt.
- nYExt: neuer vertikaler Startpunkt.
- lpSize: Zeiger auf Size-Struktur, welche anschließend den zuvor eingestellten Startpunkt beeinhaltet. Darf NULL sein.

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

BOOL SetWindowExtEx (HDC hDC, int nXExt, int nYExt, LPSIZE lpSize);

Parameter:
- hDC: Device-Context, dessen (virtuelle) Fenstergröße geändert werden soll.
- nXExt: neue Fensterbreite.
- nYExt: neue Fensterhöhe.
- lpSize: Zeiger auf Size-Struktur, welche anschließend die vorher eingestellte Größe beinhaltet. Darf NULL sein.

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

BOOL SetWindowText (HWND hWnd, LPCTSTR sTitel);,

Parameter:
- hWnd: Handle des Controls, dessen Text (= Fenstertitel) tgeändert werden soll.
- sTitel: Zeichenkette in die den neuen Text (Titel) enthält.

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war = Anzahl der gelesenen Zeichen (entspricht der Stringlänge von sTitel). Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

int StartDoc (HDC hDC, CONST DOCINFO *lpDI);,

Parameter:
- hDC: Device-Context, dessen (virtuelles) Fenster gedruckt werden soll.
- lpDI: DocInfo-Struktur für die Printer-Queue.

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war (Druckjob-ID). Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

int StartPage (HDC hDC);,

Parameter:
- hDC: Device-Context, dass nun Daten zur Druckausgabe übergeben bekommt.

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

7.15 Windowsfunktionen – T

BOOL TextOut (HDC hDC, int nXStart, int nYStart,
LPCTSTR lpString, int cbString);

Parameter:
- hDC: Device-Context des Gerätes auf dem der Text auszugeben ist.
- nXStart: x-Koordinate des Ausgabebeginns
- nYStart: y-Koordinate des Ausgabebeginns
- lpString: Adresse der auszugebenden Zeichenkette
- cbString: Länge der auszugebende Zeichenkette

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

7.16 Windowsfunktionen – W

int WINAPI WinMain (HINSTANCE hInst, HINSTANCE hPrevInst
LPSTR sCmdLine, int nCmdShow);

Parameter:
- hInst: Instance-Handle der aktuellen Programmkopie.

- hInst: Instance-Handle der vorherigen Programmkopie, für 32-Bit Programme immer NULL.
- sCmdLine: Übergebene Parameter als Text.
- nCmdShow: ID, in welchem Zustand das Fenster zu öffnen ist (u.a. SW_SHOWMAXIMIZED, SW_SHOWMINIMIZED, SW_SHOWNORMAL)

Rückgabewert:
Programmabhängig.

```
BOOL WritePrivateProfileString (LPCTSTR sSectionName,
                                LPCTSTR sKeyName,
                                LPCTSTR sWert,
                                LPCTSTR sIniDateiName);
```

Parameter:
- sSectionName: Name einer Section (Text in eckigen Klammern) in der INI-Datei
- sKeyName: Name eines Schlüsselwertes in der Section (Text vor dem Zeichen =)
- sWert: Text der hinter den Schlüsselwert geschrieben wird.
- sIniDateiName: Pfad und Dateiname der INI-Datei.

Rückgabewert:
Ungleich Null, wenn die Funktion erfolgreich war. Ist der Rückgabewert Null, so kann mittels **GetLastError** eine genaue Fehlermeldung ermitteln werden.

7.17 Windowsfunktionen – Z

```
VOID ZeroMemory (PVOID pDestination, DWORD nLength);
```

Parameter:
- pDestination: Startadressen, ab welcher der Speicher mit Nullen gefüllt werden soll.
- nLength: Anzahl der Zeichen, die mit Null überschrieben werden sollen.

8 Weitere Beispielprogramme

Nachfolgend einige Beispielprogramme, die unterschiedliche Techniken zeigen.

8.1 Einfache Ausgabe

```
//*************************************************************************
//   P1.cpp - Öffnet eine eigene Client-Area und gibt dort einen Text aus
//*************************************************************************
// Include-Datei <windows.h> ist unverzichtbar, sie enthält alle wichtigen
// defines, die für ein Windows-Programm benötigt werden
//-------------------------------------------------------------------------
#include <windows.h>

//-------------------------------------------------------------------------
// Prototyp der Handlerfunction, die die Messages der Client-Area verarbeitet
// Das Schlüsselwort PASCAL ist notwendig, weil das Windowskernel ursprüng-
// lich in Pascal programmiert wurde. Pascal und C legen die Parameter in
// unterschiedlicher Reihenfolge auf dem Stack ab, so daß der C-Funktion mit-
// geteilt werden muß, daß die Parameter in anderer Reihenfolge vom Stack ab-
// zuholen sind
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------
// VERALTETE DEKLARATIONSFORM:
// long FAR PASCAL _export WndProc (HWND, UINT, UINT, LONG);
//-------------------------------------------------------------------------
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);
BOOL MessagePreprocess (HWND hwin, MSG &aMessage);

//-------------------------------------------------------------------------
// Alle Windowsprogramme beginnen mit "WinMain" statt mit "main"
//-------------------------------------------------------------------------
// Das pragma "argsused" ist Borland-spezifisch und unterdrückt das Warning,
// daß wir den Standardparameter "lpszCmdParam" nirgendwo benutzt haben...
//-------------------------------------------------------------------------
// in Parameter "hInstance"
// übergibt das Windowskernel die Kennung des Programms. Dies ist nötig, da
// unter Windows ja mehrere Programme Parallel laufen können. Jedes Programm
// erhält deshalb eine eindeutige ID-Nummer, über welche die "Messages" zuge-
// stellt werden können.
//-------------------------------------------------------------------------
// 16-Bit Windows: Im Parameter "hPrevInstance" teilt Windows mit, ob das
// gleiche Programm bereits im Speicher vorliegt, also erneut gestartet
// worden ist. In diesem Fall dürfen einige Teile des Programmcodes nicht
// erneut durchlaufen werden. "hPrevInstance" ist NULL, wenn das Programm zum
// ersten Mal in den Speicher geladen wird, ansonsten erhält man hier die
// Instanzen-Nummer des vorherigen Aufrufes.
// 32-Bit Windows: Parameter ist immer NULL
//-------------------------------------------------------------------------
// in Parameter "lpszCmdParam"
// stehen die Kommandozeilen-Parameter, die ggf. beim Aufruf übergeben wurden
//-------------------------------------------------------------------------
// in Parameter "nCmdShow"
// ist festgelegt, ob das Programm als Pictogramm auf der Oberfläche
// erscheint (inconified) oder als Fenster.
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------
// VERALTETE DEKLARATIONSFORM:
// int PASCAL WinMain (HANDLEhInstance, HANDLE hPrevInstance, LPSTR lpszCmdLine,
// int nCmdShow)
//-------------------------------------------------------------------------
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpszCmdParam, int nCmdShow)
{
   //-----------------------------------------------------------------------//
   // Windows benötigt einen eindeutigen Namen (die Fensterklasse), um fest-  //
   // stellen zu können, ob das gleiche Programm schon läuft. Ein Kopieren    //
   // und Umbenennen der Exe-Datei genügt also nicht, um Windows zu über-     //
   // listen... Der Sinn dahinter ist, daß bei einem erneuten Aufruf des Pro- //
   // gramms nur der Datenbereich und nicht der Programmbereich neu angelegt  //
   // wird.                                                                   //
   // PRÄFIX "sz" steht für String (String Zeroterminated)                    //
   //-----------------------------------------------------------------------//
   static char szWindowclassname [] = "P1";
```

```
//------------------------------------------------------------------------//
// Datentyp HWND bedeutet Window-Handle, also eindeutige Kennnummer des   //
// Fensters                                                               //
// PRÄFIX "h" steht für Handle, also eine eindeutige Kennziffer           //
//------------------------------------------------------------------------//
HWND        hWindow;

//------------------------------------------------------------------------//
// Datentyp MSG bedeutet Message, eine Struct, die die Daten einer an das //
// Programm weitergeleiteten Nachricht enthält                            //
// PRÄFIX "a" steht für eine Struct                                       //
//------------------------------------------------------------------------//
MSG         aMessage;

//------------------------------------------------------------------------//
// Datentyp WNDCLASS meldet die Client-Area des Programms beim Windows-   //
// Message-Dispatcher als Nachrichtenempfänger an                         //
// PRÄFIX "a" steht für eine Struct                                       //
//------------------------------------------------------------------------//
WNDCLASS    aWindowclass;

//------------------------------------------------------------------------//
// Die Abfrage auf hPrevInstance stellt sicher, daß der Ablaufcode eines  //
// Programms immer nur einmal geladen ist. Bei Multi-Instanz-Programmen   //
// wird immer nur der Datenteil neu angelegt (16Bit)                      //
// Unter 32-Bit-Windows ist dieser Parameter immer NULL! - sollte aber bei//
// Multi-Plattformprogrammen (Programme für Win3.x und Win95) sicherheits-//
// halber abgefragt werden                                                //
//------------------------------------------------------------------------//
if (!hPrevInstance)
{
   //---------------------------------------------------------------------//
   // Wenn das Programm als erste Instanz in den Speicher geladen wird, muß//
   // daß Hauptfenster (die Client-Area) beim Windows-Kernel angemeldet   //
   // werden. Dazu ist eine WNDCLASS-Sturct auszufüllen, die das Fenster  //
   // als Fensterklasse anmeldet und ein bestimmtes Aussehen vereinbart   //
   //---------------------------------------------------------------------//
   // Der Class-Style (CS_xxxx) legt bestimmte Verhaltensweisen des Haupt-//
   // fensters (d.h. der Client-Area) fest. Am häufigsten werden verwendet://
   // CS_HREDRAW = Gibt an, daß das gesamte Fenster neu zu zeichnen ist,  //
   //              wenn Das Fenster in horizontaler Richtung bewegt oder  //
   //              in seiner Breite verändert wurde                       //
   // CS_VREDRAW = Gibt an, daß das gesamte Fenster neu zu zeichnen ist,  //
   //              wenn Das Fenster in vertikaler Richtung bewegt oder    //
   //              in seiner Höhe verändert wurde                         //
   // CS_NOCLOSE = Schaltet das "Schließsymbol" in der obersten Zeile ab  //
   // CS_DBLCLKS = Weist Windows an, eine Doppelclick-Message an die      //
   //              Message-Handler-Function des Programms zu schicken, wenn//
   //              der Benutzer einen Doppelclick in irgendeinem der zur  //
   //              Anwendung gehörenden Fenster eingibt.                  //
   // Außerdem gibt es noch die seltener verwendeten Styles:              //
   // CS_BYTEALIGNCLIENT, CS_BYTEALIGNWINDOW, CS_CLASSDC,CS_GLOBALCLASS,  //
   // CS_OWNDC, CS_PARENTDC und CS_SAVEBITS                               //
   //---------------------------------------------------------------------//
   aWindowclass.style         = CS_HREDRAW | CS_VREDRAW;

   //---------------------------------------------------------------------//
   // Name der Function, die in diesem Programm für die Abarbeitung der   //
   // Messages sorgt. Die Funktion "DispatchMessage" ruft diesen Message- //
   // Handler automatisch auf, weil er Teil der Klassenbeschreibung ist   //
   // entspricht einem Pointer auf diese Function                         //
   //---------------------------------------------------------------------//
   aWindowclass.lpfnWndProc   = WndProc;

   //---------------------------------------------------------------------//
   // Speicherplatz, der für eigene Daten im Rahmen der Fensterklasse und //
   // des Fensters vom Benutzer als Erweiterung der Struktur vorgesehene  //
   // sind. Wird hier nicht und auch sonst selten benutzt                 //
   //---------------------------------------------------------------------//
   aWindowclass.cbClsExtra    = 0;
   aWindowclass.cbWndExtra    = 0;

   //---------------------------------------------------------------------//
   // Die Erst-Instance wird ebenfalls in der Struktur gesichert, bei     //
   // weiteren Instancen sorgt Windows für die Fortführung der Zählerei in//
   // einer Liste (schließlich muß ja auch die hPrevInstance irgendwo her-//
   // kommen, d.h. Windows muß sie verwalten                              //
   //---------------------------------------------------------------------//
   aWindowclass.hInstance     = hInstance;

   //---------------------------------------------------------------------//
   // Angabe des Icons, welches angezeigt wird, wenn das Programm als Icon//
   // auf den Desktop heruntergeclickt wird. Hier das Standardicon        //
```

```
   //---------------------------------------------------------------------//
   aWindowclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);

   //---------------------------------------------------------------------//
   // Angabe des Mouse-Cursors, der angezeigt wird, wenn ddie Mouse sich  //
   // im Progframmfenster befindet. Hier der Standardcursor               //
   //---------------------------------------------------------------------//
   aWindowclass.hCursor       = LoadCursor (NULL, IDC_ARROW);

   //---------------------------------------------------------------------//
   // Angabe des Hintergrund-Pinsels (Farbe/Muster), mit dem die Fläche des//
   // Fensters bei Programmstart gefüllt wird. Hier ein Standardbrush     //
   //---------------------------------------------------------------------//
   aWindowclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);

   //---------------------------------------------------------------------//
   // Angabe der Menüstruktur, die benutzt werden soll um das Menü des    //
   // Fensters zu bilden. Hier Null, weil kein Menü verwendet wird.       //
   //---------------------------------------------------------------------//
   aWindowclass.lpszMenuName  = NULL;

   //---------------------------------------------------------------------//
   // Eintragen des Klassennamens (s.o.) zur eindeutigen Identifikation   //
   // Meistens wird der Name des Programms verwendet                      //
   //---------------------------------------------------------------------//
   aWindowclass.lpszClassName = szWindowclassname;

   //---------------------------------------------------------------------//
   // Nachdem das "Formular" zur Beantragung einer Klasseneintragung im   //
   // Windowskernel korrekt ausgefüllt ist, muß es beim Windowskernel auch //
   // eingereicht werden                                                  //
   //---------------------------------------------------------------------//
   if (!RegisterClass (&aWindowclass))
   {
      MessageBox (NULL, "Registrierung nicht möglich", "P1", MB_OK);
      return (0);
   }
}

//-------------------------------------------------------------------------//
// Nachdem wir die Klasse registriert haben, sind wir beim Kernel (dem     //
// Einwohnermeldeamt von Windows) bekannt. Um nun ein Fenster eröffnen zu  //
// dürfen rufen wir die Function CreateWindow auf. Erst  dadurch wird      //
// intern der Speicherplatz geschaffen, um ein Fenster zu verwalten. Dieses//
// erste Fenster ist zugleich das wichtigste, denn es ist Bezugspunkt für  //
// alle weiteren Unterfenster. Das Hauptfenster wird daher auch Kunden-    //
// bereich (Client-Area) genannt. Zudem können wir in groben Zügen fest-   //
// legen, wie das Fenster aussehen soll.                                   //
// Als Ergebnis der Funktion erhalten wir ein Handle auf das erzeugte      //
// Fenster, also eine eindeutige Identifikationsnummer, mit der das Fen-   //
// ster bestimmt werden kann.                                              //
//-------------------------------------------------------------------------//
// Parameter 1: Name der Fensterklasse, zu dem die Client-Area gehört      //
//-------------------------------------------------------------------------//
// Parameter 2: Text, der in der Titelzeile des Fensters steht.            //
//-------------------------------------------------------------------------//
// Parameter 3: Fensterstil. Mögliche Werte sind (mit dem binär-Oder-      //
//              Operator verbunden:                                        //
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - //
// Titelzeilen-Elemente                                                    //
// WS_CAPTION          Fenster mit Titelzeile (beinhaltet WS_BORDER Stil)  //
// WS_MAXIMIZEBOX      Fenster hat Vergrößerungsbutton in Titelzeile       //
// WS_MINIMIZEBOX      Fenster hat Verkleinerungsbutton in der Titelzeile  //
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - //
// Randelemente                                                            //
// WS_BORDER           Fenster mit schmalem Rand                           //
// WS_THICKFRAME       Fenster mit breitem Rand, über den Fenstergröße     //
//                     eingestellt werden kann. Identisch mit WS_SIZEBOX   //
// WS_SIZEBOX          siehe WS_THICKFRAME                                 //
// WS_HSCROLL          Erzeugt ein Fenster mit einer senkrechten Scrollbar //
// WS_VSCROLL          Erzeugt ein Fenster mit einer horizontal-Scrollbar  //
// WS_SYSMENU          Erzeugt ein Fenster mit dem System-Menü in der      //
//                     Titelzeile. Dazu muß Stil WS_CAPTION gesetzt sein   //
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - //
// Zusammengefaßte Elemente                                                //
// WS_OVERLAPPED       Erzeugt sogenanntes OVERLAPPED-Fenster. Hat immer   //
//                     Titelzeile (WS_CAPTION) und Rand (WS_BORDER). Iden- //
//                     tisch mit WS_TILED                                  //
// WS_TILED            siehe WS_OVERLAPPED                                 //
// WS_DLGFRAME         Fenster mit dem typischen Aussehen einer Dialogbox  //
//                     Kann keine Titelzeile (WS_CAPTION) haben!           //
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - //
// Darstellungsgrößen                                                      //
```

```
// WS_ICONIC           Erzeugt Fenster, das zu Beginn als Icon vorliegt.   //
// WS_MAXIMIZE         Erzeugt ein Fenster maximaler Größe.                //
// WS_MINIMIZE         identisch mit WS_ICONIC                             //
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - //
// sonstige Eigenschaften                                                  //
// WS_DISABLED         Fenster ist abgeschaltet                            //
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - //
// Zusammenfassende Stil-Pakete                                            //
// WS_OVERLAPPEDWINDOW Erzeugt ein Fenster mit den Stilen WS_OVERLAPPED,   //
//                     WS_CAPTION, WS_SYSMENU, WS_THICKFRAME,              //
//                     WS_MINIMIZEBOX, und WS_MAXIMIZEBOX.                 //
// WS_TILEDWINDOW      identisch mit WS_OVERLAPPEDWINDOW                   //
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - //
// Stile für abhängige Fenster (Popupfenster / Childfenster)               //
// WS_CHILD            Definiert ein Unterfenster (nicht bei Client-Area   //
//                     möglich!) Schließt sich mit WS_POPUP gegenseitig aus//
// WS_CHILDWINDOW        identisch mit WS_CHILD                              //
// WS_CLIPCHILDREN     Die Bereiche von Child-Fenstern werden ausgelassen, //
//                     wenn das übergeordnete Fenster neu gezeichnet wird. //
// WS_CLIPSIBLINGS     Die Bereiche von überlappenden Child-Fenstern       //
//                     werden beachtet, so daß nicht in die überlappenden  //
//                     Bereiche geschrieben/gezeichnet wird                //
// WS_POPUP            Erzeugt ein Dialogfenster. (nicht für Client-Area   //
//                     möglich!) Schließt sich mit WS_CHILD gegenseitig aus//
// WS_POPUPWINDOW      Erzeugt ein Dialogfenster mit Standardaussehen.     //
//                     Beinhaltet WS_BORDER, WS_POPUP, und WS_SYSMENU.     //
//                     Muß mit WS_CAPTION kombiniert werden, um das        //
//                     System-Menü sichtbar zu machen                      //
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - //
// Stile für Dialogelemente                                                //
// WS_GROUP            Bezeichnet das erste Fenster einer Child-Gruppe.    //
//                     Wird für die Fenster von Buttons und andere Dialog- //
//                     elemente verwendet. Zwischen Fenstern der Gruppe    //
//                     kann der Focus mit den Cursortasten bewegt werden.  //
//                     Das nächste Fenster mit dem WS_GROUP beendet die    //
//                     Gruppe wieder und startet eine neue                 //
// WS_TABSTOP          Besagt, daß Dialogelement mittels Tabulatortaste    //
//                     erreicht werden kann.                               //
// WS_VISIBLE          Erzeugt sichtbares Dialogelement. Unsichtbare       //
//                     Elemente sind zugleich auf WS_DISABLED geschaltet   //
//-------------------------------------------------------------------------//
// Parameter 4: Fensterposition X-Koordinate. Angabe in Pixel. Die Angabe  //
//              von CW_USEDEFAULT überläßt die Entscheidung dem  Kernel    //
//-------------------------------------------------------------------------//
// Parameter 5: Fensterposition Y-Koordinate. Angabe in Pixel. Die Angabe  //
//              von CW_USEDEFAULT überläßt die Entscheidung dem  Kernel    //
//-------------------------------------------------------------------------//
// Parameter 6: Fensterbreite in Pixeln. Die Angabe von CW_USEDEFAULT      //
//              überläßt die Entscheidung dem Kernel                       //
//-------------------------------------------------------------------------//
// Parameter 7: Fensterhöhe in Pixeln. Die Angabe von CW_USEDEFAULT über-  //
//              läßt die Entscheidung dem Kernel                           //
//-------------------------------------------------------------------------//
// Parameter 8: Handle des übergeordneten Fensters. Da es sich hier um die //
//              Client-Area handelt, gibt es kein übergeordnetes Fenster   //
//-------------------------------------------------------------------------//
// Parameter 9: Handle des Menüs, welches im Fenster dargestellt werden    //
//              soll. Da hier kein Menü vereinbart wurde ist der Wert      //
//              logischerweise NULL. Bei CHILD-Fenstern wird dieser Ein-   //
//              trag als eindeutige ID-Nummer benutzt                      //
//-------------------------------------------------------------------------//
// Parameter 10: Instanzenhandle. Hier wird eingetragen, zu welcher Auf-   //
//               rufinstanz das Fenster gehört. Das ist wichtig, damit in  //
//               den Dialogen die richtigen Datenbereiche angezeigt werden //
//-------------------------------------------------------------------------//
// Parameter 11. Pointer für die in der Klasse reservierten Speicher-      //
//               bereiche (s.o.) wird nur selten genutzt.                  //
//-------------------------------------------------------------------------//
hWindow = CreateWindow (szWindowclassname, "Hurra, es geht !!!",
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL);
if (!hWindow)
{
   MessageBox (NULL, "Fenster konnte nicht erzeugt werden", "P1", MB_OK);
   return (0);
}

//-------------------------------------------------------------------------//
// Das Erzeugen des Fensters beinhaltet nicht, daß das Fenster auch ange-  //
// zeigt wird. Gelegentlich will man ja etwas im Hintergrund tun und das   //
// zugehörige Fenster nur bei Bedarf anzeigen. Deshalb muß das Aufblenden  //
```

```
   // des Fensters ausdrücklich aufgerufen werden.                          //
   // Als Parameter übergibt man das Handle des erzeugten Fensters (Identifi- //
   // kationsnummer) und den Darstellungsmodus, der als Eigenschaft (z.B.     //
   // "als Icon") von Windows als Parameter an WinMain übergeben wurde.       //
   // Dieser Befehl darf nur EINMAL, hier im WinMain verwendet werden)        //
   //-------------------------------------------------------------------------//
   ShowWindow (hWindow, nCmdShow);

   //-------------------------------------------------------------------------//
   // Das Aufblenden des Fensters zeigt im Grunde nur die Fläche an. Aus Ge-  //
   // schwindigkeitsgründen werden die Fensterinhalte nur bei Bedarf oder     //
   // Aufforderung gemalt. Die folgende Zeile malt das ganze Fenster neu.     //
   // (Es wird eine WM_PAINT-Message ausgelöst - s.u.)                        //
   //-------------------------------------------------------------------------//
   if (!UpdateWindow (hWindow))
      {
         MessageBox (NULL, "WM_PAINT fehlgeschlagen", "P1", MB_OK);
         return (0);
      }

   //-------------------------------------------------------------------------//
   // Das Fenster ist bereit und gezeichnet, jetzt müssen wir auf alle an-    //
   // kommenden Botschaften reagieren. GetMessage gibt solange TRUE zurück,   //
   // bis eine WM_QUIT-Message anzeigt, daß das Programm beendet wurde. Die   //
   // WM_QUIT-Message ist die letzte Message, die ein Programm erhält.        //
   // Bei ungültigen Messages (an nicht mehr vorhandenes Window-Handle) er-   //
   // gibt GetMessage den Wert -1                                             //
   //-------------------------------------------------------------------------//
   while (GetMessage (&aMessage, NULL, 0, 0))
   {
      //----------------------------------------------------------------------//
      // Filterfunktion für Messages allgemeiner Bedeutung, die nicht an      //
      // jedem einzelnen Control abgegriffen werden sollen                    //
      //----------------------------------------------------------------------//
      MessagePreprocess (hWindow, aMessage);

      //----------------------------------------------------------------------//
      // TranslateMessage übersetzt die Windows-Tastaturcodes (Virtual Keys)  //
      // in Characterdaten. Natürlich nur, wenn solche Daten auch in der      //
      // Message vorliegen. Die MUSS gemacht werden, da man sonst mit den Key-//
      // messages nichts anfangen kann.                                       //
      //----------------------------------------------------------------------//
      TranslateMessage (&aMessage);

      //----------------------------------------------------------------------//
      // DispatchMessage gibt die Message-struct an den Handler weiter, der   //
      // in der Windowsklasse eingetragen wurde (s.o.). Erst dort wird die    //
      // Nachricht verarbeitet. Sind mehrere Fenster vorhanden, so entschei-  //
      // det der Dispatcher anhand des in der Message eingetragenen Handles,  //
      // an welche Handlerroutine die Message geht.                           //
      //----------------------------------------------------------------------//
      DispatchMessage (&aMessage);
   }

   //-------------------------------------------------------------------------//
   // Zu guter letzt, bevor wir uns verabschieden teilen wir dem Windows-     //
   // kernel noch mit, daß wir die WM_QUIT-Message verarbeitet haben, indem   //
   // wir den wparam der Message zurückgeben.                                 //
   //-------------------------------------------------------------------------//
   return (aMessage.wParam);
}

//----------------------------------------------------------------------------//
// Funktion zum Herausfiltern der F1-Taste.                                   //
// Anschließend wird die allgemeine Windowshilfe aufgerufen.                  //
//----------------------------------------------------------------------------//
BOOL MessagePreprocess (HWND hwin, MSG &aMessage)
{
   //-------------------------------------------------------------------------//
   // Nur regieren wenn eine WM_KEYUP (Taste losgelassen) Message vorliegt    //
   //-------------------------------------------------------------------------//
   if (WM_KEYUP == aMessage.message)
   {
      //----------------------------------------------------------------------//
      // nur dann reagieren, wenn die losgelassene Taste die F1 Taste war     //
      //----------------------------------------------------------------------//
      if (VK_F1 == (int)aMessage.wParam)
      {
         //-------------------------------------------------------------------//
         // Die allgemeine Windowshilfe aufrufen                              //
         //-------------------------------------------------------------------//
         WinHelp (hwin, "C:\\WINDOWS\\HELP\\WINHLP32.HLP", HELP_INDEX, 0);
         return (TRUE);
```

```
      }
   }
   return (FALSE);
}

//---------------------------------------------------------------------------//
// Handlerfunktion der Client-Area, wird von DispatchMessage aufgerufen.     //
// Parameter "hwnd" gibt an, an welches Fensterhandle sich die Botschaft     //
//          richtet.                                                         //
// Parameter "message" gibt die Messageart an                                //
// Parameter "wParam" ist ein Datenfeld, das abhängig von der Messageart zu  //
//                    interpretieren ist.                                    //
// Parameter "lParam" ist ein Datenfeld, das abhängig von der Messageart zu  //
//                    interpretieren ist.                                    //
//---------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   //------------------------------------------------------------------------//
   // Handle des Devicecontextes. Der DC kümmert sich um die Belange des anzu-//
   // sprechenden Gerätes (hier Bildschirm), so daß wir uns nicht selbst um  //
   // die Umrechnung von Farben etc. auf die Bildschirmauflösung und Farb-   //
   // tiefe kümmern müssen. Dadurch funktionieren alle Windowsprogramme meist //
   // problemlos mit fas jeglicher, windowsfähiger Hardware. Der DC greift   //
   // auf die entsprechenden Treiber zu, ohne daß der Entwickler sich darum  //
   // im Normalfall Gedanken machen muß.                                     //
   //------------------------------------------------------------------------//
   HDC         hdc ;

   //------------------------------------------------------------------------//
   // Die Paintstruct wird von Windows benötigt, wenn ein Teil des Fensters  //
   // neu zu zeichnen ist.                                                   //
   //------------------------------------------------------------------------//
   PAINTSTRUCT ps ;

   //------------------------------------------------------------------------//
   // rect ist eine Struktur, die einen rechteckigen Bereich des Bildschirms //
   // beschreibt. Sie enthält nur die Koordinaten der linken, oberen und der //
   // rechten unteren Ecke.                                                  //
   //------------------------------------------------------------------------//
   RECT        rect ;

   //------------------------------------------------------------------------//
   // Der switch-Block reagiert auf die Messagearten, die von Windows als    //
   // Nachricht geschickt werden. Hier nur zwei von SEHR VIELEN (ca. 240)    //
   // verschiedenen Messagearten, die es gibt. WM steht für Windowsmessage,  //
   // wobei diese unter Umständen in den Parametern wParam und lParam noch   //
   // weitere Untergruppen oder Informationen beinhalten                     //
   //------------------------------------------------------------------------//
   switch (message)
   {
      //---------------------------------------------------------------------//
      // Die WM_PAINT-Message gibt an, was alles zu tun ist, wenn die Client- //
      // Area neu zu zeichnen ist.                                           //
      // Windows schickt eine WM_PAINT-Message, wenn z.B. die Fenstergröße   //
      // verändert wurde oder man das Fenster verschoben hat.                //
      //---------------------------------------------------------------------//
      case WM_PAINT :
           //----------------------------------------------------------------//
           // BeginPaint leitet das Zeichnen ein. während dieses Vorgangs    //
           // kann in Windows kaum etwas anderes gemacht werden, der Bild-   //
           // schirm wird kurzfristig exclusiv für dieses Programm gebraucht //
           // NIEMALS hier einen Breakpoint setzen, denn das führt in 99%    //
           // aller Fälle zu einem Absturz!                                  //
           //----------------------------------------------------------------//
           hdc = BeginPaint (hwnd, &ps);

           //----------------------------------------------------------------//
           // Hier werden die gerade gültigen Koordinaten der Fenstergröße   //
           // der Client-Area ermittelt. Da der Anwender die Fenstergröße    //
           // nahezu beliebig ändern kann, ist dies notwendig um die gerade  //
           // eingestellte Größe zu ermitteln                                //
           // Der Bereich der ZUrückgegeben wird, ist der innere Fenster-    //
           // bereich (Ohne Titelzeile und Rand!). Das Ergebnis wird in die  //
           // Rechteck-Struktur (rect)                                       //
           //----------------------------------------------------------------//
           GetClientRect (hwnd, &rect);

           //----------------------------------------------------------------//
           // Drawtext schreibt einen Text in die Client-Area.               //
           // 1. Parameter ist der HDC, den Windows in BeginPaint ermittelt  //
           //              hat.                                              //
           // 2. Parameter ist der auszugebende Text, oder der Pointer auf   //
           //              den auszugebenden Text.                           //
```

```
     // 3. Parameter ist die auszugebende Textlänge. Wird hier -1 ange- //
     //              geben, so zählt Windows selber. Vorsicht, ist die  //
     //              Angabe Größer als die Textlänge, so steht Unsinn   //
     //              auf dem Bildschirm                                 //
     // 4. Parameter ist der rechteckige Bildschirmbereich, in den der  //
     //              Text ausgegeben werden soll                        //
     // 5. Parameter gibt an, wie die Ausgabe erfolgen soll. Die Anga-  //
     //              können über binäres Oder verknüpft werden.         //
     //              Mögliche Angaben sind:                             //
     //              DT_BOTTOM          Am unteren Randbereich des Rec- //
     //                                 tangles ausgeben. Nur zusammen  //
     //                                 mit DT_SINGLELINE möglich       //
     //              DT_CALCRECT        Berechnet des benötigten Platz  //
     //                                 bei Multiline-Ausgaben selbst-  //
     //                                 tätig.                          //
     //              DT_CENTER          Zentriert den Text in der hori- //
     //                                 zontalen Ebene                  //
     //              DT_EXPANDTABS      Erweitert Tabzeichen auf die    //
     //                                 entsprechende Länge. Vorgabe    //
     //                                 ist die Breite von 8 Blanks     //
     //              DT_EXTERNALLEADING Berechnet typographische Unter- //
     //                                 und Überlängen in den benötig-  //
     //                                 ten Platz mit ein               //
     //              DT_LEFT            Linksbündige Ausgabe im ange-   //
     //                                 gebenen Rectangle               //
     //              DT_NOCLIP          Ausgabe ohne Berücksichtigung   //
     //                                 von verdeckten Bereichen. Ist   //
     //                                 schneller, macht aber ggf.      //
     //                                 Probleme mit Childwindows       //
     //              DT_NOPREFIX        Normalerweise wird ein im Text  //
     //                                 befindlichen "&"-Zeichen als    //
     //                                 Unterstreichung für das folgen- //
     //                                 de Zeichen gewertet und ein     //
     //                                 "&&" als Ausgabe des &-Zeichens //
     //                                 Diese Methode kann über diesen  //
     //                                 Stil abgeschaltet werden        //
     //              DT_RIGHT           Rechtsbündige Ausgabe im ange-  //
     //                                 gebenen Rectangle               //
     //              DT_SINGLELINE      Einzeilige Ausgabe Zeilenvor-   //
     //                                 schubzeichen (CRLF / "\n") wer- //
     //                                 den nicht beachtet              //
     //              DT_TABSTOP         Setzt einen Tabulatorstop       //
     //              DT_TOP             Am oberen Randbereich des Rec-  //
     //                                 tangles ausgeben. Nur zusammen  //
     //                                 mit DT_SINGLELINE möglich       //
     //              DT_VCENTER         Zentriert den Text in der verti-//
     //                                 kalen Ebene. Nur zusammen mit   //
     //                                 DT_SINGLELINE möglich           //
     //              DT_WORDBREAK       Gibt an, daß nur ganze Worte    //
     //                                 umgebrochen werden sollen, bei  //
     //                                 mehrzeiliger Ausgabe.           //
     //-----------------------------------------------------------------//
     DrawText (hdc, "unglaublich, aber es geht!", -1, &rect,
               DT_SINGLELINE | DT_CENTER | DT_VCENTER);

     //-----------------------------------------------------------------//
     // Bendet die geschütze Zeichnerei                                 //
     //-----------------------------------------------------------------//
     EndPaint (hwnd, &ps);

     //-----------------------------------------------------------------//
     // Abarbeitung der WM_PAINT-Message an Windeowskernel bestätigen   //
     //-----------------------------------------------------------------//
     return (0);

//-------------------------------------------------------------------------//
// Die WM_DESTROY-Message wird empfangen, also der Wunsch des Benutzers    //
// das Programm zu ebeenden. ggf. ist dies nicht möglich, dann kann        //
// die Beendigung hier unterbunden werden. Ist die Beendigung erlaubt,     //
// So ist hier die letzte Möglichkeit Programmresourcen und Speicher-      //
// platz freizugeben, die dynamisch erzeugt worden sind !                  //
//-------------------------------------------------------------------------//
case WM_DESTROY :

     //-----------------------------------------------------------------//
     // Wenn nichts dagegen spricht, dann hier das Programmende durch-  //
     // führen. PostQuitMessage schickt eine WM-QUIT-Message an unser   //
     // eigenes Programm, so daß die Handler-Schleife in WinMain (die   //
     // GetMessage-Schleife) beendet wird                               //
     //-----------------------------------------------------------------//
     PostQuitMessage (0);

     //-----------------------------------------------------------------//
```

```
            // Abarbeitung der WM_DESTROY-Message an Windeowskernel bestätigen //
            //-----------------------------------------------------------------//
            return (0);
    }
    //-------------------------------------------------------------------------//
    // da uns alle anderen Messages nicht interessieren, müssen wir dem Kernel //
    // mitteilen, daß es die Nachricht als erledigt betrachten kann. Dazu wer- //
    // den sie formal in einem Standard-Handler des Kernels weiterverarbeitet. //
    //-------------------------------------------------------------------------//
    return DefWindowProc (hwnd, message, wParam, lParam);
}
```

8.2 Systemgrenzen

```
//*************************************************************************//
//   P2.cpp - Ermittelt einige wichtige Systemgrenzen                      //
//*************************************************************************//

#include <windows.h>

//-------------------------------------------------------------------------
inline int min (int a, int b) { if (a < b) return (a); else return (b); }
inline int max (int a, int b) { if (a > b) return (a); else return (b); }
struct { int  nIndex; char *szLabel; char *szDesc; } sysmetrics [] =
{
{SM_CXSCREEN,         "SM_CXSCREEN",         "Bildschirmbreite",              },
{SM_CYSCREEN,         "SM_CYSCREEN",         "Bildschirmhöhe",                },
{SM_CXVSCROLL,        "SM_CXVSCROLL",        "Breite vertikaler Rollpfeil",   },
{SM_CYHSCROLL,        "SM_CYHSCROLL",        "Höhe horizontaler Rollpfeil",   },
{SM_CYCAPTION,        "SM_CYCAPTION",        "Höhe der Titelleiste",          },
{SM_CXBORDER,         "SM_CXBORDER",         "Rahmenbreite",                  },
{SM_CYBORDER,         "SM_CYBORDER",         "Rahmenhöhe",                    },
{SM_CXDLGFRAME,       "SM_CXDLGFRAME",       "Rahmenbreite von Dialogen",     },
{SM_CYDLGFRAME,       "SM_CYDLGFRAME",       "Rahmenhöhe von Dialogen",       },
{SM_CYVTHUMB,         "SM_CYVTHUMB",         "Höhe der Marke in Scrollbalken",},
{SM_CXHTHUMB,         "SM_CXHTHUMB",         "Breite d.Marke in Scrollbalken",},
{SM_CXICON,           "SM_CXICON",           "Breite von Piktogrammen",       },
{SM_CYICON,           "SM_CYICON",           "Höhe von Piktogrammen",         },
{SM_CXCURSOR,         "SM_CXCURSOR",         "Breite des Cursors",            },
{SM_CYCURSOR,         "SM_CYCURSOR",         "Höhe des Cursors",              },
{SM_CYMENU,           "SM_CYMENU",           "Höhe der Menüleiste",           },
{SM_CXFULLSCREEN,     "SM_CXFULLSCREEN",     "Breite vergrößerter Fenster",   },
{SM_CYFULLSCREEN,     "SM_CYFULLSCREEN",     "Höhe vergrößerter Fenster",     },
{SM_CYKANJIWINDOW,    "SM_CYKANJIWINDOW",    "Höhe von Kanji-Fenstern",       },
{SM_MOUSEPRESENT,     "SM_MOUSEPRESENT",     "Flag <Maus vorhanden>",         },
{SM_CYVSCROLL,        "SM_CYVSCROLL",        "Höhe vertikaler Rollpfeil",     },
{SM_CXHSCROLL,        "SM_CXHSCROLL",        "Breite horizontaler Rollpfeil", },
{SM_DEBUG,            "SM_DEBUG",            "Flag <Debug-Version>",          },
{SM_SWAPBUTTON,       "SM_SWAPBUTTON",       "Flag <Mausknöpfe vertauscht>",  },
{SM_RESERVED1,        "SM_RESERVED1",        "Reserviert",                    },
{SM_RESERVED2,        "SM_RESERVED2",        "Reserviert",                    },
{SM_RESERVED3,        "SM_RESERVED3",        "Reserviert",                    },
{SM_RESERVED4,        "SM_RESERVED4",        "Reserviert",                    },
{SM_CXMIN,            "SM_CXMIN",            "Minimale Fensterbreite",        },
{SM_CYMIN,            "SM_CYMIN",            "Minimale Fensterhöhe",          },
{SM_CXSIZE,           "SM_CXSIZE",           "Breite der Zoom-Schaltflächen", },
{SM_CYSIZE,           "SM_CYSIZE",           "Höhe der Zoom-Schaltflächen",   },
{SM_CXFRAME,          "SM_CXFRAME",          "Fensterrahmenbreite",           },
{SM_CYFRAME,          "SM_CYFRAME",          "Fensterrahmenhöhe",             },
{SM_CXMINTRACK,       "SM_CXMINTRACK",       "Minimalbreite für Verschieben", },
{SM_CYMINTRACK,       "SM_CYMINTRACK",       "Minimalhöhe für Verschieben",   },
{SM_CXDOUBLECLK,      "SM_CXDOUBLECLK",      "Max. X-Bew. bei Doppelklicks",  },
{SM_CYDOUBLECLK,      "SM_CYDOUBLECLK",      "Max. Y-Bew. bei Doppelklicks",  },
{SM_CXICONSPACING,    "SM_CXICONSPACING",    "horiz. Abstand von Ikonen",     },
{SM_CYICONSPACING,    "SM_CYICONSPACING",    "vertikal. Abstand von Piktogr.",},
{SM_MENUDROPALIGNMENT,"SM_MENUDROPALIGNMENT","Menüs links oder rechts",       },
{SM_PENWINDOWS,       "SM_PENWINDOWS",       "PenWindows-Erw. installiert"    },
};

//-------------------------------------------------------------------------//
// Prototyp der Handlerfunction, die die Messages der Client-Area verarbeitet //
//-------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

//-------------------------------------------------------------------------//
// Alle Windowsprogramme beginnen mit "WinMain" statt mit "main"           //
//-------------------------------------------------------------------------//
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine,
                    int nCmdShow)
{
```

```
   static char szWindowclassname[] = "P2" ;
   HWND        hWindow;
   MSG         aMessage;
   WNDCLASS    aWndclass;

   if (!hPrevInstance)
   {
      aWndclass.style         = CS_HREDRAW | CS_VREDRAW;
      aWndclass.lpfnWndProc   = WndProc;
      aWndclass.cbClsExtra    = 0;
      aWndclass.cbWndExtra    = 0;
      aWndclass.hInstance     = hInstance;
      aWndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
      aWndclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
      aWndclass.hbrBackground = (HBRUSH)GetStockObject (WHITE_BRUSH);
      aWndclass.lpszMenuName  = NULL;
      aWndclass.lpszClassName = szWindowclassname;

      RegisterClass (&aWndclass);
   }

   hWindow = CreateWindow (szWindowclassname, "Wichtige Maße des Systems",
                           WS_OVERLAPPEDWINDOW | WS_VSCROLL | WS_HSCROLL,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL);

   ShowWindow (hWindow, nCmdShow);
   UpdateWindow (hWindow);

   while (GetMessage (&aMessage, NULL, 0, 0))
   {
      TranslateMessage (&aMessage);
      DispatchMessage (&aMessage);
   }
   return (aMessage.wParam);
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   //-------------------------------------------------------------------------//
   // die folgenden Variablen sind vom Typ STATIC INT, weil sie nur einmal,   //
   // Beginn des Programms ermittelt werden, wenn Windows die WM_CREATE-      //
   // Message sendet. Da die WndProc-Function ständig aufgerufen und wieder   //
   // verlassen wird, ist es günstiger die Daten zu erhalten                  //
   //-------------------------------------------------------------------------//
   // cxChar      - durchschnittliche Zeichenbreite (Windows-Standardfont)    //
   // cxCaps      - durchschnittliche Zeichenbreite von Großbuchstaben        //
   // cyChar      - Höhe eines Zeichens                                       //
   // cyClient    - aktuelle Höhe der Client-Area                             //
   //-------------------------------------------------------------------------//
   static int   cxChar, cxCaps, cyChar, cyClient, nVscrollPos;
   static int   nAnz = sizeof sysmetrics / sizeof sysmetrics [0];
   char         szBuffer[50];
   HDC          hdc;
   int          i, y;
   PAINTSTRUCT  ps;
   TEXTMETRIC   tm;

   switch (message)
   {
      //-------------------------------------------------------------------//
      // Windows sendet eine WM-CREATE-Message, kurz bevor das Fenster ange-  //
      // zeigt wird. Dies ist also der richtige Ort, um Daten zu ermitteln,   //
      // die dort angezeigt werden sollen. Aber Vorsicht! - jedes Fenster     //
      // bekommt diese Message nur ein einziges Mal! - Der Bereich eignet     //
      // sich also nur zur INITIALISIERUNG!                                   //
      //-------------------------------------------------------------------//
      case WM_CREATE :
           //-------------------------------------------------------------//
           // Device-Context ermitteln                                    //
           //-------------------------------------------------------------//
           hdc = GetDC (hwnd) ;

           //-------------------------------------------------------------//
           // Wichtige Anzeige-Daten des Eingestellten Schrifttyps holen      //
           // als Zwischenpuffer dient die TEXTMETRIC-Struktur                //
           //-------------------------------------------------------------//
           GetTextMetrics (hdc, &tm) ;

           //-------------------------------------------------------------//
           // Die Standardbreite eines Kleinbuchstaben entspricht üblicher-   //
           // weise der Breite des Zeichens 'x'. Dieser wird in der TEXTMETRIC//
```

```
     // Struktur festgehalten und hier auf die Static-Variable kopiert  //
     //-----------------------------------------------------------------//
     cxChar = tm.tmAveCharWidth ;

     //-----------------------------------------------------------------//
     // Die TEXTMETRIC-Struktur enthält auch Informationen darüber, ob  //
     // der Standardfont eine Proportionalschrift ist, oder ein Font    //
     // gleichbleibender Breite ist.                                    //
     // Ist das Bit TMPF_FIXED_PITCH gesetzt, so handelt es sich um     //
     // eine Schrift gleichbleibender Breite. D.h. alle Zeichen sind    //
     // gleich breit. Bei Proportionalfonts sind die Großbuchstaben in  //
     // etwa 1,5 mal so breit wie ein einfacher Buchstabe               //
     //-----------------------------------------------------------------//
     cxCaps = (tm.tmPitchAndFamily & TMPF_FIXED_PITCH   ? 3:2) * cxChar/2;

     //-----------------------------------------------------------------//
     // Die Komponente "tmHeight" enthält die Höhe des Zeichens inkl.   //
     // der Unterlängen. "tmExternalLeading" beschreibt der Platz, der  //
     // zwischen zwei Zeilen benötigt wird, damit die Buchstaben nicht  //
     // aneinander "kleben"                                             //
     //-----------------------------------------------------------------//
     cyChar = tm.tmHeight + tm.tmExternalLeading ;

     //-----------------------------------------------------------------//
     // Nicht vergessen, den DC wieder freigeben - Denn der DC ist eine //
     // limitierte Resource (es kann nur einen pro Gerät, hier also     //
     // der Bildschirm, geben                                           //
     //-----------------------------------------------------------------//
     ReleaseDC (hwnd, hdc) ;

     SetScrollRange (hwnd, SB_VERT, 0, nAnz - cyClient/cyChar, TRUE) ;
     SetScrollPos   (hwnd, SB_VERT, nVscrollPos, TRUE) ;

     //-----------------------------------------------------------------//
     // Den Wert Null zurückgeben, als Zeichen, daß keine weitere Bear- //
     // beitung notwendig ist.                                          //
     //-----------------------------------------------------------------//
     return 0;

//-------------------------------------------------------------------------//
// Windows sendet eine WM-SIZE-Message, jedesmal, wenn die Größe des       //
// Fensters geändert wird. Im Parameter "lParam" sind die neuen            //
// Fenstergrößen abgreifbar.                                               //
//-------------------------------------------------------------------------//
case WM_SIZE :
     //-----------------------------------------------------------------//
     // Die y-Ausdehnung (Höhe) ist im Highword von lParam zu finden    //
     //-----------------------------------------------------------------//
     cyClient = HIWORD (lParam) ;

     SetScrollRange (hwnd, SB_VERT, 0, nAnz - cyClient/cyChar, TRUE) ;
     SetScrollPos   (hwnd, SB_VERT, nVscrollPos, TRUE) ;
     return 0;

//-------------------------------------------------------------------------//
// Windows sendet eine VScroll-Message, wenn die Vertikale Bildlauf-       //
// leiste betätigt wird                                                    //
//-------------------------------------------------------------------------//
case WM_VSCROLL :
     //-----------------------------------------------------------------//
     // Im WPARAM der VScroll-Message steht, in welcher Form die Bild-  //
     // laufleiste betätigt wird                                        //
     //-----------------------------------------------------------------//
     switch (LOWORD(wParam))
     {
        //--------------------------------------------------------------//
        // Betätigungsform: gehe an den Beginn der Liste...             //
        //--------------------------------------------------------------//
        case SB_TOP :
             nVscrollPos = 0;
             break;

        //--------------------------------------------------------------//
        // Betätigungsform: gehe an das Ende der Liste...               //
        //--------------------------------------------------------------//
        case SB_BOTTOM :
             nVscrollPos = nAnz - cyClient/cyChar;
             break;

        //--------------------------------------------------------------//
        // Betätigungsform: gehe eine Zeile hoch...                     //
        //--------------------------------------------------------------//
        case SB_LINEUP :
```

```
            nVscrollPos -= 1;
            break;

         //-------------------------------------------------------------//
         // Betätigungsform: gehe eine Zeile runter...                  //
         //-------------------------------------------------------------//
         case SB_LINEDOWN :
            nVscrollPos += 1;
            break;

         //-------------------------------------------------------------//
         // Betätigungsform: gehe eine Seite hoch...                    //
         //-------------------------------------------------------------//
         case SB_PAGEUP :
            nVscrollPos -= cyClient / cyChar;
            break;

         //-------------------------------------------------------------//
         // Betätigungsform: gehe eine Seite runter...                  //
         //-------------------------------------------------------------//
         case SB_PAGEDOWN :
            nVscrollPos += cyClient / cyChar;
            break;

         //-------------------------------------------------------------//
         // Betätigungsform: Mit der Maus den Scrollbutton ziehen...    //
         //-------------------------------------------------------------//
         case SB_THUMBPOSITION :
            nVscrollPos = HIWORD (wParam);
            break;

         //-------------------------------------------------------------//
         // sonstige Betätigungsform: mache gar nix...                  //
         //-------------------------------------------------------------//
         default :
            break;
      }
      nVscrollPos = max (0, min (nVscrollPos, nAnz - cyClient/cyChar));
      if (nVscrollPos != GetScrollPos (hwnd, SB_VERT))
      {
         SetScrollPos (hwnd, SB_VERT, nVscrollPos, TRUE) ;
         InvalidateRect (hwnd, NULL, TRUE);
      }
      return 0;

case WM_PAINT :
      hdc = BeginPaint (hwnd, &ps) ;

      //-------------------------------------------------------------------//
      // Aufbau der Paint-Struktur:                                        //
      // hdc         Der device-context des Displays, für welches ge-      //
      //             zeichnet wird                                         //
      // fErase      Wenn dieser Wert ungleich Null ist, dann soll         //
      //             soll Windows beim Zeichnen den Hintergrund löschen    //
      // rcPaint     Hierbei handelt es sich um eine RECTANGLE-Struktur,   //
      //             die den zu zeichnenden Bereich beschreibt (linke      //
      //             obere und rechte untere Ecke                          //
      // Die restlichen Parameter sind reserviert und dürfen nur vom       //
      // Windowssystem selbst benutzt werden                               //
      //-------------------------------------------------------------------//

      //-------------------------------------------------------------------//
      // Schleife von Zeilen nPaintBeg bis nPaintEnd                       //
      //-------------------------------------------------------------------//
      for (i = 0 ; i < nAnz ; i++)
      {
         //-------------------------------------------------------------//
         // Leider sind jetzt die Zeilennummern wieder in Pixel umzu-   //
         // rechnen... gleiches gilt natürlich auch für die horizontale //
         // Ausrichtung                                                 //
         //-------------------------------------------------------------//
         y = cyChar * (1 - nVscrollPos + i) - cyChar;

         //-------------------------------------------------------------//
         // und jetzt die Textausgabe 1. Spalte                         //
         //-------------------------------------------------------------//
         TextOut (hdc, cxChar, y, sysmetrics[i].szLabel,
                  lstrlen (sysmetrics[i].szLabel)) ;

         //-------------------------------------------------------------//
         // und jetzt die Textausgabe 2. Spalte                         //
         //-------------------------------------------------------------//
         TextOut (hdc, cxChar + 22 * cxCaps, y, sysmetrics[i].szDesc,
```

```
                     lstrlen (sysmetrics[i].szDesc));

            //-------------------------------------------------------------//
            // und jetzt die Textausgabe 3. Spalte                         //
            //-------------------------------------------------------------//
            SetTextAlign (hdc, TA_RIGHT | TA_TOP);
            TextOut (hdc, cxChar + 22 * cxCaps + 40 * cxChar, y, szBuffer,
                     wsprintf (szBuffer, "%5d",
                               GetSystemMetrics (sysmetrics[i].nIndex)));
            SetTextAlign (hdc, TA_LEFT | TA_TOP);
         }
         EndPaint (hwnd, &ps);
         return 0;

   case WM_DESTROY :
         PostQuitMessage (0);
         return 0;
   }
   return DefWindowProc (hwnd, message, wParam, lParam);
}
```

8.3 Tastatur und Zeichen

```
//*************************************************************************//
//   P3.cpp - Einblick in die Tasten- und Zeichenmessages                  //
//*************************************************************************//
#include <windows.h>
#include <stdio.h>

//-------------------------------------------------------------------------
#define BLANK ' '

//-------------------------------------------------------------------------//
// Prototyp der Handlerfunction, die die Messages der Client-Area verarbeitet //
//-------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

//-------------------------------------------------------------------------//
// Globale Variablen                                                       //
//-------------------------------------------------------------------------//

RECT rect;
int  cxChar;
int  cyChar;
char szTop[] = "Botschaft      Taste Zeichen  Wdh.  OEM-Code Erw. ALT   vorher"
               " jetzt";
char szUnd[] = "_________      _____ ________ ______ ________ ____ ___   ______"
               " ________";

//-------------------------------------------------------------------------//
// Alle Windowsprogramme beginnen mit "WinMain" statt mit "main"           //
//-------------------------------------------------------------------------//
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam,
                    int nCmdShow)
{
   //----------------------------------------------------------------------//
   // Hier wieder die vier unverzichtbaren Variablen...                    //
   //----------------------------------------------------------------------//
   static char szWindowclassname [] = "P3";
   HWND        hWindow;
   MSG         aMessage;
   WNDCLASS    aWindowclass;

   if (!hPrevInstance)
   {
      aWindowclass.style         = CS_HREDRAW | CS_VREDRAW;
      aWindowclass.lpfnWndProc   = WndProc;
      aWindowclass.cbClsExtra    = 0;
      aWindowclass.cbWndExtra    = 0;
      aWindowclass.hInstance     = hInstance;
      aWindowclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
      aWindowclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
      aWindowclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);
      aWindowclass.lpszMenuName  = NULL;
      aWindowclass.lpszClassName = szWindowclassname;

      RegisterClass (&aWindowclass);
   }

   //----------------------------------------------------------------------//
   // Fenster erzeugen und anzeigen                                        //
```

```
   //------------------------------------------------------------------------//
   hWindow = CreateWindow (szWindowclassname, "Tastaturereignisse",
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL);
   ShowWindow   (hWindow, nCmdShow);
   UpdateWindow (hWindow);

   //------------------------------------------------------------------------//
   // Verarbeiten aller Botschaften an unser Programm                        //
   //------------------------------------------------------------------------//
   while (GetMessage (&aMessage, NULL, 0, 0))
   {
      TranslateMessage (&aMessage);
      DispatchMessage  (&aMessage);
   }
   //------------------------------------------------------------------------//
   // Windowskernel mitteilen, daß die WM_QUIT-Message verarbeitet wurde     //
   //------------------------------------------------------------------------//
   return (aMessage.wParam);
}

//--------------------------------------------------------------------------//
// Diese Routine wird bei jeder CHAR-Botschaft benötigt und ist daher ver-  //
// allgemeinert und ausgelagert                                             //
//--------------------------------------------------------------------------//
void ShowKey (HWND hwnd, char *szMessage, UINT wParam, LONG lParam)
{
   //------------------------------------------------------------------------//
   // Formatstring des Ausgabetextes und andere Textkonstanten               //
   //------------------------------------------------------------------------//
   static char *szFormat = {"%-14s %5d    %c %8u %9d %3s %3s %9s %8s"};
   static char szUp   [] = "gelöst";
   static char szDown [] = "gedrückt";

   //------------------------------------------------------------------------//
   // Stringpuffer für die Aufbereitung des Ausgabestrings                   //
   //------------------------------------------------------------------------//
   char        szBuffer[90];
   int         nTextlen;
   HDC         hdc;

   //------------------------------------------------------------------------//
   // Fenster um eine Zeile nach oben scrollen                               //
   //------------------------------------------------------------------------//
   ScrollWindow (hwnd, 0, -cyChar, &rect, &rect);

   //------------------------------------------------------------------------//
   // Systemfont zur Textausgabe heranziehen                                 //
   //------------------------------------------------------------------------//
   hdc = GetDC (hwnd);
   SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT));

   //------------------------------------------------------------------------//
   // Text aufbereiten                                                       //
   //------------------------------------------------------------------------//
   nTextlen = wsprintf (szBuffer, szFormat,
                        (LPSTR) szMessage, wParam,
                        (BYTE)  wParam,
                        LOWORD (lParam),
                        HIWORD (lParam) & 0xFF,
                        (LPSTR) (0x01000000 & lParam ? " * "  : ""),
                        (LPSTR) (0x20000000 & lParam ? " * "  : ""),
                        (LPSTR) (0x40000000 & lParam ? szDown : szUp),
                        (LPSTR) (0x80000000 & lParam ? szUp : szDown));

   //------------------------------------------------------------------------//
   // Text in der untersten Zeile ausgeben                                   //
   //------------------------------------------------------------------------//
   TextOut (hdc, cxChar, rect.bottom - cyChar, szBuffer, nTextlen);
   ReleaseDC (hwnd, hdc);

   //------------------------------------------------------------------------//
   // Windows bescheid geben, daß der Bereich nicht neu gezeichnet werden muß //
   //------------------------------------------------------------------------//
   ValidateRect (hwnd, NULL);
}

//--------------------------------------------------------------------------//
// Messagehandler der Client-Area                                           //
//--------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
```

```
{
   HDC         hdc;
   PAINTSTRUCT ps;
   TEXTMETRIC  tm;

   switch (message)
   {
      //--------------------------------------------------------------------//
      // Windows sendet eine WM-CREATE-Message, kurz bevor das Fenster ange-  //
      // zeigt wird. Der Bereich eignet sich nur zur INITIALISIERUNG!         //
      //--------------------------------------------------------------------//
      case WM_CREATE :
           //--------------------------------------------------------------//
           // Höhe einer Zeile feststellen, damit bei jeder Tastenbotschaft   //
           // um genau diesen Betrag nach oben gescrollt werden kann          //
           //--------------------------------------------------------------//
           hdc = GetDC (hwnd);
           SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT));
           GetTextMetrics (hdc, &tm);
           cxChar = tm.tmAveCharWidth;
           cyChar = tm.tmHeight;
           ReleaseDC (hwnd, hdc);
           //--------------------------------------------------------------//
           // Oberen Rand des Fensterbereiches für Überschrift vom Scroll-    //
           // bereich abziehen !                                              //
           //--------------------------------------------------------------//
           rect.top = 3 * cyChar / 2;
           return (0);

      //--------------------------------------------------------------------//
      // Windows schickt eine WM-SIZE-Message, wenn die Größe des Anzeige-    //
      // fensters verändert wird                                             //
      //--------------------------------------------------------------------//
      case WM_SIZE :
           //--------------------------------------------------------------//
           // Ausgabe-Rectangle neu setzen                                    //
           //--------------------------------------------------------------//
           rect.right  = LOWORD (lParam);
           rect.bottom = HIWORD (lParam);
             //----------------------------------------------------------------//
           // Fenster neu zeichnen, UpdateWindow generiert eine neue WM_PAINT //
           // Message                                                         //
           //--------------------------------------------------------------//
           UpdateWindow (hwnd);
           return (0);

      //--------------------------------------------------------------------//
      // Windows schickt eine WM-Paint-Message, wenn der Programmierer es als //
      // inhaltlich erforderlich ansieht, Bildschirmbereiche neu zu zeichnen, //
      // weil sie verändert wurden                                           //
      //--------------------------------------------------------------------//
      case WM_PAINT :
           //--------------------------------------------------------------//
           // den neu zu zeichnenden Bereich festlegen, hier: alles neu       //
           //--------------------------------------------------------------//
           InvalidateRect (hwnd, NULL, TRUE);
           hdc = BeginPaint (hwnd, &ps);
           SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT));
           SetBkMode (hdc, TRANSPARENT);
           //--------------------------------------------------------------//
           // Überschrift neu ausgeben                                        //
           //--------------------------------------------------------------//
           TextOut (hdc, cxChar, cyChar / 2, szTop, (sizeof szTop) - 1);
           TextOut (hdc, cxChar, cyChar / 2, szUnd, (sizeof szUnd) - 1);
           EndPaint (hwnd, &ps);
           return (0);

      //--------------------------------------------------------------------//
      // WM-KEYDOWN wird gesendet, wenn eine nicht-Systemtaste gedrückt wird  //
      //--------------------------------------------------------------------//
      case WM_KEYDOWN :
           ShowKey (hwnd, "WM_KEYDOWN", wParam, lParam);
           return (0);

      //--------------------------------------------------------------------//
      // WM-KEYDOWN wird gesendet, wenn eine nicht-Systemtaste gelöst wird    //
      //--------------------------------------------------------------------//
      case WM_KEYUP :
           ShowKey (hwnd, "WM_KEYUP", wParam, lParam);
           return (0);

      //--------------------------------------------------------------------//
      // WM-CHAR sendet den modifizierten Zeichencode einer gedrückten Taste  //
```

```
      //------------------------------------------------------------------------//
      case WM_CHAR :
           ShowKey (hwnd, "WM_CHAR", wParam, lParam);
           return (0);

      //------------------------------------------------------------------------//
      // WM-DEADCHAR wird gesendet, wenn eine weitere Taste gedrückt werden   //
      // muß, um den Zeichencode eindeutig zu spezifizieren (Akzent-Tasten)   //
      //------------------------------------------------------------------------//
      case WM_DEADCHAR :
           ShowKey (hwnd, "WM_DEADCHAR", wParam, lParam);
           return (0);

      //------------------------------------------------------------------------//
      // WM-SYSKEYDOWN wird gesendet, wenn eine Steuerungstaste (ALT) ge-     //
      // drückt wird                                                          //
      //------------------------------------------------------------------------//
      // SYSTEM-Tasten werden von der Default-Windows-Message-Procedure ver-  //
      // arbeitet, daher NICHT RETURN(0) sondern BREAK !!!!!                  //
      //------------------------------------------------------------------------//
      case WM_SYSKEYDOWN :
           ShowKey (hwnd, "WM_SYSKEYDOWN", wParam, lParam);
           break;

      //------------------------------------------------------------------------//
      // WM-SYSKEYUP wird gesendet, wenn eine Steuerungstaste (ALT) gelöst    //
      // wird                                                                 //
      //------------------------------------------------------------------------//
      // SYSTEM-Tasten werden von der Default-Windows-Message-Procedure ver-  //
      // arbeitet, daher NICHT RETURN(0) sondern BREAK !!!!!                  //
      //------------------------------------------------------------------------//
      case WM_SYSKEYUP :
           ShowKey (hwnd, "WM_SYSKEYUP", wParam, lParam);
           break;

      //------------------------------------------------------------------------//
      // WM-CHAR sendet den Zeichencode einer gedrückten System-Taste         //
      //------------------------------------------------------------------------//
      // SYSTEM-Tasten werden von der Default-Windows-Message-Procedure ver-  //
      // arbeitet, daher NICHT RETURN(0) sondern BREAK !!!!!                  //
      //------------------------------------------------------------------------//
      case WM_SYSCHAR :
           ShowKey (hwnd, "WM_SYSCHAR", wParam, lParam);
           break;

      //------------------------------------------------------------------------//
      // WM-SYSDEADCHAR wird gesendet, wenn eine weitere Taste gedrückt       //
      // werden muß, um den Zeichencode eindeutig zu spezifizieren            //
      //------------------------------------------------------------------------//
      // SYSTEM-Tasten werden von der Default-Windows-Message-Procedure ver-  //
      // arbeitet, daher NICHT RETURN(0) sondern BREAK !!!!!                  //
      //------------------------------------------------------------------------//
      case WM_SYSDEADCHAR :
           ShowKey (hwnd, "WM_SYSDEADCHAR", wParam, lParam);
           break;

      //------------------------------------------------------------------------//
      // WM-DESTROY wird gesendet, wenn das Programm beendet werden soll      //
      //------------------------------------------------------------------------//
      case WM_DESTROY :
           PostQuitMessage (0);
           return (0);
   }
   return DefWindowProc (hwnd, message, wParam, lParam);
}
```

8.4 Maus 1 - Graphikbefehle

```
//************************************************************************//
//   P4.cpp - Maus-Demonstrationsprogramm                                 //
//************************************************************************//
#include <windows.h>

//------------------------------------------------------------------------
#define MAXPOINTS 1000

//------------------------------------------------------------------------//
// Prototyp der Handlerfunction, die die Messages der Client-Area verarbeitet //
//------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

//------------------------------------------------------------------------//
// Alle Windowsprogramme beginnen mit "WinMain" statt mit "main"          //
//------------------------------------------------------------------------//
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine,
                    int nCmdShow)
{
   //---------------------------------------------------------------------//
   // Hier wieder die vier unverzichtbaren Variablen...                   //
   //---------------------------------------------------------------------//
   static char szWindowclassname [] = "P4";
   HWND        hWindow;
   MSG         aMessage;
   WNDCLASS    aWindowclass;

   if (!hPrevInstance)
   {
      aWindowclass.style         = CS_HREDRAW | CS_VREDRAW;
      aWindowclass.lpfnWndProc   = WndProc;
      aWindowclass.cbClsExtra    = 0;
      aWindowclass.cbWndExtra    = 0;
      aWindowclass.hInstance     = hInstance;
      aWindowclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
      aWindowclass.hCursor       = LoadCursor (NULL, IDC_CROSS);
      aWindowclass.hbrBackground = (HBRUSH)GetStockObject (WHITE_BRUSH);
      aWindowclass.lpszMenuName  = NULL;
      aWindowclass.lpszClassName = szWindowclassname;

      RegisterClass (&aWindowclass);
   }
   //---------------------------------------------------------------------//
   // Fenster erzeugen und anzeigen                                       //
   //---------------------------------------------------------------------//
   hWindow = CreateWindow (szWindowclassname, "Maus-Demo",
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL);
   ShowWindow   (hWindow, nCmdShow);
   UpdateWindow (hWindow);

   //---------------------------------------------------------------------//
   // Verarbeiten aller Botschaften an unser Programm                     //
   //---------------------------------------------------------------------//
   while (GetMessage (&aMessage, NULL, 0, 0))
   {
      TranslateMessage (&aMessage);
      DispatchMessage  (&aMessage);
   }
   //---------------------------------------------------------------------//
   // Windowskernel mitteilen, daß die WM_QUIT-Message verarbeitet wurde  //
   //---------------------------------------------------------------------//
   return (aMessage.wParam);
}

//------------------------------------------------------------------------//
// Messagehandler der Client-Area                                         //
//------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   //---------------------------------------------------------------------//
   // Lokale Variablen                                                    //
   //---------------------------------------------------------------------//
   // Points ist ein Array, daß MAXPOINTS x- und y-Koordinaten speichern kann //
   // nCount enthält die relevante Anzahl von gesetzten Punkten           //
   // STATIC-VARIABLEN werden beim Verlassen der Funktion im "Gedächtnis" be- //
```

```
// halten und stehen beim nächsten Eintritt in die Fzunktion wieder zur    //
// Verfügung                                                               //
//-------------------------------------------------------------------------//
static POINT points[MAXPOINTS];
static short nCount;
HDC          hdc;
PAINTSTRUCT  ps;
short        i, j;


switch (message)
{
   //-----------------------------------------------------------------------//
   // Windows schickt eine WM-LBUTTONDOWN-Message, wenn die linke Maus-    //
   // taste gedrückt wird                                                   //
   //-----------------------------------------------------------------------//
   case WM_LBUTTONDOWN :
        //-------------------------------------------------------------------//
        // Beim drücken der linken Maustaste wieder von vorne beginnen,    //
        // d.h. der Counter wird zurückgesetzt und die gesamte Client-Area //
        // invalidiert                                                       //
        //-------------------------------------------------------------------//
        nCount = 0;
        InvalidateRect (hwnd, NULL, TRUE);
        return (0);

   //-----------------------------------------------------------------------//
   // Windows schickt eine WM-MOUSEMOVE-Message, wenn die Maus bewegt wird //
   //-----------------------------------------------------------------------//
   case WM_MOUSEMOVE :
        //-------------------------------------------------------------------//
        // im wParam ist ein Flag gesetzt, wenn die linke Maustaste ge-    //
        // drückt gehalten wird außerdem ist die maximale Anzahl der       //
        // speicherbaren Punkte zu beachten !                                //
        //-------------------------------------------------------------------//
        if ((wParam & MK_LBUTTON) && (nCount < MAXPOINTS))
        {
           //----------------------------------------------------------------//
           // wenn beide Bedingungen erfüllt sind, dann den Punkt in die   //
           // Punktliste mit aufnehmen. die Koordinaten können dem lParam  //
           // entnommen werden                                              //
           //----------------------------------------------------------------//
           // unter 32-Bit Windows gibt es das Makro MAKEPOINT nicht mehr  //
           // man muß die Koordinaten daher per Hand zuweisen               //
           //----------------------------------------------------------------//
             points[nCount].x = LOWORD(lParam);
             points[nCount].y = HIWORD(lParam);
           nCount++;

           //----------------------------------------------------------------//
           // Damit man den Punkt auch sehen kann, setzen wir an der ent-  //
           // sprechenden Stelle ein Pixel                                  //
           //----------------------------------------------------------------//
           hdc = GetDC (hwnd);
           SetPixel (hdc, LOWORD (lParam), HIWORD (lParam), 0L);
           ReleaseDC (hwnd, hdc);
        }
        return (0);

   //-----------------------------------------------------------------------//
   // Windows schickt eine WM-LBUTTONUP-Message, wenn die linke Maustaste  //
   // losgelassen wird                                                      //
   //-----------------------------------------------------------------------//
   case WM_LBUTTONUP :
        //-------------------------------------------------------------------//
        // gesamte Clientarea neu zeichnen                                   //
        //-------------------------------------------------------------------//
        InvalidateRect (hwnd, NULL, FALSE);
        return (0);

   //-----------------------------------------------------------------------//
   // Zeichnen                                                              //
   //-----------------------------------------------------------------------//
   case WM_PAINT :
        {
            hdc = BeginPaint (hwnd, &ps);
            int k = random (5);
           // k = 4;
           //----------------------------------------------------------------//
           // alle Punkte miteinander verbinden, wenn nCount 0 ist, wird   //
           // diese Schleife nicht ausgeführt !                             //
           //----------------------------------------------------------------//
```

```
        if (k==0)
        {
             for (i = 0; i < nCount - 1; i++)
           {
              for (j = i; j < nCount; j++)
              {
                 //-----------------------------------------------------//
                   // unter 32-Bit Windows existiert die Funktion MoveTo  //
                 // nicht mehr, man nehme stattdessen MoveToEx und setze//
                 // den letzten Parameter auf NULL                      //
                 //-----------------------------------------------------//
                   MoveToEx (hdc, points[i].x, points[i].y, NULL);
                 LineTo (hdc, points[j].x, points[j].y);
              }
           }
        }
        if (k==1)
        {
           SetPolyFillMode (hdc, WINDING);

             for (i = 0; i < nCount - 1; i++)
           {
              COLORREF cref = RGB (random(255),random(255),random (255));
              HBRUSH hbr = CreateSolidBrush (cref);
              SelectObject (hdc, hbr);

                switch (random (3))
              {
                   case 0: Rectangle (hdc, points[i].x, points[i].y,
                                      points[i].x+random(100),
                                      points[i].y+random(100)); break;
                 case 1: Ellipse (hdc, points[i].x, points[i].y,
                                      points[i].x+random(100),
                                      points[i].y+random(100)); break;
                 case 2: k = random (100);
                           Ellipse (hdc, points[i].x, points[i].y,
                                      points[i].x+k,
                                      points[i].y+k); break;
              }
              DeleteObject (hbr);
           }
        }

        if (k==2)
        {
             MoveToEx (hdc, points[0].x, points[0].y, NULL);
             for (i = 0; i < nCount - 1; i++)
                LineTo (hdc, points[i].x, points[i].y);
        }

        if (k==3)
        {
           SetPolyFillMode (hdc, WINDING);
           COLORREF cref = RGB (random(255),random(255),random (255));
           HBRUSH hbr = CreateSolidBrush (cref);
           SelectObject (hdc, hbr);
           Polygon (hdc, points, nCount);
           DeleteObject (hbr);
        }

        if (k==4)
        {
           SetPolyFillMode (hdc, WINDING);
             for (i = 0; i < nCount; i++)
           {
              COLORREF cref = RGB (random(255),random(255),random (255));
                HBRUSH hbr = CreateSolidBrush (cref);
              SelectObject (hdc, hbr);
              RoundRect (hdc, points[i].x, points[i].y,
                         points[i].x+random(100),
                         points[i].y+random(100),
                           random(50), random(50));
                DeleteObject (hbr);
           }
        }
     }
     EndPaint (hwnd, &ps);
     return (0);

//---------------------------------------------------------------------//
// Programm beenden                                                    //
//---------------------------------------------------------------------//
case WM_DESTROY :
```

```
            PostQuitMessage (0);
            return (0);
   }
   return DefWindowProc (hwnd, message, wParam, lParam);
}
```

8.5 Maus 2 – Positionsberechnungen 1

```
//*************************************************************************//
//   P5.cpp - Maus-Demo 2                                                  //
//*************************************************************************//
#include <windows.h>

//-------------------------------------------------------------------------
#define TEILUNGEN 3

//-------------------------------------------------------------------------//
// Prototyp der Handlerfunction, die die Messages der Client-Area verarbeitet //
//-------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

//-------------------------------------------------------------------------//
// Alle Windowsprogramme beginnen mit "WinMain" statt mit "main"           //
//-------------------------------------------------------------------------//
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine,
                    int nCmdShow)
{
   //----------------------------------------------------------------------//
   // Hier wieder die vier unverzichtbaren Variablen...                    //
   //----------------------------------------------------------------------//
   static char szWindowclassname [] = "P5";
   HWND        hWindow;
   MSG         aMessage;
   WNDCLASS    aWindowclass;

   if (!hPrevInstance)
   {
      aWindowclass.style         = CS_HREDRAW | CS_VREDRAW;
      aWindowclass.lpfnWndProc   = WndProc;
      aWindowclass.cbClsExtra    = 0;
      aWindowclass.cbWndExtra    = 0;
      aWindowclass.hInstance     = hInstance;
      aWindowclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
      aWindowclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
      aWindowclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);
      aWindowclass.lpszMenuName  = NULL;
      aWindowclass.lpszClassName = szWindowclassname;

      RegisterClass (&aWindowclass);
   }
   //----------------------------------------------------------------------//
   // Fenster erzeugen und anzeigen                                        //
   //----------------------------------------------------------------------//
   hWindow = CreateWindow (szWindowclassname, "Maus-Demo 2",
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL);
   ShowWindow   (hWindow, nCmdShow);
   UpdateWindow (hWindow);

   //----------------------------------------------------------------------//
   // Verarbeiten aller Botschaften an unser Programm                      //
   //----------------------------------------------------------------------//
   while (GetMessage (&aMessage, NULL, 0, 0))
   {
      TranslateMessage (&aMessage);
      DispatchMessage  (&aMessage);
   }
   //----------------------------------------------------------------------//
   // Windowskernel mitteilen, daß die WM_QUIT-Message verarbeitet wurde   //
   //----------------------------------------------------------------------//
   return (aMessage.wParam);
}

//-------------------------------------------------------------------------//
// Messagehandler der Client-Area                                          //
//-------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   static int   fState[TEILUNGEN][TEILUNGEN];
```

```
static int   cxBlock, cyBlock;
HDC          hdc;
PAINTSTRUCT  ps;
RECT         rect;
int          x, y;
static int   what = 1;

switch (message)
{
   //--------------------------------------------------------------------//
   // Größe des Fensters wird verändert                                  //
   //--------------------------------------------------------------------//
   case WM_SIZE :
        //---------------------------------------------------------------//
        // neue Breite eines Karos feststellen                           //
        //---------------------------------------------------------------//
        cxBlock = LOWORD (lParam) / TEILUNGEN;
        //---------------------------------------------------------------//
        // neue Höhe eines Karos feststellen                             //
        //---------------------------------------------------------------//
        cyBlock = HIWORD (lParam) / TEILUNGEN;
        return (0);

   //--------------------------------------------------------------------//
   // linke Maustaste wird gedrückt                                      //
   //--------------------------------------------------------------------//
   case WM_LBUTTONDOWN :
        //---------------------------------------------------------------//
        // feststellen, in welchem Karo sich der Mauszeiger befindet, in- //
        // dem die aktuellen Koordinaten durch die xy-Größe des Karos ge- //
        // teilt werden                                                  //
        //---------------------------------------------------------------//
        x = LOWORD (lParam) / cxBlock;
        y = HIWORD (lParam) / cyBlock;

        //---------------------------------------------------------------//
        // Wenn es sich beim Ergebnis um einen gültigen Karoindex handelt //
        //---------------------------------------------------------------//
        if ((x < TEILUNGEN) && (y < TEILUNGEN))
        {
           //------------------------------------------------------------//
           // Status des Karos über EXCLUSIVE-OR umschalten !            //
           //------------------------------------------------------------//

           if (!fState [x][y])
           {
              fState [x][y] = what;
                if (what == 1) what = 2;
              else what = 1;
           }

           //------------------------------------------------------------//
           // für die nachfolgende Graphik-Operation das Rectangle des   //
           // Karos feststellen                                          //
           //------------------------------------------------------------//
           rect.left   = x * cxBlock;
           rect.top    = y * cyBlock;
           rect.right  = (x + 1) * cxBlock;
           rect.bottom = (y + 1) * cyBlock;
           //------------------------------------------------------------//
           // genau dieses Rectangle invalidieren, so daß es neu gezeich- //
           // net werden muß                                             //
           //------------------------------------------------------------//
           InvalidateRect (hwnd, &rect, FALSE);
        }
        //---------------------------------------------------------------//
        // Wenn es nicht klappt, dann piepen!                            //
         //--------------------------------------------------------------//
        else MessageBeep (0xFFFFFFFF);
        return (0);

   //--------------------------------------------------------------------//
   // neu zeichnen                                                       //
   //--------------------------------------------------------------------//
   case WM_PAINT :
        hdc = BeginPaint (hwnd, &ps);
        //---------------------------------------------------------------//
        // Schleife über alle Karos                                      //
         //--------------------------------------------------------------//
        for (x = 0; x < TEILUNGEN; x++)
        {
           for (y = 0; y < TEILUNGEN; y++)
           {
```

```
                //-------------------------------------------------------------//
                // Rectangle des Karos berechnen                               //
                 //-------------------------------------------------------------//
                Rectangle (hdc, x * cxBlock, y * cyBlock,
                           (x + 1) * cxBlock, (y + 1) * cyBlock);

                //-------------------------------------------------------------//
                // Wenn dieses Karo gesetzt ist, dann Kreuz zeichnen           //
                 //-------------------------------------------------------------//
                if (fState [x][y] == 1)
                {
                   MoveToEx (hdc,  x    * cxBlock,  y    * cyBlock, NULL);
                   LineTo   (hdc, (x+1) * cxBlock, (y+1) * cyBlock);
                   MoveToEx (hdc,  x    * cxBlock, (y+1) * cyBlock, NULL);
                   LineTo   (hdc, (x+1) * cxBlock,  y    * cyBlock);
                }
                if (fState [x][y] == 2)
                {
                   Ellipse  (hdc, x*cxBlock+1, y*cyBlock+1,
                             (x+1)*cxBlock-1, (y+1)*cyBlock-1);
                }
             }
          }
          EndPaint (hwnd, &ps);
          return (0);

     //-------------------------------------------------------------------------//
     // Programm beenden                                                        //
     //-------------------------------------------------------------------------//
     case WM_DESTROY :
          PostQuitMessage (0);
          return (0);
   }
   return DefWindowProc (hwnd, message, wParam, lParam);
}
```

8.6 Maus 3 – Positionsberechnungen 2

```
//*************************************************************************//
//   P5-1.cpp - Maus-Demo 3                                                //
//*************************************************************************//
#include <windows.h>

//-------------------------------------------------------------------------
#define TEILUNGEN 5

//-------------------------------------------------------------------------//
// Inline-Funktionen (Makros)                                              //
//-------------------------------------------------------------------------//
inline int min (int a, int b) { if (a < b) return (a); else return (b); }
inline int max (int a, int b) { if (a > b) return (a); else return (b); }

//-------------------------------------------------------------------------//
// Prototyp der Handlerfunction, die die Messages der Client-Area verarbeitet //
//-------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

//-------------------------------------------------------------------------//
// Alle Windowsprogramme beginnen mit "WinMain" statt mit "main"           //
//-------------------------------------------------------------------------//
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine,
                    int nCmdShow)
{
   //----------------------------------------------------------------------//
   // Hier wieder die vier unverzichtbaren Variablen...                    //
   //----------------------------------------------------------------------//
   static char szWindowclassname [] = "P5_1";
   HWND        hWindow;
   MSG         aMessage;
   WNDCLASS    aWindowclass;

   if (!hPrevInstance)
   {
      aWindowclass.style         = CS_HREDRAW | CS_VREDRAW;
      aWindowclass.lpfnWndProc   = WndProc;
      aWindowclass.cbClsExtra    = 0;
      aWindowclass.cbWndExtra    = 0;
      aWindowclass.hInstance     = hInstance;
      aWindowclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
      aWindowclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
      aWindowclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);
      aWindowclass.lpszMenuName  = NULL;
      aWindowclass.lpszClassName = szWindowclassname;

      RegisterClass (&aWindowclass);
   }
   //----------------------------------------------------------------------//
   // Fenster erzeugen und anzeigen                                        //
   //----------------------------------------------------------------------//
   hWindow = CreateWindow (szWindowclassname, "Maus-Demo 2.1",
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL);
   ShowWindow   (hWindow, nCmdShow);
   UpdateWindow (hWindow);

   //----------------------------------------------------------------------//
   // Verarbeiten aller Botschaften an unser Programm                      //
   //----------------------------------------------------------------------//
   while (GetMessage (&aMessage, NULL, 0, 0))
   {
      TranslateMessage (&aMessage);
      DispatchMessage  (&aMessage);
   }
   //----------------------------------------------------------------------//
   // Windowskernel mitteilen, daß die WM_QUIT-Message verarbeitet wurde   //
   //----------------------------------------------------------------------//
   return (aMessage.wParam);
}

//-------------------------------------------------------------------------//
// Messagehandler der Client-Area                                          //
//-------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
```

```
static BOOL  fState[TEILUNGEN][TEILUNGEN];
static int   cxBlock, cyBlock;
HDC          hdc;
PAINTSTRUCT  ps;
RECT         rect;
int          x, y;
POINT        point;

switch (message)
{
   //---------------------------------------------------------------------//
   // Größe des Fensters wird verändert                                   //
   //---------------------------------------------------------------------//
   case WM_SIZE :
            cxBlock = LOWORD (lParam) / TEILUNGEN;
            cyBlock = HIWORD (lParam) / TEILUNGEN;
            return (0);

   //---------------------------------------------------------------------//
   // Ein Fenster bekommt eine SETFOCUS-Message, wenn es den KEYBOARD-    //
   // Focus (Cursor) erhält                                               //
   //---------------------------------------------------------------------//
   case WM_SETFOCUS :
        //-----------------------------------------------------------------//
        // Cursor zeigen                                                   //
        //-----------------------------------------------------------------//
        ShowCursor (TRUE);
        return (0);

   //---------------------------------------------------------------------//
   // Ein Fenster bekommt eine KILLFOCUS-Message, wenn es den KEYBOARD-   //
   // Focus (Cursor) an ein anderes Fenster abgibt                        //
   //---------------------------------------------------------------------//
   case WM_KILLFOCUS :
        //-----------------------------------------------------------------//
        // Cursor verbergen                                                //
        //-----------------------------------------------------------------//
        ShowCursor (FALSE);
        return (0);

   //---------------------------------------------------------------------//
   // Message: Taste wurde gedrückt                                       //
   //---------------------------------------------------------------------//
   case WM_KEYDOWN :
        //-----------------------------------------------------------------//
        // Bildschirmkoordinaten des Cursors ermitteln (= in elchem Karo?) //
        //-----------------------------------------------------------------//
        GetCursorPos (&point);

        //-----------------------------------------------------------------//
        // aus Bildschirmkoordinaten Fensterkoordinaten machen             //
        //-----------------------------------------------------------------//
        ScreenToClient (hwnd, &point);

        //-----------------------------------------------------------------//
        // Entsprechendes Karo ermitteln (karoindex)                       //
        //-----------------------------------------------------------------//
        x = max (0, min (TEILUNGEN - 1, point.x / cxBlock));
        y = max (0, min (TEILUNGEN - 1, point.y / cyBlock));

        //-----------------------------------------------------------------//
        // Auswerten der gedrückten Taste                                  //
        // Return/Eingabe und Leertaste simulieren einen Click der linken  //
        // Maustaste, indem sie einfach die entsprechende Message senden...//
          //---------------------------------------------------------------//
        switch (wParam)
        {
           case VK_UP     : y--;                 break;
           case VK_DOWN   : y++;                 break;
           case VK_LEFT   : x--;                 break;
           case VK_RIGHT  : x++;                 break;
           case VK_PRIOR  : y = 0;             break;
           case VK_NEXT   : y = TEILUNGEN - 1; break;
           case VK_HOME   : x = 0; y = 0;      break;
           case VK_END    : x = TEILUNGEN - 1;
                            y = TEILUNGEN - 1; break;
           case VK_ESCAPE : SendMessage (hwnd, WM_DESTROY, 0, 0L);
                            break;
           case VK_RETURN :
           case VK_SPACE  : SendMessage (hwnd, WM_LBUTTONDOWN, MK_LBUTTON,
                                         MAKELONG (x*cxBlock, y*cyBlock));
                            break;
        }
```

```
     //---------------------------------------------------------------//
     // Für die Cursorpositionierung ist die Mittelposition des Karos //
     // zu berechnen                                                  //
     //---------------------------------------------------------------//
     x = (x + TEILUNGEN) % TEILUNGEN;
     y = (y + TEILUNGEN) % TEILUNGEN;
     point.x = x * cxBlock + cxBlock / 2;
     point.y = y * cyBlock + cyBlock / 2;

     //---------------------------------------------------------------//
     // Fensterkoordinaten wieder in Bildschirmkoordinaten umsetzen und //
     // Cursorposition ändern                                         //
     //---------------------------------------------------------------//
     ClientToScreen (hwnd, &point);
     SetCursorPos (point.x, point.y);
     return (0);

//-----------------------------------------------------------------------//
// linke Maustaste wird gedrückt                                         //
//-----------------------------------------------------------------------//
case WM_LBUTTONDOWN :
     //---------------------------------------------------------------//
     // feststellen, in welchem Karo sich der Mauszeiger befindet, in- //
     // dem die aktuellen Koordinaten durch die xy-Größe des Karos ge- //
     // teilt werden                                                  //
     //---------------------------------------------------------------//
     x = LOWORD (lParam) / cxBlock;
     y = HIWORD (lParam) / cyBlock;

     //---------------------------------------------------------------//
     // Wenn es sich beim Ergebnis um einen gültigen Karoindex handelt //
     //---------------------------------------------------------------//
     if ((x < TEILUNGEN) && (y < TEILUNGEN))
     {
        //---------------------------------------------------------//
        // Status des Karos über EXCLUSIVE-OR umschalten !         //
        //---------------------------------------------------------//
        fState [x][y] ^= 1;

        //---------------------------------------------------------//
        // für die nachfolgende Graphik-Operation das Rectangle des //
        // Karos feststellen                                       //
        //---------------------------------------------------------//
        rect.left   = x * cxBlock;
        rect.top    = y * cyBlock;
        rect.right  = (x + 1) * cxBlock;
        rect.bottom = (y + 1) * cyBlock;
        //---------------------------------------------------------//
        // genau dieses Rectangle invalidieren, so daß es neu gezeich- //
        // net werden muß                                          //
        //---------------------------------------------------------//
        InvalidateRect (hwnd, &rect, FALSE);
     }
     //---------------------------------------------------------------//
     // Wenn es nicht klappt, dann piepen!                            //
      //-------------------------------------------------------------//
     else MessageBeep (0xFFFFFFFF);
     return (0);

//-----------------------------------------------------------------------//
// neu zeichnen                                                          //
//-----------------------------------------------------------------------//
case WM_PAINT :
     hdc = BeginPaint (hwnd, &ps);
     //---------------------------------------------------------------//
     // Schleife über alle Karos                                      //
      //-------------------------------------------------------------//
     for (x = 0; x < TEILUNGEN; x++)
     {
        for (y = 0; y < TEILUNGEN; y++)
        {
           //------------------------------------------------------//
           // Rectangle des Karos berechnen                        //
            //-----------------------------------------------------//
           Rectangle (hdc, x * cxBlock, y * cyBlock,
                           (x + 1) * cxBlock, (y + 1) * cyBlock);

           //------------------------------------------------------//
           // Wenn dieses Karo gesetzt ist, dann Kreuz zeichnen    //
            //-----------------------------------------------------//
           if (fState [x][y])
           {
```

```
                MoveToEx (hdc,  x    * cxBlock,  y    * cyBlock, NULL);
                LineTo   (hdc, (x+1) * cxBlock, (y+1) * cyBlock);
                MoveToEx (hdc,  x    * cxBlock, (y+1) * cyBlock, NULL);
                LineTo   (hdc, (x+1) * cxBlock,  y    * cyBlock);
             }
          }
       }
       EndPaint (hwnd, &ps);
       return (0);

    //-------------------------------------------------------------------------//
    // Programm beenden                                                        //
    //-------------------------------------------------------------------------//
    case WM_DESTROY :
       PostQuitMessage (0);
       return (0);
  }
  return DefWindowProc (hwnd, message, wParam, lParam);
}
```

8.7 Maus 4 – Unterfenster

```
//*************************************************************************//
//   P5-2.cpp - Maus-Demo 4                                                  //
//*************************************************************************//
#include <windows.h>

#define TEILUNGEN 10

//-------------------------------------------------------------------------//
// Inline-Funktionen (Makros)                                              //
//-------------------------------------------------------------------------//
inline int min (int a, int b) { if (a < b) return (a); else return (b); }
inline int max (int a, int b) { if (a > b) return (a); else return (b); }

//-------------------------------------------------------------------------//
// Prototyp der Handlerfunktionen Client-Area und Child-Windows            //
//-------------------------------------------------------------------------//
LRESULT CALLBACK WndProc      (HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK ChildWndProc (HWND, UINT, WPARAM, LPARAM);

char szChildClassName[] = "P5_2_Child";

//-------------------------------------------------------------------------//
// Alle Windowsprogramme beginnen mit "WinMain" statt mit "main"           //
//-------------------------------------------------------------------------//
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine,
                    int nCmdShow)
{
   //----------------------------------------------------------------------//
   // Hier wieder die vier unverzichtbaren Variablen...                    //
   //----------------------------------------------------------------------//
   static char szWindowclassname [] = "P5_2";
   HWND        hWindow;
   MSG         aMessage;
   WNDCLASS    aWindowclass;

   if (!hPrevInstance)
   {
      //-------------------------------------------------------------------//
      // Registrierung der Client-Area                                     //
      //-------------------------------------------------------------------//
      aWindowclass.style         = CS_HREDRAW | CS_VREDRAW;
      aWindowclass.lpfnWndProc   = WndProc;
      aWindowclass.cbClsExtra    = 0;
      aWindowclass.cbWndExtra    = 0;
      aWindowclass.hInstance     = hInstance;
      aWindowclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
      aWindowclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
      aWindowclass.hbrBackground = (HBRUSH)GetStockObject (WHITE_BRUSH);
      aWindowclass.lpszMenuName  = NULL;
      aWindowclass.lpszClassName = szWindowclassname;

      RegisterClass (&aWindowclass);

      //-------------------------------------------------------------------//
      // Registrierung der Child-Windowsklasse                             //
      //-------------------------------------------------------------------//
      aWindowclass.lpfnWndProc   = ChildWndProc;
      aWindowclass.cbWndExtra    = sizeof (WORD);
      aWindowclass.hIcon         = NULL;
      aWindowclass.lpszClassName = szChildClassName;

      RegisterClass (&aWindowclass);
   }

   //----------------------------------------------------------------------//
   // Fenster erzeugen und anzeigen                                        //
   //----------------------------------------------------------------------//
   hWindow = CreateWindow (szWindowclassname, "Maus-Demo 2.2",
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL);
   ShowWindow   (hWindow, nCmdShow);
   UpdateWindow (hWindow);

   //----------------------------------------------------------------------//
   // Verarbeiten aller Botschaften an unser Programm                      //
   //----------------------------------------------------------------------//
```

```
   while (GetMessage (&aMessage, NULL, 0, 0))
   {
      TranslateMessage (&aMessage);
      DispatchMessage  (&aMessage);
   }
   //-------------------------------------------------------------------------//
   // Windowskernel mitteilen, daß die WM_QUIT-Message verarbeitet wurde      //
   //-------------------------------------------------------------------------//
   return (aMessage.wParam);
}

//-------------------------------------------------------------------------//
// Messagehandler der Client-Area                                          //
//-------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND hwnd,UINT message, WPARAM wParam, LPARAM lParam)
{
   static HWND hwndChild [TEILUNGEN] [TEILUNGEN];
   int         cxBlock, cyBlock;
   int         x, y;

   switch (message)
   {
      //-------------------------------------------------------------------//
      // Als erste Maßnahme in der Initialisierung erzeugen wird TEILUNGEN *  //
      // TEILUNGEN unabhängige Fensterchen, wichtig ist, daß die Daten des    //
      // übergeordneten Fensters mit eingetragen werden (hwnd)               //
      //-------------------------------------------------------------------//
      case WM_CREATE :
           for (x = 0; x < TEILUNGEN; x++)
           {
              for (y = 0; y < TEILUNGEN; y++)
              {
                 //-----------------------------------------------------------//
                 // Ein Child-Window ist ein (fast ganz) normales Fenster     //
                 // lediglich das MENU-HANDLE wird anders verwendet, nämlich  //
                 // als Speicherplatz für eine ID-Nummer des Fensters. Die ID //
                 // wird vom Programmierer vergeben und hier aus dem x und y- //
                 // Index abgeleitet. Das Handle der Programm-Instance wird   //
                 // hier direkt aus der Fensterbeschreibung der Client-Area   //
                 // gewonnen, indem auf die Windowsinternen Fensterdaten zu-  //
                 // gegriffen wird                                            //
                 //-----------------------------------------------------------//
                 hwndChild [x][y] = CreateWindow (szChildClassName, NULL,
                                    WS_CHILDWINDOW | WS_VISIBLE,
                                    0, 0, 0, 0, hwnd, (HMENU)((y << 8) | x),
                                    (HINSTANCE)GetWindowLong (hwnd, GWL_HINSTANCE),
                                    NULL);
              }
           }
           return (0);

      //-------------------------------------------------------------------//
      // Größe des Fensters wird verändert. Diese Message kommt zu Beginn     //
      // des Programms einmal ganz automatisch, da natürlich das Erzeugen des //
      // des Fensters eine Größenveränderung ist                             //
      //-------------------------------------------------------------------//
      case WM_SIZE :
           //-------------------------------------------------------------//
           // Größe jedes Childfensters ermitteln                         //
           //-------------------------------------------------------------//
           cxBlock = LOWORD (lParam) / TEILUNGEN;
           cyBlock = HIWORD (lParam) / TEILUNGEN;

           //-------------------------------------------------------------//
           // Jedes Childwindow an den richtigen Platz rücken             //
           //-------------------------------------------------------------//
           for (x = 0; x < TEILUNGEN; x++)
           {
              for (y = 0; y < TEILUNGEN; y++)
              {
                 MoveWindow (hwndChild [x][y], x * cxBlock, y * cyBlock,
                             cxBlock, cyBlock, TRUE);
              }
           }
           return (0);

      //-------------------------------------------------------------------//
      // Warnsignal, wenn die linke Maustaste über der Client-Area abgefangen //
      // wird                                                                //
      //-------------------------------------------------------------------//
      case WM_LBUTTONDOWN :
           MessageBeep (0);
           return (0);
```

```
      //------------------------------------------------------------------------//
      // Programm beenden                                                       //
      //------------------------------------------------------------------------//
      case WM_DESTROY :
           PostQuitMessage (0);
           return (0);
   }
   return (DefWindowProc (hwnd, message, wParam, lParam));
}

//----------------------------------------------------------------------------//
// Messagehandler der Child-Fenster                                           //
//----------------------------------------------------------------------------//
LRESULT CALLBACK ChildWndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   HDC         hdc;
   PAINTSTRUCT ps;
   RECT        rect;
   static int  what;

   switch (message)
   {
      //------------------------------------------------------------------------//
      // Jedes Childfenster bekommt für sich jeweils eine WM_CREATE-Message   //
      // Daher können wir diese benutzen, um die Werte (an/aus) zu initiali-  //
      // sieren                                                                 //
      //------------------------------------------------------------------------//
      case WM_CREATE :
           SetWindowWord (hwnd, 0, 0);
           what = 1;
           return (0);

      //------------------------------------------------------------------------//
      // Wenn über einem Childwindow die linke Maustaste betätigt wird, dann  //
      // erhält dieses Fenster-Handle (hwnd) eine WM_LBUTTONDOWN Message      //
      //------------------------------------------------------------------------//
      case WM_LBUTTONDOWN :
           //--------------------------------------------------------------------//
           // Zustand (an/aus) umschalten                                        //
           //--------------------------------------------------------------------//
           if (GetWindowWord (hwnd, 0) == 0)
           {
                if (what == 1)
              {
                   SetWindowWord (hwnd, 0, (WORD)1);
                 what = 2;
              }
              else
              {
                 SetWindowWord (hwnd, 0, (WORD)2);
                 what = 1;
              }

           }

           //--------------------------------------------------------------------//
           // Jetzt invalidieren wir genau dieses Fenster, so daß auch nur    //
           // das Childfenster neu gezeichnet werden muß (erheblich schneller)//
           //--------------------------------------------------------------------//
           InvalidateRect (hwnd, NULL, FALSE);
           return (0);

      //------------------------------------------------------------------------//
      // Zeichnen eines Childfensters                                           //
      //------------------------------------------------------------------------//
      case WM_PAINT :
           hdc = BeginPaint (hwnd, &ps);
           //--------------------------------------------------------------------//
           // hwnd ist jetzt das Handle des CHILD-Fensters, eine Umrechnung   //
           // über Indices ist also nicht mehr nötig !!                          //
           //--------------------------------------------------------------------//
           GetClientRect (hwnd, &rect);

           //--------------------------------------------------------------------//
           // Rechteck um das Childwindow malen                                  //
           //--------------------------------------------------------------------//
           Rectangle (hdc, 0, 0, rect.right, rect.bottom);

           //--------------------------------------------------------------------//
           // ggf. Karo mit Kreuz versehen                                       //
           //--------------------------------------------------------------------//
           if (GetWindowWord (hwnd, 0))
```

```
            {
                 if (GetWindowWord (hwnd, 0) == 1)
               {
                    MoveToEx (hdc, 0,          0, NULL);
                  LineTo   (hdc, rect.right, rect.bottom);
                  MoveToEx (hdc, 0,        rect.bottom, NULL);
                  LineTo   (hdc, rect.right, 0);
               }
               if (GetWindowWord (hwnd, 0) == 2)
               {
                    Ellipse (hdc, 1, 1, rect.right-1, rect.bottom-1);
               }
            }
            EndPaint (hwnd, &ps);
            return (0);
      }
      return (DefWindowProc (hwnd, message, wParam, lParam));
}
```

8.8 Controls1 – Einige Elemente

```
//************************************************************************//
//   P6.cpp - Kontrollelemente-Demo                                       //
//************************************************************************//
#include <windows.h>
#include <stdio.h>

//------------------------------------------------------------------------//
// globale Struktur für Kontrollelemente                                  //
//------------------------------------------------------------------------//
struct { long style; char *text; } button [] =
       {
          {BS_PUSHBUTTON,      "PUSHBUTTON"   },
          {BS_DEFPUSHBUTTON,   "DEFPUSHBUTTON"},
            {BS_CHECKBOX,        "CHECKBOX"     },
            {BS_AUTOCHECKBOX,    "AUTOCHECKBOX" },
            {BS_RADIOBUTTON,     "RADIOBUTTON"  },
            {BS_3STATE,          "3STATE"       },
            {BS_AUTO3STATE,      "AUTO3STATE"   },
            {BS_GROUPBOX,        "GROUPBOX"     },
            {BS_AUTORADIOBUTTON, "AUTORADIO"    },
            {BS_OWNERDRAW,       "OWNERDRAW"    }
       };
#define NUM (sizeof button / sizeof button [0])

//------------------------------------------------------------------------//
// Prototyp der Handlerfunction, die die Messages der Client-Area verarbeitet //
//------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

//------------------------------------------------------------------------//
// Alle Windowsprogramme beginnen mit "WinMain" statt mit "main"          //
//------------------------------------------------------------------------//
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine,
                    int nCmdShow)
{
   //---------------------------------------------------------------------//
   // Hier wieder die vier unverzichtbaren Variablen...                   //
   //---------------------------------------------------------------------//
   static char szWindowclassname [] = "P6";
   HWND        hWindow;
   MSG         aMessage;
   WNDCLASS    aWindowclass;

   if (!hPrevInstance)
   {
      aWindowclass.style         = CS_HREDRAW | CS_VREDRAW;
      aWindowclass.lpfnWndProc   = WndProc;
      aWindowclass.cbClsExtra    = 0;
      aWindowclass.cbWndExtra    = 0;
      aWindowclass.hInstance     = hInstance;
      aWindowclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
      aWindowclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
      aWindowclass.hbrBackground = (HBRUSH)GetStockObject (WHITE_BRUSH);
      aWindowclass.lpszMenuName  = NULL;
      aWindowclass.lpszClassName = szWindowclassname;

      RegisterClass (&aWindowclass);
   }
   //---------------------------------------------------------------------//
   // Fenster erzeugen und anzeigen                                       //
   //---------------------------------------------------------------------//
   hWindow = CreateWindow (szWindowclassname, "Kontrollelemente",
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL);
   ShowWindow   (hWindow, nCmdShow);
   UpdateWindow (hWindow);

   //---------------------------------------------------------------------//
   // Verarbeiten aller Botschaften an unser Programm                     //
   //---------------------------------------------------------------------//
   while (GetMessage (&aMessage, NULL, 0, 0))
   {
      TranslateMessage (&aMessage);
      DispatchMessage  (&aMessage);
   }
   //---------------------------------------------------------------------//
```

```
   // Windowskernel mitteilen, daß die WM_QUIT-Message verarbeitet wurde      //
   //-------------------------------------------------------------------------//
   return (aMessage.wParam);
}

//-------------------------------------------------------------------------//
// Messagehandler der Client-Area                                          //
//-------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   static char  szTop []    = "Botschaft     wParam         lParam";
   static char  szUnd []    = "_________ _____________  ____________";
   static char  szFormat [] = "%-16s%6X%8X-%04X";
   static char  szBuffer [50];
   static HWND  hwndButton [NUM];
   static RECT  rect;
   static int   cxChar, cyChar;
   HDC          hdc;
   PAINTSTRUCT  ps;
   int          i;
   TEXTMETRIC   tm;

   switch (message)
   {
      //-------------------------------------------------------------------//
      // Initialisierung des Fensters                                      //
      //-------------------------------------------------------------------//
      case WM_CREATE :
           hdc = GetDC (hwnd);
           //--------------------------------------------------------------//
           // durchschnittliche Zeichengrößen ermitteln                    //
           //--------------------------------------------------------------//
           SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT));
           GetTextMetrics (hdc, &tm);
           cxChar = tm.tmAveCharWidth;
           cyChar = tm.tmHeight + tm.tmExternalLeading;
           ReleaseDC (hwnd, hdc);

           //--------------------------------------------------------------//
           // Dialogelemente (hier Buttons) erzeugen. Auch dies sind nur   //
           // Child-Fenster, allerdings mit jeweils genau festgelegten,    //
           // speziellen Eigenschaften                                     //
           //--------------------------------------------------------------//
           for (i = 0; i < NUM; i++)
           {
              hwndButton [i] = CreateWindow ("button", button[i].text,
                               WS_CHILD | WS_VISIBLE | button[i].style,
                               cxChar, cyChar * (1 + 2 * i),
                               20 * cxChar, 7 * cyChar / 4,
                               hwnd, (HMENU)i,
                               (HINSTANCE)((LPCREATESTRUCT) lParam) -> hInstance,
                               NULL);
           }
           return (0);

      //-------------------------------------------------------------------//
      // Größe des Fensters wird verändert                                 //
      //-------------------------------------------------------------------//
      case WM_SIZE :
           rect.left   = 24 * cxChar;
           rect.top    =  3 * cyChar;
           rect.right  = LOWORD (lParam);
           rect.bottom = HIWORD (lParam);
           return (0);

      //-------------------------------------------------------------------//
      // Neuzeichnen des Fensters                                          //
      //-------------------------------------------------------------------//
      case WM_PAINT :
           InvalidateRect (hwnd, &rect, TRUE);
           hdc = BeginPaint (hwnd, &ps);
           SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT));
           SetBkMode (hdc, TRANSPARENT);
           TextOut (hdc, 24 * cxChar, 2 * cyChar, szTop, sizeof szTop - 1);
           TextOut (hdc, 24 * cxChar, 2 * cyChar, szUnd, sizeof szUnd - 1);
           EndPaint (hwnd, &ps);
           return (0);

      //-------------------------------------------------------------------//
      // Wird ein Dialogelement betätigt, so erhält man eine WM_COMMNAD-   //
      // Message. Man muß dann normalerweise erst feststellen, welches Dialog-//
      // element betätigt und was damit gemacht wurde                      //
      // Die DRAWITEM-Message kommt, wenn ein visiueller Aspekt des Dialog-   //
```

```
      // Fenster geändert wurde. Diese Message ist insbesondere Dann wichtig, //
      // wenn man selbstgestaltete Dialogelemente einsetzt                     //
      //-------------------------------------------------------------------------//
      case WM_COMMAND :
      case WM_DRAWITEM:
           ScrollWindow (hwnd, 0, -cyChar, &rect, &rect);
           hdc = GetDC (hwnd);
           SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT));
           sprintf (szBuffer, szFormat, (LPSTR)
                    (message == WM_COMMAND? "WM_COMMAND" : "WM_DRAWITEM"),
                    wParam, HIWORD (lParam), LOWORD (lParam));
           TextOut (hdc, 24 * cxChar, cyChar * (rect.bottom / cyChar - 1),
                    szBuffer, strlen (szBuffer));
           ReleaseDC (hwnd, hdc);
           ValidateRect (hwnd, NULL);
           return (0);

      //-------------------------------------------------------------------------//
      // Beenden des Programms                                                   //
      //-------------------------------------------------------------------------//
      case WM_DESTROY :
           PostQuitMessage (0);
           return (0);
   }
   return DefWindowProc (hwnd, message, wParam, lParam);
}
```

8.9 Timer 1

```
//*************************************************************************//
//   P6b.cpp - Timer-Demo 1                                                //
//*************************************************************************//
#include <windows.h>

#define ID_TIMER    1

//-------------------------------------------------------------------------//
// Prototyp der Handlerfunction, die die Messages der Client-Area verarbeitet //
//-------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

//-------------------------------------------------------------------------//
// Alle Windowsprogramme beginnen mit "WinMain" statt mit "main"           //
//-------------------------------------------------------------------------//
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine,
                    int nCmdShow)
{
   static char szWindowclassname[] = "P6b" ;
   HWND        hWindow;
   MSG         aMessage;
   WNDCLASS    aWndclass;
   int         nAntwort;

   if (!hPrevInstance)
   {
      aWndclass.style         = CS_HREDRAW | CS_VREDRAW;
      aWndclass.lpfnWndProc   = WndProc;
      aWndclass.cbClsExtra    = 0;
      aWndclass.cbWndExtra    = 0;
      aWndclass.hInstance     = hInstance;
      aWndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
      aWndclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
      aWndclass.hbrBackground = (HBRUSH)GetStockObject (WHITE_BRUSH);
      aWndclass.lpszMenuName  = NULL;
      aWndclass.lpszClassName = szWindowclassname;

      RegisterClass (&aWndclass);
   }

   hWindow = CreateWindow (szWindowclassname, "Timer-Demo 1",
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL);

   //-----------------------------------------------------------------------//
   // mit der Funktion SetTimer kann man bei Windows den Wunsch anmelden, daß //
   // Windows in bestimmten Zeitintervallen eine genau definierte Message     //
   // schicken soll (ggf. unabhängig davon, ob das Programm gerade den Focus  //
   // hat oder nicht. SetTimer benötigt dazu (als Parameter) die folgenden    //
   // Angaben:                                                                //
   // 1. Wohin sollen die Messages geschickt werden (Handle des Fensters)     //
   // 2. Welche Nummer soll der Timer als ID-haben, denn es ist auch möglich  //
   //    einem Fenster mehrere Timer mit unterschiedlichen Zeitabständen zu-  //
   //    zuweisen                                                             //
   // 3. In welchen Zeitabständen soll der Timer kommen (in Millisekunden)    //
   // 4. normalerweise wird als Adresse NULL angegeben, die Message kommt wie //
   //    gehabt über die Messagequeue (also nicht, wenn das Programm den      //
   //    Focus nicht hat), für andere Maßnahmen kann hier die Adresse einer   //
   //    Funktion angegeben werden, die dann aufgerufen wird.                 //
   //-----------------------------------------------------------------------//
   while (!SetTimer (hWindow, ID_TIMER, 500, NULL))
   {
      //--------------------------------------------------------------------//
      // MessageBox ist eine sehr einfache und nützliche Funktion, um dem An- //
      // wender Hinweise bzw. Fehlermeldungen zukommen zu lassen....        //
      //--------------------------------------------------------------------//
      nAntwort = MessageBox (hWindow, "Zuviele Timer!", szWindowclassname,
                             MB_ICONEXCLAMATION | MB_RETRYCANCEL);
      if (IDCANCEL == nAntwort) return FALSE;
   }

   ShowWindow (hWindow, nCmdShow);
   UpdateWindow (hWindow);

   while (GetMessage (&aMessage, NULL, 0, 0))
   {
```

```
      TranslateMessage (&aMessage);
      DispatchMessage (&aMessage);
   }
   return (aMessage.wParam);
}

//--------------------------------------------------------------------------//
// Handler-Function der Client-Area                                         //
//--------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   static BOOL nColChange = FALSE;
   HBRUSH      hBrush ;
   HDC         hdc ;
   PAINTSTRUCT ps ;
   RECT        rc ;

   switch (message)
   {

      //--------------------------------------------------------------------//
      // diese Message kommt jetzt so häufig wie angegeben. Da wir nur einen  //
      // Timer benutzen, entfällt die Notwendigkeit, die Timer anhand ihrer   //
      // ID (wParam) zu unterscheiden.                                        //
      //--------------------------------------------------------------------//
      case WM_TIMER :
           //----------------------------------------------------------------//
           // Bei jedem Wechsel piepsen, Farbwert umschalten und - damit in   //
           // neuer Farbe gemalt wird - invalidieren !                       //
           //----------------------------------------------------------------//
           // MessageBeep (0);
           MessageBeep (MB_ICONASTERISK);     // Kritischer Abbruch
           nColChange = !nColChange;
           InvalidateRect (hwnd, NULL, FALSE);
           return (0);

      //--------------------------------------------------------------------//
      // da wir im Timer die Clientarea invalidieren schickt uns Windows so-  //
      // fort die Aufforderung alles neu zu zeichnen - in Form einer WM_PAINT //
      // Message                                                            //
      //--------------------------------------------------------------------//
      case WM_PAINT :
           hdc = BeginPaint (hwnd, &ps);
           //----------------------------------------------------------------//
           // Größe der Client-Area feststellen, damit wir auch ein schönes   //
           // Rechteck malen können...                                       //
           //----------------------------------------------------------------//
           GetClientRect (hwnd, &rc);

             //--------------------------------------------------------------//
           // Hier erzeugen wir einen farbigen Stift, um die Clientarea aus-  //
           // malen zu können                                                //
           //----------------------------------------------------------------//
           hBrush = CreateSolidBrush (nColChange ? RGB(255,0,0) : RGB(0,0,255));
           FillRect (hdc, &rc, hBrush);
           EndPaint (hwnd, &ps);

           //----------------------------------------------------------------//
           // GANZ WICHTIG! da wir ein neues Resource-Objekt (hier einen      //
           // Brush) erzeugt haben, müssen wir dafür sorgen, daß er auch wie- //
           // der freigegeben wird (anders als StockObjekte)                 //
           //----------------------------------------------------------------//
           DeleteObject (hBrush);
           return (0);

      //--------------------------------------------------------------------//
      // Wie immer bei Programmende....                                     //
      //--------------------------------------------------------------------//
      case WM_DESTROY :
           //----------------------------------------------------------------//
           // GANZ WICHTIG! da wir ein Resource-Objekt (hier einen Timer) für //
           // unser Programm exklusiv erhalten haben und Windows diesen Timer //
           // nur für uns vorhält und verwaltet, müssen wir mitteilen, daß    //
           // wir keine Timerbotschaften mehr benötigen. Und der Timer (es    //
           // gibt nur eine begrenzte Anzahl insgesamt) anderweitig verwendet //
           // werden kann                                                    //
           //----------------------------------------------------------------//
           KillTimer (hwnd, ID_TIMER);
           PostQuitMessage (0);
           return (0);
     }
   return DefWindowProc (hwnd, message, wParam, lParam);
   }
```

8.10 Timer 2 – Callbackfunktion

```
//***************************************************************************//
//   P6c.cpp - Timer-Demo 2                                                  //
//***************************************************************************//
#include <windows.h>

#define ID_TIMER    1

//---------------------------------------------------------------------------//
// Prototyp der Handlerfunction, die die Messages der Client-Area verarbeitet //
//---------------------------------------------------------------------------//
LRESULT CALLBACK WndProc   (HWND, UINT, WPARAM, LPARAM);

//---------------------------------------------------------------------------//
// Prototyp der Callback-Function des Timers                                 //
//---------------------------------------------------------------------------//
VOID CALLBACK TimerProc (HWND, UINT, UINT, DWORD);

//---------------------------------------------------------------------------//
// Alle Windowsprogramme beginnen mit "WinMain" statt mit "main"             //
//---------------------------------------------------------------------------//
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine,
                    int nCmdShow)
{
   static char szWindowclassname[] = "P6c";
   HWND        hWindow;
   MSG         aMessage;
   WNDCLASS    aWndclass;
   int         nAntwort;

   //---------------------------------------------------------------------------//
   // Hier speichern wir die Adresse der Timerfunction                          //
   //---------------------------------------------------------------------------//
   FARPROC     lpfnTimerProc;

   if (!hPrevInstance)
   {
      aWndclass.style         = CS_HREDRAW | CS_VREDRAW;
      aWndclass.lpfnWndProc   = WndProc;
      aWndclass.cbClsExtra    = 0;
      aWndclass.cbWndExtra    = 0;
      aWndclass.hInstance     = hInstance;
      aWndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
      aWndclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
      aWndclass.hbrBackground = (HBRUSH)GetStockObject (WHITE_BRUSH);
      aWndclass.lpszMenuName  = NULL;
      aWndclass.lpszClassName = szWindowclassname;

      RegisterClass (&aWndclass);
   }

   hWindow = CreateWindow (szWindowclassname, "Timer-Demo 2",
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL);

   //---------------------------------------------------------------------------//
   // mit der Funktion SetTimer kann man bei Windows den Wunsch anmelden, daß //
   // Windows in bestimmten Zeitintervallen eine genau definierte Message     //
   // schicken soll (ggf. unabhängig davon, ob das Programm gerade den Focus  //
   // hat oder nicht. SetTimer benötigt dazu (als Parameter) die folgenden    //
   // Angaben:                                                                //
   // 1. Wohin sollen die Messages geschickt werden (Handle des Fensters)     //
   // 2. Welche Nummer soll der Timer als ID-haben, denn es ist auch möglich  //
   //    einem Fenster mehrere Timer mit unterschiedlichen Zeitabständen zu-  //
   //    zuweisen                                                             //
   // 3. In welchen Zeitabständen soll der Timer kommen (in Millisekunden)    //
   // 4. Die Adresse der Function TimerProc wird in eine Instanz-umgewandelt, //
   //    damit Windows sich merken kann, wie oft diese Function in Gebrauch   //
   //    ist (wichtig, falls die Function aus einer DLL kommt, da es dann     //
   //    von der Anzahl der Instanzen abhängt, ob die DLL aus dem Speicher    //
   //    entfernt werden kann                                                 //
   //---------------------------------------------------------------------------//
   lpfnTimerProc = MakeProcInstance ((FARPROC)TimerProc, hInstance);

   while (!SetTimer (hWindow, ID_TIMER, 500, (TIMERPROC)lpfnTimerProc))
   {
      //---------------------------------------------------------------------------//
      // MessageBox ist eine sehr einfache und nützliche Funktion, um dem An- //
```

```
      // wender Hinweise bzw. Fehlermeldungen zukommen zu lassen....          //
      //-------------------------------------------------------------------//
      nAntwort = MessageBox (hWindow, "Zuviele Timer!", szWindowclassname,
                             MB_ICONEXCLAMATION | MB_RETRYCANCEL);
      if (IDCANCEL == nAntwort) return FALSE;
   }

   ShowWindow (hWindow, nCmdShow);
   UpdateWindow (hWindow);

   while (GetMessage (&aMessage, NULL, 0, 0))
   {
      TranslateMessage (&aMessage);
      DispatchMessage (&aMessage);
   }
   return (aMessage.wParam);
}


//-------------------------------------------------------------------------//
// Handler-Function der Client-Area                                        //
//-------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   switch (message)
   {
      //-------------------------------------------------------------------//
      // Wie immer bei Programmende....                                    //
      //-------------------------------------------------------------------//
      case WM_DESTROY :
           //--------------------------------------------------------------//
           // GANZ WICHTIG! da wir ein Resource-Objekt (hier einen Timer) für //
           // unser Programm exklusiv erhalten haben und Windows diesen Timer //
           // nur für uns vorhält und verwaltet, müssen wir mitteilen, daß    //
           // wir keine Timerbotschaften mehr benötigen. Und der Timer (es    //
           // gibt nur eine begrenzte Anzahl insgesamt) anderweitig verwendet //
           // werden kann                                                     //
           //--------------------------------------------------------------//
           KillTimer (hwnd, ID_TIMER);
           PostQuitMessage (0);
           return (0);
   }
   return DefWindowProc (hwnd, message, wParam, lParam);
}

//-------------------------------------------------------------------------//
// Handler-Function des Timers                                             //
//-------------------------------------------------------------------------//
#pragma argsused
VOID CALLBACK TimerProc (HWND hwnd, UINT message, UINT wParam,
                                  DWORD lParam)
{
   static BOOL nColChange = FALSE;
   HBRUSH      hBrush;
   HDC         hdc;
   RECT        rc;

   //-------------------------------------------------------------------//
   // da diese Funktion automatisch bei einer Timerbotschaft aufgerufen wird, //
   // ist hier nur ggf. eine Fallunterscheidung der Timer nötig (Falls alle   //
   // unterschiedlichen Timer über die gleiche TimerProc laufen)              //
   //-------------------------------------------------------------------//
   MessageBeep (0);
   nColChange = !nColChange;
   GetClientRect (hwnd, &rc);
   hdc = GetDC (hwnd);

   //-------------------------------------------------------------------//
   // Hier erzeugen wir einen farbigen Stift, um die Clientarea auszumalen    //
   //-------------------------------------------------------------------//
   hBrush = CreateSolidBrush (nColChange ? RGB(255,0,0) : RGB(0,0,255));
   FillRect (hdc, &rc, hBrush);
   ReleaseDC (hwnd, hdc);

   //-------------------------------------------------------------------//
   // GANZ WICHTIG! da wir ein neues Resource-Objekt (hier einen Brush) er-   //
   // zeugt haben, müssen wir dafür sorgen, daß er auch wieder freigegeben    //
   // wird (anders als StockObjekte)                                          //
   //-------------------------------------------------------------------//
   DeleteObject (hBrush);
}
```

8.11 Timer 3 – Digitaluhr

```
//*************************************************************************//
//   P6d.cpp - Digitaluhr                                                  //
//*************************************************************************//
#include <windows.h>
#include <time.h>

#define ID_TIMER    1

//-------------------------------------------------------------------------//
// Makros                                                                  //
//-------------------------------------------------------------------------//
#define YEAR  (datetime->tm_year % 100)
#define MONTH (datetime->tm_mon  + 1)
#define MDAY  (datetime->tm_mday)
#define WDAY  (datetime->tm_wday)
#define HOUR  (datetime->tm_hour)
#define MIN   (datetime->tm_min)
#define SEC   (datetime->tm_sec)

//-------------------------------------------------------------------------//
// Prototyp der Handlerfunction, die die Messages der Client-Area verarbeitet //
//-------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

//-------------------------------------------------------------------------//
// Funktionsprototypen                                                     //
//-------------------------------------------------------------------------//
void SizeTheWindow    (int&, int&, int&, int&);
void SetInternational (void);
void WndPaint (HWND hwnd, HDC hdc);

//-------------------------------------------------------------------------//
// globale Variablen                                                       //
//-------------------------------------------------------------------------//
char  sDate [2], sTime [2], sAMPM [2][5];
int   iDate, iTime;

//-------------------------------------------------------------------------//
// Alle Windowsprogramme beginnen mit "WinMain" statt mit "main"           //
//-------------------------------------------------------------------------//
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine,
                    int nCmdShow)
{
   static char szWindowclassname[] = "P6d";
   HWND        hWindow;
   MSG         aMessage;
   int         xStart, yStart, xClient, yClient;
   WNDCLASS    aWndclass;

   if (!hPrevInstance)
   {
      aWndclass.style         = CS_HREDRAW | CS_VREDRAW;
      aWndclass.lpfnWndProc   = WndProc;
      aWndclass.cbClsExtra    = 0;
      aWndclass.cbWndExtra    = 0;
      aWndclass.hInstance     = hInstance;
      aWndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
      aWndclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
      aWndclass.hbrBackground = (HBRUSH)GetStockObject (WHITE_BRUSH);
      aWndclass.lpszMenuName  = NULL;
      aWndclass.lpszClassName = szWindowclassname;

      RegisterClass (&aWndclass);
   }

   //----------------------------------------------------------------------//
   // Fenster auf die benötigte Größe bringen                              //
   //----------------------------------------------------------------------//
   SizeTheWindow (xStart, yStart, xClient, yClient);

   hWindow = CreateWindow (szWindowclassname, "Timer-Demo 2",
                           WS_POPUP | WS_DLGFRAME | WS_SYSMENU,
                           xStart-40,  yStart-40,
                           xClient, yClient,
                           NULL, NULL, hInstance, NULL);

   //----------------------------------------------------------------------//
   // mit der Funktion SetTimer kann man bei Windows den Wunsch anmelden, daß //
   // Windows in bestimmten Zeitintervallen eine genau definierte Message     //
```

```
   // schicken soll                                                          //
   //-------------------------------------------------------------------------//
   if (!SetTimer (hWindow, ID_TIMER, 1000, NULL))
   {
      //-------------------------------------------------------------------//
      // MessageBox ist eine sehr einfache und nützliche Funktion, um dem An- //
      // wender Hinweise bzw. Fehlermeldungen zukommen zu lassen....          //
      //-------------------------------------------------------------------//
      MessageBox (hWindow, "Zuviele Timer!", szWindowclassname,
                           MB_ICONEXCLAMATION | MB_OK);
      return FALSE;
   }

   ShowWindow   (hWindow, SW_SHOWNOACTIVATE);
   UpdateWindow (hWindow);

   while (GetMessage (&aMessage, NULL, 0, 0))
   {
      TranslateMessage (&aMessage);
      DispatchMessage (&aMessage);
   }
   return (aMessage.wParam);
}


//-----------------------------------------------------------------------------//
// Funktion zum Ändern der Fenstergröße                                        //
//-----------------------------------------------------------------------------//
void SizeTheWindow (int& pxStart,  int& pyStart, int& pxClient, int& pyClient)
{
   HDC        hdc;
   TEXTMETRIC tm;

   //-------------------------------------------------------------------//
   // mit CREATEIC können wir den InformationContext abrufen, d.h. z.B. Infos //
   // über den Bildschirmtreiber etc. Hier erfolgt der Aufruf um die ent-     //
   // sprechenden Systemvariablen zu aktualisieren                        //
   //-------------------------------------------------------------------//
   hdc = CreateIC ("DISPLAY", NULL, NULL, NULL);
   GetTextMetrics (hdc, &tm);
   DeleteDC (hdc);

   pxClient = 2 * GetSystemMetrics (SM_CXDLGFRAME) + 16 * tm.tmAveCharWidth;
   pxStart  =     GetSystemMetrics (SM_CXSCREEN)   - pxClient;
   pyClient = 2 * GetSystemMetrics (SM_CYDLGFRAME) + 2 * tm.tmHeight;
   pyStart  =     GetSystemMetrics (SM_CYSCREEN)   - pyClient;
}

//-----------------------------------------------------------------------------//
// Funktion zum Ändern der Darstellung                                         //
//-----------------------------------------------------------------------------//
void SetInternational (void)
{
   static char cName [] = "intl";

   //-------------------------------------------------------------------//
   // GetProfileInt liest einen Integerwert aus der WIN.INI, hier die beiden  //
   // Einträge "iDate=" und "iTime=" aus der Section "intl". Werden die Ein-  //
   // träge nicht gefunden, so wird 0 (3. Parameter) als Voreinstellung zu-   //
   // rückgegeben.-                                                           //
   // Für andere .INI-Dateien verwendet man die Function GetPrivateProfileInt //
   // die als 4. Parameter den Filenamen der INI-Datei enthält                //
   //-------------------------------------------------------------------//
   iDate = GetProfileInt (cName, "iDate", 0);
   iTime = GetProfileInt (cName, "iTime", 0);

   //-------------------------------------------------------------------//
   // GetProfileInt liest einen String aus der WIN.INI, hier u.a. die beiden  //
   // Einträge "sDate=" und "sTime=" aus der Section "intl". Werden die Ein-  //
   // träge nicht gefunden, so wird der angegebene String (3. Parameter) auf  //
   // die angeführte Stringvariable kopiert (4. Parameter) der 5. Parameter   //
   // enthält die Größe (Buffersize) der aufnehmenden Variablen               //
   // Für andere INI-Dateien verwendet man GetPrivateProfileString, welche    //
   // als 6. Parameter den Filenamen der INI-Datei enthält                    //
   //-------------------------------------------------------------------//
   GetProfileString (cName, "sDate",  "/", sDate,     2);
   GetProfileString (cName, "sTime",  ":", sTime,     2);
   GetProfileString (cName, "s1159", "AM", sAMPM [0], 5);
   GetProfileString (cName, "s2359", "PM", sAMPM [1], 5);
}

//-----------------------------------------------------------------------------//
// Ausgelagerte Funktion für die WM_PAINT-Message                              //
```

```
//--------------------------------------------------------------------------//
void WndPaint (HWND hwnd, HDC hdc)
{
   static char szWday[] = "So\0Mo\0Di\0Mi\0Do\0Fr\0Sa";
   char        cBuffer[40];
   long        lTime;
   RECT        rect;
   int         nLength;
   struct tm   *datetime;

   //--------------------------------------------------------------------------//
   // Uhrzeit holen                                                            //
   //--------------------------------------------------------------------------//
   time (&lTime);
   datetime = localtime (&lTime);

   //--------------------------------------------------------------------------//
   // Datum   gemäß der in der WIN.INI eingestellten Form aufbereiten          //
   //--------------------------------------------------------------------------//
   nLength = wsprintf (cBuffer, "  %s  %d%s%02d%s%02d  \r\n",
             (LPSTR) szWday + 3 * WDAY,
             iDate == 1 ? MDAY  : iDate == 2 ? YEAR  : MONTH, (LPSTR) sDate,
             iDate == 1 ? MONTH : iDate == 2 ? MONTH : MDAY,  (LPSTR) sDate,
             iDate == 1 ? YEAR  : iDate == 2 ? MDAY  : YEAR);

   //--------------------------------------------------------------------------//
   // Uhrzeit gemäß der in der WIN.INI eingestellten Form (24/12)aufbereiten   //
   //--------------------------------------------------------------------------//
   if (iTime == 1)
   {
      wsprintf (cBuffer + nLength, "  %02d%s%02d%s%02d  ",
                HOUR, (LPSTR) sTime, MIN, (LPSTR) sTime, SEC);
   }
   else
   {
      wsprintf (cBuffer + nLength, "  %d%s%02d%s%02d %s  ",
                (HOUR % 12) ? (HOUR % 12) : 12,
                (LPSTR) sTime, MIN, (LPSTR) sTime, SEC,
                (LPSTR) sAMPM [HOUR / 12]);
   }

   //--------------------------------------------------------------------------//
   // und ausgeben                                                             //
   //--------------------------------------------------------------------------//
   GetClientRect (hwnd, &rect);
   DrawText (hdc, cBuffer, -1, &rect, DT_CENTER | DT_NOCLIP);
}

//--------------------------------------------------------------------------//
// Handler-Function der Client-Area                                         //
//--------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   HDC         hdc;
   PAINTSTRUCT ps;

   switch (message)
   {
      //--------------------------------------------------------------------//
      // Initialisierung beim Öffnen des Fensters - ausgelagert             //
      //--------------------------------------------------------------------//
      case WM_CREATE :
           SetInternational ();
           return (0);

      //--------------------------------------------------------------------//
      // Timer-Message - führt zur Invalidierung, um Zeit neu auszugeben    //
      //--------------------------------------------------------------------//
      case WM_TIMER :
           InvalidateRect (hwnd, NULL, FALSE);
           return (0);

      //--------------------------------------------------------------------//
      // Neuzeichnen bei Invalisierung des Fensters - ausgelagert           //
      //--------------------------------------------------------------------//
      case WM_PAINT :
           hdc = BeginPaint (hwnd, &ps);
           WndPaint (hwnd, hdc);
           EndPaint (hwnd, &ps);
           return (0);

      //--------------------------------------------------------------------//
      // Wir wollen mitbekommen, wenn die WIN.INI geändert wird, könnte uns //
```

```
      // ja hinsichtlich des Zeitformats betreffen. Windows teilt dies allen  //
      // laufenden Programmen durch eine WM_WININICHANGE-Message mit          //
      //-------------------------------------------------------------------------//
      case WM_WININICHANGE :
           SetInternational ();
           InvalidateRect (hwnd, NULL, TRUE);
           return (0);

      //-------------------------------------------------------------------------//
      // beim beenden des Programms schön den Timer wieder entfernen...       //
      //-------------------------------------------------------------------------//
      case WM_DESTROY :
           KillTimer (hwnd, ID_TIMER);
           PostQuitMessage (0);
           return (0);
   }
   return DefWindowProc (hwnd, message, wParam, lParam);
}
```

8.12 Timer 4 – Freier Speicher

```
//************************************************************************//
//   P6e.cpp - Freier Speicherplatz                                       //
//************************************************************************//

#include <windows.h>
#include <stdio.h>

#define ID_TIMER    1

//------------------------------------------------------------------------//
// Prototyp der Handlerfunction, die die Messages der Client-Area verarbeitet //
//------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

//------------------------------------------------------------------------//
// Alle Windowsprogramme beginnen mit "WinMain" statt mit "main"          //
//------------------------------------------------------------------------//
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine,
                    int nCmdShow)
{
   static char szWindowclassname[] = "P6e";
   HWND        hWindow;
   MSG         aMessage;
   WNDCLASS    aWndclass;

   if (hPrevInstance) return (FALSE);

   aWndclass.style         = CS_HREDRAW | CS_VREDRAW;
   aWndclass.lpfnWndProc   = WndProc;
   aWndclass.cbClsExtra    = 0;
   aWndclass.cbWndExtra    = 0;
   aWndclass.hInstance     = hInstance;
   aWndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
   aWndclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
   aWndclass.hbrBackground = (HBRUSH)GetStockObject (WHITE_BRUSH);
   aWndclass.lpszMenuName  = NULL;
   aWndclass.lpszClassName = szWindowclassname;

   RegisterClass (&aWndclass);

   hWindow = CreateWindow (szWindowclassname, "Freier Speicher",
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL);

   //---------------------------------------------------------------------//
   // mit der Funktion SetTimer kann man bei Windows den Wunsch anmelden, daß //
   // Windows in bestimmten Zeitintervallen eine genau definierte Message     //
   // schicken soll                                                           //
   //---------------------------------------------------------------------//
   if (!SetTimer (hWindow, ID_TIMER, 500, NULL))
   {
      //------------------------------------------------------------------//
      // MessageBox ist eine sehr einfache und nützliche Funktion, um dem An- //
      // wender Hinweise bzw. Fehlermeldungen zukommen zu lassen....          //
      //------------------------------------------------------------------//
      MessageBox (hWindow, "Zuviele Timer!", szWindowclassname,
                           MB_ICONEXCLAMATION | MB_OK);
      return FALSE;
   }

   //---------------------------------------------------------------------//
   // Programm als Icon (Taskbar) starten                                 //
   //---------------------------------------------------------------------//
   ShowWindow   (hWindow, SW_MINIMIZE);
   UpdateWindow (hWindow);

   while (GetMessage (&aMessage, NULL, 0, 0))
   {
      TranslateMessage (&aMessage);
      DispatchMessage (&aMessage);
   }
   return (aMessage.wParam);
}


//------------------------------------------------------------------------//
```

```
// Handlerfunction für Client-Area                                          //
//--------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   MEMORYSTATUS ms;
   // static DWORD  dwFreeMem;
   char          cBuffer [20];

   switch (message)
   {
      //--------------------------------------------------------------------//
      // Bei jedem Timeraufruf den Titel des Programms ändern und darin den  //
      // freien Speicherplatz darstellen                                     //
      //--------------------------------------------------------------------//
      case WM_TIMER :
           // dwFreeMem = GetFreeSpace (0);
           GlobalMemoryStatus (&ms);
           sprintf (cBuffer, "%.ld",ms.dwAvailPhys);
           SetWindowText (hwnd, cBuffer);
           return (0);


      //--------------------------------------------------------------------//
      // Ein Vergrößern des Programms nicht zulassen                         //
      //--------------------------------------------------------------------//
      case WM_QUERYOPEN :
           return (0);

      //--------------------------------------------------------------------//
      // Beenden des Programms (Timer freigeben)                             //
      //--------------------------------------------------------------------//
      case WM_DESTROY :
           KillTimer (hwnd, ID_TIMER);
           PostQuitMessage (0);
           return (0);
   }
   return DefWindowProc (hwnd, message, wParam, lParam);
}
```

8.13 Controls 2 – Weitere Elemente

```
//***************************************************************************//
//   P7.cpp - Kontrollelemente-Demo                                          //
//***************************************************************************//
#include <windows.h>
#include <stdlib.h>
#include <string.h>

#define  MAXENV  4096
#define  MYLISTBOX1 1L
#define  MYSTATTXT1 2L

//---------------------------------------------------------------------------//
// Prototyp der Handlerfunction, die die Messages der Client-Area verarbeitet //
//---------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

//---------------------------------------------------------------------------//
// Alle Windowsprogramme beginnen mit "WinMain" statt mit "main"             //
//---------------------------------------------------------------------------//
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine,
                    int nCmdShow)
{
   static char szWindowclassname[] = "P7";
   HWND        hWindow;
   MSG         aMessage;
   WNDCLASS    aWndclass;

   if (!hPrevInstance)
   {
      aWndclass.style         = CS_HREDRAW | CS_VREDRAW;
      aWndclass.lpfnWndProc   = WndProc;
      aWndclass.cbClsExtra    = 0;
      aWndclass.cbWndExtra    = 0;
      aWndclass.hInstance     = hInstance;
      aWndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
      aWndclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
      aWndclass.hbrBackground = (HBRUSH)GetStockObject (WHITE_BRUSH);
      aWndclass.lpszMenuName  = NULL;
      aWndclass.lpszClassName = szWindowclassname;

      RegisterClass (&aWndclass);
   }

   hWindow = CreateWindow (szWindowclassname, "Kontrollelemente-Demo",
                           WS_OVERLAPPEDWINDOW | WS_VSCROLL | WS_HSCROLL,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL);

   ShowWindow (hWindow, nCmdShow);
   UpdateWindow (hWindow);

   while (GetMessage (&aMessage, NULL, 0, 0))
   {
      TranslateMessage (&aMessage);
      DispatchMessage (&aMessage);
   }
   return (aMessage.wParam);
}

//---------------------------------------------------------------------------//
// Messagehandler der Client-Area                                            //
//---------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   static char szBuffer [MAXENV + 1];
   static HWND hwndList, hwndText;
   HDC         hdc;
   TEXTMETRIC  tm;
   WORD        n;

   switch (message)
   {
      //-----------------------------------------------------------------//
      // Initialisierung bei Erzeugung des Fensters                      //
      //-----------------------------------------------------------------//
      case WM_CREATE :
           hdc = GetDC (hwnd);
```

```
      //---------------------------------------------------------------//
      // Eigenschaften (Maße) des Standardfonts ernitteln              //
      //---------------------------------------------------------------//
      GetTextMetrics (hdc, &tm);
      ReleaseDC (hwnd, hdc);

      //---------------------------------------------------------------//
      // Wir erzeugen uns eine Listbox, auch diese ist "nur" ein Fenster //
      // dh. hier ein Child der Client-Area                            //
      // Da ein Child kein Menü haben kann, wird wieder das Menu-Handle  //
      // als ID-Nummer mißbraucht (siehe DEFINE)                       //
      // Alle Listbox-Eigenschaften beginnen mit LBS_ (ListBoxStyle...)  //
      //---------------------------------------------------------------//
      hwndList = CreateWindow ("listbox", NULL,
                 WS_CHILD | WS_VISIBLE | LBS_STANDARD,
                 tm.tmAveCharWidth,
                 tm.tmHeight * 3,
                 tm.tmAveCharWidth * 30 + GetSystemMetrics (SM_CXVSCROLL),
                 tm.tmHeight * 7,
                 hwnd, (HMENU)MYLISTBOX1,
                 (HINSTANCE)GetWindowLong (hwnd, GWL_HINSTANCE), NULL);

      //---------------------------------------------------------------//
      // Erzeugung eines "Statisches Textfeldes". Auf diesem Feld kann   //
      // nur vom Programm aus ausgegeben werden, Eingaben sind nicht     //
      // möglich                                                       //
      // Alle Statictext-Eigenschaften beginnen mit SS_ (StaticString..) //
      //---------------------------------------------------------------//
      hwndText = CreateWindow ("static", NULL,
                 WS_CHILD | WS_VISIBLE | SS_LEFT,
                 tm.tmAveCharWidth,
                 tm.tmHeight,
                 tm.tmAveCharWidth * MAXENV, tm.tmHeight,
                 hwnd, (HMENU)MYSTATTXT1,
                 (HINSTANCE)GetWindowLong (hwnd, GWL_HINSTANCE), NULL);

      //---------------------------------------------------------------//
      // ENVIRON ist eine syystemweites, globales Variablenfeld (char *) //
      //---------------------------------------------------------------//
      for (n = 0; environ[n]; n++)
      {
          //-----------------------------------------------------------//
         // Wenn der String zu lang ist, dann machen wir einfach mit dem //
         // nächsten Eintrag weiter, ansonsten kopieren wir ihn auf den  //
         // Pufferstring und schneiden den zugewiesenen Inhalt der ENV   //
         // variablen in unserer Kopie ab. Den Namen der ENV-Variablen   //
         // fügen wir dann unserer Listbox hinzu                         //
          //-----------------------------------------------------------//
         if (strlen (environ [n]) > MAXENV) continue;
         *strchr (strcpy (szBuffer, environ [n]), '=') = '\0';
         SendMessage (hwndList, LB_ADDSTRING, 0, (LONG) (LPSTR) szBuffer);
      }
      return (0);

//---------------------------------------------------------------------//
// Das Fenster erhält eine WM_SETFOCUS-Message, wenn es vom Anwender    //
// oder vom System aktiviert wird                                      //
//---------------------------------------------------------------------//
case WM_SETFOCUS :
      SetFocus (hwndList);
      SendMessage (hwndList, LB_SETCURSEL, 2, 0L);
      SendMessage (hwnd, WM_COMMAND, MAKEWPARAM (MYLISTBOX1,LBN_SELCHANGE),
                   (LPARAM)hwndList);
      return (0);

//---------------------------------------------------------------------//
// Ein Programm erhält eine WM_COMMAND-Message, jedesmal, wenn mit      //
// einem der Dialogelemente etwas geschieht. ALLE Dialogelemente eines  //
// werden über diese eine COMMAND-Anweisung behandelt.                 //
//---------------------------------------------------------------------//
case WM_COMMAND :
      //---------------------------------------------------------------//
      // Um auf das Ereignis (Message) zu reagieren, muß man zunächst    //
      // ermitteln, mit welchem Dialogelement das gemeldete Ereignis in  //
      // Verbindung steht. Dazu muß man zunächst die ID-Nummer (siehe    //
      // DEFINE) aus dem wParam herausholen, sie steht im Low-Word       //
      //---------------------------------------------------------------//
      switch (LOWORD(wParam))
      {
         //-----------------------------------------------------------//
         // Wenn man weiß, mit welchem Dialogelement gerade etwas ge-    //
         // schehen ist, muß man ermitteln, ob einen das gemeldete Ereig-//
         // nis überhaupt interessiert. Die Art des Ereignisses, der so- //
```

```
            // genannte NOTIFY-Code, steht im High-Word von wParam          //
            // der lParam dient ggf. zum Transporrt weiterer Informationen  //
            // falls die Art des Ereignisses dies nötig macht.              //
            //--------------------------------------------------------------//
              case MYLISTBOX1 :
                  //-------------------------------------------------------//
                   // Der einzige Fall der uns hier interessiert, ist, daß    //
                  // der Anwender ein neues Element in der Listbox angewählt //
                  // hat. Dies meldet Windows mit dem Notify-Code            //
                  // LB_SELCHANGE (ListBox_SELectionCHANGEd)                 //
                  //-------------------------------------------------------//
                  if (HIWORD (wParam) == LBN_SELCHANGE)
                  {
                     //-----------------------------------------------------//
                     // Da der Anwender eine ENV-Variable ausgewählt hat,    //
                     // wollen wir jetzt den zugehörigen Inhalt im ober      //
                     // erzeugten STATICTEXT ausgeben. Dazu brauchen wir zu  //
                     // zunächst den Namen der in der Listbox ausgewählten   //
                     // Variablen. Dazu gibt es die Möglichkeit, der LISTBOX //
                     // die Message LB_GETTEXT zu senden. Diese Message      //
                     // benötigt aber den Index der zurückzugebenden Text-   //
                     // zeile aus der Box, denn LB_GETTEXT kann unabhängig   //
                     // vom ausgewählten Element aufgerufen werden. Der      //
                     // Index des Textes wird der Box im wParam übergeben.   //
                     // Außerdem erwartet die Listbox den Zeiger einer       //
                     // Stringvariablen, in die der Text kopiert wird (wird  //
                     // im lParam übergeben).                                //
                     // Uns fehlt also noch der Index des ausgewählten LB-   //
                     // elements, welcher aber über die Message LB_GETCURSEL //
                     // (ListBox_GETCURrentSELected) beschafft werden kann   //
                     //-----------------------------------------------------//
                     n = (WORD) SendMessage (hwndList, LB_GETCURSEL, 0, 0L);

                     //-----------------------------------------------------//
                     // Praktischerweise gibt die Message LG_GETTEXT die     //
                     // Länge des kopierten Textes zurück....                //
                     //-----------------------------------------------------//
                     n = (WORD) SendMessage (hwndList, LB_GETTEXT, n,
                                             (LONG)(LPSTR)szBuffer);
                     //-----------------------------------------------------//
                     // and den Namen der ENV-Variablen hängen wir nun den   //
                     // zugehörigen Inhalt an, der über die Standardfunction //
                     // getenv ermittelt werden kann...                      //
                     //-----------------------------------------------------//
                     strcpy (szBuffer + n + 1, getenv (szBuffer));
                     *(szBuffer + n) = '=';

                     //-----------------------------------------------------//
                     // SetWindowText setzt den Beschreibungstext eines Dia- //
                     // logelementes. Da STATIC-Texte nur aus diese Be-      //
                     // schreibung bestehen, haben wir damit unser Ziel be-  //
                     // reits erreicht...                                    //
                     //-----------------------------------------------------//
                     SetWindowText (hwndText, szBuffer);
                  }
                  break;

            //--------------------------------------------------------------//
            // Ein COMMAND zu MYSTATTXT1 kann nicht vorkommen, und dient    //
            // hier nur zur Veranschaulichung, wie mehrere Dialogelemente   //
            // mit einem COMMAND behandelt werden                           //
            //--------------------------------------------------------------//
            case MYSTATTXT1 :
                 break;
         }
         return (0);

      //-----------------------------------------------------------------//
      // Message, wenn das Programm beendet werden soll                  //
      //-----------------------------------------------------------------//
      case WM_DESTROY :
           PostQuitMessage (0);
           return 0;
   }
   return DefWindowProc (hwnd, message, wParam, lParam);
}
```

8.14 Controls 3 – Dateiverwaltung

```
//*************************************************************************//
//   P8.cpp - Dateiverwaltungs-Demo                                        //
//*************************************************************************//
#include <windows.h>
#include <io.h>
#include <string.h>
#include <direct.h>

#define  MAXREAD    8192
#define  MYLISTBOX1 1L
#define  MYSTATTXT1 2L

//-------------------------------------------------------------------------//
// Prototypen der Handlerfunktionen                                        //
//-------------------------------------------------------------------------//
LRESULT CALLBACK WndProc  (HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK ListProc (HWND, UINT, WPARAM, LPARAM);

//-------------------------------------------------------------------------//
// globale Variablen                                                       //
//-------------------------------------------------------------------------//
WNDPROC lpfnOldList;

//-------------------------------------------------------------------------//
// Alle Windowsprogramme beginnen mit "WinMain" statt mit "main"           //
//-------------------------------------------------------------------------//
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine,
                    int nCmdShow)
{
   static char szWindowclassname[] = "P8";
   HWND        hWindow;
   MSG         aMessage;
   WNDCLASS    aWndclass;

   if (!hPrevInstance)
   {
      aWndclass.style         = CS_HREDRAW | CS_VREDRAW;
      aWndclass.lpfnWndProc   = WndProc;
      aWndclass.cbClsExtra    = 0;
      aWndclass.cbWndExtra    = 0;
      aWndclass.hInstance     = hInstance;
      aWndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
      aWndclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
      aWndclass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
      aWndclass.lpszMenuName  = NULL;
      aWndclass.lpszClassName = szWindowclassname;

      RegisterClass (&aWndclass);
   }

   hWindow = CreateWindow (szWindowclassname, "Dateiverwaltung-Demo",
                           WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL);

   ShowWindow (hWindow, nCmdShow);
   UpdateWindow (hWindow);

   while (GetMessage (&aMessage, NULL, 0, 0))
   {
      TranslateMessage (&aMessage);
      DispatchMessage (&aMessage);
   }
   return (aMessage.wParam);
}

//-------------------------------------------------------------------------//
// Messagehandler der Client-Area                                          //
//-------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   static BOOL     bValidFile;
   static char     sReadBuffer [MAXREAD];
   static char     szFile [16];
   static HWND     hwndList, hwndText;
   static OFSTRUCT ofs;
   static RECT     rect;
```

```
char            szBuffer [MAXPATH + 1];
HDC             hdc;
int             iHandle, i;
PAINTSTRUCT     ps;
TEXTMETRIC      tm;

switch (message)
{
   //---------------------------------------------------------------------//
   // Initialisierung bei Erzeugung des Fensters                          //
   //---------------------------------------------------------------------//
   case WM_CREATE :
        hdc = GetDC (hwnd);
        //-----------------------------------------------------------------//
        // Eigenschaften (Maße) des Standardfonts ernitteln                //
        //-----------------------------------------------------------------//
        SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT));
        GetTextMetrics (hdc, &tm);
        ReleaseDC (hwnd, hdc);
        rect.left = 40 * tm.tmAveCharWidth;
        rect.top  =  3 * tm.tmHeight;
        //-----------------------------------------------------------------//
        // Wir erzeugen uns eine Listbox, auch diese ist "nur" ein Fenster //
        // dh. hier ein Child der Client-Area                              //
        //-----------------------------------------------------------------//
        hwndList = CreateWindow ("listbox", NULL,
                   WS_CHILDWINDOW | WS_VISIBLE | LBS_STANDARD,
                   tm.tmAveCharWidth,
                   tm.tmHeight * 3,
                   tm.tmAveCharWidth * 30 + GetSystemMetrics (SM_CXVSCROLL),
                   tm.tmHeight * 10,
                   hwnd, (HMENU)MYLISTBOX1,
                   (HINSTANCE)GetWindowLong (hwnd, GWL_HINSTANCE),
                   NULL);

        //-----------------------------------------------------------------//
        // Erzeugung eines "Statisches Textfeldes". Auf diesem Feld kann   //
        // nur vom Programm aus ausgegeben werden, Eingaben sind nicht     //
        // möglich                                                         //
        //-----------------------------------------------------------------//
        hwndText = CreateWindow ("static", getcwd (szBuffer, MAXPATH),
                   WS_CHILDWINDOW | WS_VISIBLE | SS_LEFT,
                   tm.tmAveCharWidth,
                   tm.tmHeight,
                   tm.tmAveCharWidth * MAXPATH,
                   tm.tmHeight,
                   hwnd, (HMENU)MYSTATTXT1,
                   (HINSTANCE)GetWindowLong (hwnd, GWL_HINSTANCE),
                   NULL);

        //-----------------------------------------------------------------//
        // Als Child des Hauptfensters hängt die Listbox automatisch an    //
        // dessen Handler, dies wollen wir ändern. Da wir den bisherigen   //
        // Handler als Default-Handler angeben wollen, merken wir uns die  //
        // Adresse des Handlers                                            //
        //-----------------------------------------------------------------//
        lpfnOldList = (WNDPROC)GetWindowLong (hwndList, GWL_WNDPROC);

        //-----------------------------------------------------------------//
        // Jetzt tragen wir den neuen Handler ein                          //
        //-----------------------------------------------------------------//
        SetWindowLong (hwndList, GWL_WNDPROC,
                       (LONG) MakeProcInstance ((FARPROC) ListProc,
                       GetWindowWord (hwnd, GWW_HINSTANCE)));

        //-----------------------------------------------------------------//
        // Mit LB_DIR lassen wir Windows die Listbox mit dem Verzeichnis   //
        // füllen                                                          //
        //-----------------------------------------------------------------//
        SendMessage (hwndList, LB_DIR, 0x37, (LONG) (LPSTR) "*.*");
        return (0);

   //---------------------------------------------------------------------//
   // Veränderung der Fenstergröße                                        //
   //---------------------------------------------------------------------//
   case WM_SIZE :
        rect.right  = LOWORD (lParam);
        rect.bottom = HIWORD (lParam);
        return (0);

   //---------------------------------------------------------------------//
   // Das Fenster erhält eine WM_SETFOCUS-Message, wenn es vom Anwender    //
   // oder vom System aktiviert wird                                      //
```

```
//-----------------------------------------------------------------------//
case WM_SETFOCUS :
     SetFocus (hwndList);
     return (0);

//-----------------------------------------------------------------------//
// Ein Programm erhält eine WM_COMMAND-Message, jedesmal, wenn mit      //
// einem der Dialogelemente etwas geschieht. ALLE Dialogelemente eines  //
// werden über diese eine COMMAND-Anweisung behandelt.                  //
//-----------------------------------------------------------------------//
case WM_COMMAND :
       //-------------------------------------------------------------//
     // Um auf das Ereignis (Message) zu reagieren, muß man zunächst    //
     // ermitteln, mit welchem Dialogelement das gemeldete Ereignis in  //
     // Verbindung steht. Dazu muß man zunächst die ID-Nummer (siehe    //
     // DEFINE) aus dem wParam herausholen, sie steht im Low-Word       //
     //-------------------------------------------------------------//
     switch (LOWORD(wParam))
     {
        //----------------------------------------------------------//
        // Wenn man weiß, mit welchem Dialogelement gerade etwas ge-    //
        // schehen ist, muß man ermitteln, ob einen das gemeldete Ereig-//
        // nis überhaupt interessiert. Die Art des Ereignisses, der so- //
        // genannte NOTIFY-Code, steht im High-Word von wParam          //
        // der lParam dient ggf. zum Transporrt weiterer Informationen  //
        // falls die Art des Ereignisses dies nötig macht.              //
        //----------------------------------------------------------//
          case MYLISTBOX1 :
              //-------------------------------------------------//
             // uns interessiert der Doppelclick auf einen Eintrag      //
              //-------------------------------------------------//
             if (HIWORD (wParam) == LBN_DBLCLK)
             {
                 //-------------------------------------------------//
                // Index des doppelgecklickten Eintrags holen           //
                 //-------------------------------------------------//
                i = (WORD)(SendMessage (hwndList, LB_GETCURSEL, 0, 0L));
                if (LB_ERR == i) break;

                 //-------------------------------------------------//
                // Text  des doppelgecklickten Eintrags holen           //
                 //-------------------------------------------------//
                SendMessage (hwndList, LB_GETTEXT, i,
                             (LONG)(char far *)szBuffer);

                 //-------------------------------------------------//
                // Versuch diese Datei zu öffnen                        //
                 //-------------------------------------------------//
                i = OpenFile (szBuffer, &ofs, OF_EXIST | OF_READ);

                 //-------------------------------------------------//
                // Wenn das klappt, ist i ungleich -1                   //
                 //-------------------------------------------------//
                if (-1 != i)
                {
                     //---------------------------------------------//
                   // Den Namen der Datei im Static-Text ausgeben       //
                    //---------------------------------------------//
                   bValidFile = TRUE;
                   strcpy (szFile, szBuffer);
                   getcwd (szBuffer, MAXPATH);
                   if (szBuffer [strlen (szBuffer) - 1] != '\\')
                        strcat (szBuffer, "\\");
                   SetWindowText (hwndText, strcat (szBuffer, szFile));
                }
                 //-------------------------------------------------//
                // Wenn das nicht klappt ist i gleich -1                //
                 //-------------------------------------------------//
                else
                {
                     //---------------------------------------------//
                   // Wir gehen mal davon aus, daß es dann ein Ver-     //
                   // zeichnis sein muß und wechseln selbiges           //
                    //---------------------------------------------//
                   bValidFile = FALSE;
                   szBuffer [strlen (szBuffer) - 1] = '\0';

                     //---------------------------------------------//
                   // Verzeichniswechsel und Verz.-Namen anzeigen       //
                    //---------------------------------------------//
                   chdir (szBuffer + 1);
                   getcwd (szBuffer, MAXPATH);
                   SetWindowText (hwndText, szBuffer);
```

```
                        //-------------------------------------------------//
                       // Inhalt der Listbox zurücksetzen und neu lesen     //
                        //-------------------------------------------------//
                       SendMessage (hwndList, LB_RESETCONTENT, 0, 0L);
                       SendMessage (hwndList, LB_DIR, 0x37,
                                    (LONG) (LPSTR) "*.*");
                    }
                    InvalidateRect (hwnd, NULL, TRUE);
                }
          }
          return 0;

      //-------------------------------------------------------------------//
      // Neu zeichnen                                                      //
      //-------------------------------------------------------------------//
      case WM_PAINT :
          hdc = BeginPaint (hwnd, &ps);
            //-----------------------------------------------------------//
          // Font, Stiftfarbe und Hintergrund gemäß Standardeinstellungen    //
            //-----------------------------------------------------------//
          SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT));
          SetTextColor (hdc, GetSysColor (COLOR_WINDOWTEXT));
          SetBkColor   (hdc, GetSysColor (COLOR_WINDOW));

            //-----------------------------------------------------------//
          // Wenn es sich um eine Datei handelt, ist bValid auf TRUE gesetzt //
            //-----------------------------------------------------------//
          iHandle = OpenFile (szFile, &ofs, OF_REOPEN | OF_READ);
          if ((bValidFile && -1) != iHandle)
          {
             i = _lread (iHandle, sReadBuffer, MAXREAD);
             _lclose (iHandle);
             DrawText (hdc, sReadBuffer, i, &rect,
                       DT_WORDBREAK | DT_EXPANDTABS | DT_NOCLIP | DT_NOPREFIX);
          }
          else
          {
             bValidFile = FALSE;
          }

          EndPaint (hwnd, &ps);
          return (0);

      //-------------------------------------------------------------------//
      // Programm beenden                                                  //
      //-------------------------------------------------------------------//
      case WM_DESTROY :
          PostQuitMessage (0);
          return (0);
   }
   return DefWindowProc (hwnd, message, wParam, lParam);
}

//-------------------------------------------------------------------------//
// Messagehandler der Listbox                                              //
//-------------------------------------------------------------------------//
LRESULT CALLBACK ListProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   if ((message == WM_KEYDOWN) && (wParam == VK_RETURN))
   {
   //16bit: SendMessage(GetParent(hwnd),WM_COMMAND,1,MAKELONG(hwnd,LBN_DBLCLK));
      SendMessage(GetParent(hwnd),WM_COMMAND,MAKELONG(1,LBN_DBLCLK),(long)hwnd);
   }
   return CallWindowProc (lpfnOldList, hwnd, message, wParam, lParam);
}
```

8.15 Controls 4 – Farbmischung

```
//************************************************************************//
//   P9.cpp - Farbmischung                                                //
//************************************************************************//
#include <windows.h>
#include <stdlib.h>

inline int min (int a, int b) { if (a < b) return (a); else return (b); }
inline int max (int a, int b) { if (a > b) return (a); else return (b); }

//------------------------------------------------------------------------//
// Prototypen der Handlerfunktionen                                       //
//------------------------------------------------------------------------//
LRESULT CALLBACK WndProc    (HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK ScrollProc (HWND, UINT, WPARAM, LPARAM);

//------------------------------------------------------------------------//
// globale Variablen                                                      //
//------------------------------------------------------------------------//
WNDPROC lpfnOldScr[3];
HWND    hwndScrol[3], hwndLabel[3], hwndValue[3], hwndRect;
int     color[3], nFocus;

HWND    hWindow;

//------------------------------------------------------------------------//
// Alle Windowsprogramme beginnen mit "WinMain" statt mit "main"          //
//------------------------------------------------------------------------//
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine,
                    int nCmdShow)
{
   static char szWindowclassname[] = "P9";
   static char *szColorLabel[] = { "Rot", "Grün", "Blau" };
   MSG         aMessage;
   WNDCLASS    aWndclass;
   FARPROC     lpfnScrollProc;
   short       n;

   if (hPrevInstance) return FALSE;

   aWndclass.style         = CS_HREDRAW | CS_VREDRAW;
   aWndclass.lpfnWndProc   = WndProc;
   aWndclass.cbClsExtra    = 0;
   aWndclass.cbWndExtra    = 0;
   aWndclass.hInstance     = hInstance;
   aWndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
   aWndclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
   aWndclass.hbrBackground = CreateSolidBrush (0L);
   aWndclass.lpszMenuName  = NULL;
   aWndclass.lpszClassName = szWindowclassname;

   RegisterClass (&aWndclass);

   hWindow = CreateWindow (szWindowclassname, "Farbmischung-Demo",
                           WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL);

   //---------------------------------------------------------------------//
   // Rechteck zum anzeigen der Farbe erzeugen, wir mißbrauchen dafür ein    //
   // leeres STATIC-Control                                               //
   //---------------------------------------------------------------------//
   hwndRect = CreateWindow ("static", NULL,
                            WS_CHILD | WS_VISIBLE | SS_WHITERECT,
                            0, 0, 0, 0,
                            hWindow, (HMENU)9, hInstance, NULL);

   //---------------------------------------------------------------------//
   // Adresse des eigenen Handlers für die drei Scrollbars ermitteln         //
   //---------------------------------------------------------------------//
   lpfnScrollProc = MakeProcInstance ((FARPROC) ScrollProc, hInstance);

   //---------------------------------------------------------------------//
   // die drei Scrollbars mit Überschrift und Nummernanzeige                 //
   //---------------------------------------------------------------------//
   for (n=0; n<3; n++)
   {
      //------------------------------------------------------------------//
```

```
      // Scrollbar                                                          //
      //-------------------------------------------------------------------//
      hwndScrol[n] = CreateWindow ("scrollbar", NULL,
                     WS_CHILD | WS_VISIBLE | WS_TABSTOP | SBS_VERT,
                     0, 0, 0, 0,
                     hWindow, (HMENU)n, hInstance, NULL);

      //-------------------------------------------------------------------//
      // Überschrift                                                       //
      //-------------------------------------------------------------------//
      hwndLabel[n] = CreateWindow ("static", szColorLabel[n],
                     WS_CHILD | WS_VISIBLE | SS_CENTER,
                     0, 0, 0, 0,
                     hWindow, (HMENU)(n+3), hInstance, NULL);

      //-------------------------------------------------------------------//
      // Fläche hinter des Scrollbars (bleibt farblich konstant)           //
      //-------------------------------------------------------------------//
      hwndValue[n] = CreateWindow ("static", "0",
                     WS_CHILD | WS_VISIBLE | SS_CENTER,
                     0, 0, 0, 0,
                     hWindow, (HMENU)(n+6), hInstance, NULL);

      //-------------------------------------------------------------------//
      // Eintragen des eigenen Scrollbarhandlers und sichern der alten     //
      // Scrollbar-Handleradresse                                          //
      //-------------------------------------------------------------------//
      lpfnOldScr[n] = (WNDPROC) GetWindowLong (hwndScrol[n], GWL_WNDPROC);
      SetWindowLong  (hwndScrol[n], GWL_WNDPROC, (LONG) lpfnScrollProc);

      //-------------------------------------------------------------------//
      // Scrollbar-Eingabebereich auf 0-255 setzen und mit 0 initialisieren//
      //-------------------------------------------------------------------//
      SetScrollRange (hwndScrol[n], SB_CTL, 0, 255, FALSE);
      SetScrollPos   (hwndScrol[n], SB_CTL, 0, FALSE);
   }

   ShowWindow (hWindow, nCmdShow);
   UpdateWindow (hWindow);

   while (GetMessage (&aMessage, NULL, 0, 0))
   {
      TranslateMessage (&aMessage);
      DispatchMessage (&aMessage);
   }
   return (aMessage.wParam);
}

//-------------------------------------------------------------------------//
// Handlerfunktion für die Client-Area                                     //
//-------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   static HBRUSH hBrush[3];
   HBRUSH        newBrush;
   HGDIOBJ       hObject;
   char          szbuffer[10];
   HDC           hdc;
   POINT         point;
   int           n, cxClient, cyClient, cyChar;
   TEXTMETRIC    tm;

   switch (message)
   {
      //-------------------------------------------------------------------//
      // Beim Erzeigen des Fensters drei Brushes erzeugen (dynamisch) für die //
      // Grundfarben hinter den Scrollbars                                 //
      //-------------------------------------------------------------------//
      case WM_CREATE :
           hBrush[0] = CreateSolidBrush (RGB (255, 0, 0));
           hBrush[1] = CreateSolidBrush (RGB (0, 255, 0));
           hBrush[2] = CreateSolidBrush (RGB (0, 0, 255));
           return (0);

      //-------------------------------------------------------------------//
      // Bei Veränderung der Fenstergröße                                  //
      //-------------------------------------------------------------------//
      case WM_SIZE :
           //--------------------------------------------------------------//
           // Die neue Fenstergröße ermitteln                              //
           //--------------------------------------------------------------//
           cxClient = LOWORD (lParam);
           cyClient = HIWORD (lParam);
```

```
     //---------------------------------------------------------------//
     // Texteinstellungen für den Standardfont ermitteln              //
     //---------------------------------------------------------------//
     hdc = GetDC (hwnd);
     GetTextMetrics (hdc, &tm);
     cyChar = tm.tmHeight;
     ReleaseDC (hwnd, hdc);

     //---------------------------------------------------------------//
     // Rechteck hinter den Scrollbars neu skalieren                  //
     //---------------------------------------------------------------//
     MoveWindow (hwndRect, 0, 0, cxClient / 2, cyClient, TRUE);
     for (n = 0; n < 3; n++)
     {
        //------------------------------------------------------------//
        // Die drei Scrollbars neu positionieren und skalieren        //
        //------------------------------------------------------------//
        MoveWindow (hwndScrol[n], (2*n+1)*cxClient/14, 2*cyChar,
                    cxClient/14, cyClient-4*cyChar, TRUE);
        MoveWindow (hwndLabel[n], (4*n+1)*cxClient/28, cyChar/2,
                    cxClient/7, cyChar, TRUE);
        MoveWindow (hwndValue[n], (4*n+1)*cxClient/28,cyClient-3*cyChar/2,
                    cxClient/7, cyChar, TRUE);
     }
     SetFocus (hwnd);
     return (0);

//-------------------------------------------------------------------//
// Bei Zuordnung des Eingabefocus                                    //
//-------------------------------------------------------------------//
case WM_SETFOCUS :
     SetFocus (hwndScrol[nFocus]);
     return (0);

//-------------------------------------------------------------------//
// Wenn einer der Scrollbars bedient wird                            //
//-------------------------------------------------------------------//
case WM_VSCROLL :
     //---------------------------------------------------------------//
     // Id-Nummer des entsprechenden Objekts holen (liegt zw. 0 und 2)  //
     //---------------------------------------------------------------//
     n = GetWindowLong ((HWND)lParam, GWL_ID);

     //---------------------------------------------------------------//
     // Wie wurde das Scroll-Ereignis ausgelöst ?                     //
     //---------------------------------------------------------------//
     switch (LOWORD(wParam))
     {
        //------------------------------------------------------------//
        // Seite herunter                                             //
        //------------------------------------------------------------//
        case SB_PAGEDOWN :
             color[n] += 15;
             color[n] = min (255, color[n] + 1);
             break;

        //------------------------------------------------------------//
        // Zeile herunter                                             //
        //------------------------------------------------------------//
        case SB_LINEDOWN :
             color[n] = min (255, color[n] + 1);
             break;

        //------------------------------------------------------------//
        // Seite herauf                                               //
        //------------------------------------------------------------//
        case SB_PAGEUP :
             color[n] -= 15;
             color[n] = max (0, color[n] - 1);
             break;

        //------------------------------------------------------------//
        // Zeile herauf                                               //
        //------------------------------------------------------------//
        case SB_LINEUP :
             color[n] = max (0, color[n] - 1);
             break;

        //------------------------------------------------------------//
        // Ganz nach oben                                             //
        //------------------------------------------------------------//
        case SB_TOP :
```

```
                color[n] = 0;
                break;

           //-------------------------------------------------------------//
           // Ganz nach unten                                             //
           //-------------------------------------------------------------//
           case SB_BOTTOM :
                color[n] = 255;
                break;

           //-------------------------------------------------------------//
           // Schieben                                                    //
           //-------------------------------------------------------------//
           case SB_THUMBTRACK :
                color[n] = HIWORD(wParam);
                break;

           default :
                break;
      }

      //-------------------------------------------------------------------//
      // Scrollbutton dem Wert entsprechend setzen                         //
      //-------------------------------------------------------------------//
      SetScrollPos  (hwndScrol[n], SB_CTL, color[n], TRUE);
      SetWindowText (hwndValue[n], itoa (color[n], szbuffer, 10));

      //-------------------------------------------------------------------//
      // Brush mit den neuen RGB-Werten erzeugen und als Hintergrund-      //
      // farbe des Fensters einsetzen. Anschließend Brush wieder freige-   //
      // ben und die Client-Area invalidieren                              //
      //-------------------------------------------------------------------//
      newBrush = CreateSolidBrush (RGB (color[0], color[1], color[2]));
      hObject=(HGDIOBJ)SetClassLong(hwnd,GCL_HBRBACKGROUND,(long)newBrush);
      DeleteObject (hObject);
      InvalidateRect (hwnd, NULL, TRUE);
      return (0);

 //---------------------------------------------------------------------//
 // Veränderung der Hintergrundfarbe der Scrollbars                     //
 //---------------------------------------------------------------------//
 // Achtung!!! unter 16-Bit-Anwendungen gibt es nicht eine Message pro  //
 // Objekttyp (hier SCROLLBAR), sondern nur eine allgemeine CTLCOLOR    //
 // Message. Der Objekttyp muß dann aus dem lParam ermittelt werden. Die //
 // CTLCOLOR-Message existiert im 32bit-Windows nicht mehr:             //
 //             case WM_CTLCOLOR :                                      //
 //                  if (HIWORD (lParam) == CTLCOLOR_SCROLLBAR)         //
 //---------------------------------------------------------------------//
 case WM_CTLCOLORSCROLLBAR :
      //-------------------------------------------------------------------//
      // Hintergrundfarbe für Textausgabe aus Windowseinstellungen holen //
      // Vordergrundfarbe auch und dazu die ID-Nummer der Scrollbar      //
      //-------------------------------------------------------------------//
      SetBkColor ((HDC)wParam, GetSysColor (COLOR_CAPTIONTEXT));
      SetTextColor ((HDC)wParam, GetSysColor (COLOR_WINDOWFRAME));
      n = GetWindowLong ((HWND)lParam, GWL_ID);

      //-------------------------------------------------------------------//
      // Screenposition der Clientarea ermitteln                           //
      //-------------------------------------------------------------------//
      point.x = 0;
      point.y = 0;
      ClientToScreen (hwnd, &point);

      //-------------------------------------------------------------------//
      // Palette der Scrollbar neu berechnen lassen                        //
      // logische Startposition des Brushes (für gerasterte) setzen      //
      //-------------------------------------------------------------------//
      UnrealizeObject (hBrush[n]);
      SetBrushOrgEx ((HDC)wParam, point.x, point.y, NULL);
      return ((DWORD) hBrush[n]);

 //---------------------------------------------------------------------//
 // Bei Programmende den Startzustand wiederherstellen                  //
 //---------------------------------------------------------------------//
 case WM_DESTROY :
          DeleteObject ((HGDIOBJ)SetClassLong (hwnd, GCL_HBRBACKGROUND,
                        (long)GetStockObject (WHITE_BRUSH)));
          for (n = 0; n < 3; DeleteObject (hBrush [n++]));
          PostQuitMessage (0);
          return (0);
     }
return DefWindowProc (hwnd, message, wParam, lParam);
```

```
     }

//---------------------------------------------------------------------------//
// Handlerfunktion für die Scrollbar                                         //
//---------------------------------------------------------------------------//
LRESULT CALLBACK ScrollProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   //------------------------------------------------------------------------//
   // ID-Nummer des Objektes holen, da dieser Handler für alle drei Scroll-  //
   // bars benutzt wird                                                      //
   //------------------------------------------------------------------------//
   static int n = GetWindowLong (hwnd, GWL_ID);
   static BOOL shiftisdown = FALSE;

   switch (message)
   {
      //---------------------------------------------------------------------//
      // Wenn eine Taste gedrückt wird, dann nachsehen, ob es Tab-Taste ist  //
      //---------------------------------------------------------------------//
      case WM_KEYDOWN :
           if (wParam == VK_TAB)
           {
              if (GetKeyState (VK_SHIFT) < 0) n = (n + 2) % 3;
              else n = (n + 1) % 3;
              SetFocus (hwndScrol[n]);
           }
           if (wParam == VK_SHIFT)
              shiftisdown = TRUE;
           break;

      case WM_KEYUP :
           if (wParam == VK_SHIFT)
              shiftisdown = FALSE;
           if (wParam == VK_UP)
           {
               if (shiftisdown)
                  SendMessage (hWindow, WM_VSCROLL,
                               SB_PAGEUP,
                               (LPARAM)hwndScrol[nFocus]);
             //  else
                //  SendMessage (hWindow, WM_VSCROLL,
                //  SB_LINEUP, );

           }
           return (1);

      case WM_SETFOCUS :
           nFocus = n;
           break;
   }
   return (CallWindowProc (lpfnOldScr[n], hwnd, message, wParam, lParam));
}
```

8.16 Selbstdefinierte Eingabeelemente

```
//************************************************************************//
//   P10.cpp - Selbstdefinierte Eingabeelemente                           //
//************************************************************************//
#include <windows.h>
#include <stdio.h>

#define IDC_SMALLER   1
#define IDC_LARGER    2

//------------------------------------------------------------------------//
// Präprozessor Makros                                                    //
//------------------------------------------------------------------------//
#define BTN_WIDTH     (8 * cxChar)
#define BTN_HEIGHT    (4 * cyChar)

//------------------------------------------------------------------------//
// globale Variablen                                                      //
//------------------------------------------------------------------------//
HINSTANCE hInst;

//------------------------------------------------------------------------//
// Prototypen der Handlerfunktionen                                       //
//------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

//------------------------------------------------------------------------//
// Alle Windowsprogramme beginnen mit "WinMain" statt mit "main"          //
//------------------------------------------------------------------------//
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine,
                    int nCmdShow)
{
   static char szWindowclassname[] = "P10";
   HWND        hWindow;
   MSG         aMessage;
   WNDCLASS    aWndclass;

   hInst = hInstance;

   if (!hPrevInstance)
   {
      aWndclass.style         = CS_HREDRAW | CS_VREDRAW;
      aWndclass.lpfnWndProc   = WndProc;
      aWndclass.cbClsExtra    = 0;
      aWndclass.cbWndExtra    = 0;
      aWndclass.hInstance     = hInstance;
      aWndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
      aWndclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
      aWndclass.hbrBackground = (HBRUSH)GetStockObject (WHITE_BRUSH);
      aWndclass.lpszMenuName  = NULL;
      aWndclass.lpszClassName = szWindowclassname;

      RegisterClass (&aWndclass);
   }

   hWindow = CreateWindow (szWindowclassname, "Eigene Kontrollelemente",
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL);

   ShowWindow (hWindow, nCmdShow);
   UpdateWindow (hWindow);

   while (GetMessage (&aMessage, NULL, 0, 0))
   {
      TranslateMessage (&aMessage);
      DispatchMessage  (&aMessage);
   }
   return (aMessage.wParam);
}

//------------------------------------------------------------------------//
// Funktion zum Zeichnen eines Dreiecks                                   //
//------------------------------------------------------------------------//
void Triangle(HDC hdc, POINT pt[])
{
   SelectObject (hdc, GetStockObject (BLACK_BRUSH));
   Polygon (hdc, pt, 3);
```

```
   SelectObject (hdc, GetStockObject (WHITE_BRUSH));
}

//---------------------------------------------------------------------------//
// Handler der Client-Area                                                   //
//---------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LONG lParam)
{
   static HWND       hwndSmaller, hwndLarger;
   static short      cxClient, cyClient, cxChar, cyChar;
   LPDRAWITEMSTRUCT  lpdis;
   POINT             pt[3];
   RECT              rc;
   int               cx, cy;

   switch (message)
   {
      //---------------------------------------------------------------------//
      // Fensterinitialisierung                                              //
      //---------------------------------------------------------------------//
      case WM_CREATE :
           //----------------------------------------------------------------//
           // Ermitteln der Dialoggröße in BASEUNITS. Die horizontale Base-  //
           // unit entspricht der durchschnittlichen Breite des Systemfonts  //
           // (in Pixeln). Die vertikale Baseunit entspricht der Höhe des    //
           // Systemfonts (in Pixeln).                                       //
           //----------------------------------------------------------------//
           cxChar = LOWORD (GetDialogBaseUnits ());
           cyChar = HIWORD (GetDialogBaseUnits ());

           //----------------------------------------------------------------//
           // Erzeugen der selbstdefinierten Elemente                        //
           // Da wir die Elemente selber zeichnen müssen, erhalten die Dialog-//
           // elemente den Buttonstil BS_OWNERDRAW. Wir erhalten dann eine   //
           // WM_DRAWITEM-Message, wenn ein Dialogelement neu gezeichnet wer- //
           // den muß                                                        //
           //----------------------------------------------------------------//
           hwndSmaller = CreateWindow("button", "",
                         WS_CHILD | WS_VISIBLE | BS_OWNERDRAW,
                         0, 0, BTN_WIDTH, BTN_HEIGHT,
                         hwnd, (HMENU)IDC_SMALLER, hInst, 0);
           hwndLarger  = CreateWindow("button", "",
                         WS_CHILD | WS_VISIBLE | BS_OWNERDRAW,
                         0, 0, BTN_WIDTH, BTN_HEIGHT,
                         hwnd, (HMENU)IDC_LARGER, hInst, 0);
           return (0);

      //---------------------------------------------------------------------//
      // Bei Größenveränderung des Fensters                                  //
      //---------------------------------------------------------------------//
      case WM_SIZE :
           //----------------------------------------------------------------//
           // neue Größe aus den Parametern ermitteln                        //
           //----------------------------------------------------------------//
           cxClient = LOWORD (lParam);
           cyClient = HIWORD (lParam);

           //----------------------------------------------------------------//
           // Selbstdefinierte Elemente wieder in die Mitte rücken           //
           //----------------------------------------------------------------//
           MoveWindow (hwndSmaller, cxClient / 2 - 3 * BTN_WIDTH / 2,
                                    cyClient / 2 -     BTN_HEIGHT / 2,
                                    BTN_WIDTH, BTN_HEIGHT, TRUE);
           MoveWindow (hwndLarger,  cxClient / 2 +     BTN_WIDTH / 2,
                                    cyClient / 2 -     BTN_HEIGHT / 2,
                                    BTN_WIDTH, BTN_HEIGHT, TRUE);
           return (0);

      //---------------------------------------------------------------------//
      // Benutzerbefehle / Eingaben an/in Dialogelementen                    //
      //---------------------------------------------------------------------//
       case WM_COMMAND :
           GetWindowRect (hwnd, &rc);

           //----------------------------------------------------------------//
           // Fenster um 10% vergrößern bzw. verkleinern                     //
           //----------------------------------------------------------------//
           switch (wParam)
           {
              //-------------------------------------------------------------//
              // "Kleiner" betätigt                                          //
              //-------------------------------------------------------------//
              case IDC_SMALLER:
```

```
            rc.left   += cxClient / 20;
            rc.right  -= cxClient / 20;
            rc.top    += cyClient / 20;
            rc.bottom -= cyClient / 20;
            break;

       //-------------------------------------------------------------//
       // "Größer"  betätigt                                          //
       //-------------------------------------------------------------//
       case IDC_LARGER:
            rc.left   -= cxClient / 20;
            rc.right  += cxClient / 20;
            rc.top    -= cyClient / 20;
            rc.bottom += cyClient / 20;
            break;
     }
     MoveWindow (hwnd, rc.left, rc.top, rc.right - rc.left,
                 rc.bottom - rc.top, TRUE);
     return (0);

//---------------------------------------------------------------------//
// Die DRAWITEM-Message wird gesendet, wenn es nötig ist, ein Dialog-  //
// element neu zu zeichnen (z.B. weil es verdeckt war)                 //
//---------------------------------------------------------------------//
case WM_DRAWITEM:
     //-----------------------------------------------------------------//
     // Die DRAWITEMSTRUCT enthält weitergehende Angaben dazu, was zu   //
     // zu zeichnen ist (gesamtes Element, wurde Element deselektiert   //
     // usw.) und welchen Zustand des Objekt angenommen hat (hat es den //
     // Fokus bekommen, wurde es abgeschaltet etc.)                     //
     //-----------------------------------------------------------------//
     lpdis = (LPDRAWITEMSTRUCT) lParam;

     //-----------------------------------------------------------------//
     // weißer Hintergrund, schwarzer Rahmen                            //
     //-----------------------------------------------------------------//
     FillRect (lpdis->hDC, &lpdis->rcItem,
               (HBRUSH)GetStockObject (GRAY_BRUSH));
     FrameRect(lpdis->hDC, &lpdis->rcItem,
               (HBRUSH)GetStockObject (BLACK_BRUSH));

     //-----------------------------------------------------------------//
     // Dreiecke zeichnen                                               //
     //-----------------------------------------------------------------//
     cx = lpdis->rcItem.right - lpdis->rcItem.left;
     cy = lpdis->rcItem.bottom - lpdis->rcItem.top;

     //-----------------------------------------------------------------//
     // Feststellen welches Element neu zu zeichnen ist...              //
     //-----------------------------------------------------------------//
     switch (lpdis->CtlID)
     {
        //--------------------------------------------------------------//
        // "Kleiner" zeichnen                                           //
        //--------------------------------------------------------------//
        case IDC_SMALLER:
             pt[0].x = 3 * cx / 8; pt[0].y = 1 * cy / 8;
             pt[1].x = 5 * cx / 8; pt[1].y = 1 * cy / 8;
             pt[2].x = 4 * cx / 8; pt[2].y = 3 * cy / 8;
             Triangle(lpdis->hDC, pt);

             pt[0].x = 7 * cx / 8; pt[0].y = 3 * cy / 8;
             pt[1].x = 7 * cx / 8; pt[1].y = 5 * cy / 8;
             pt[2].x = 5 * cx / 8; pt[2].y = 4 * cy / 8;
             Triangle(lpdis->hDC, pt);

             pt[0].x = 5 * cx / 8; pt[0].y = 7 * cy / 8;
             pt[1].x = 3 * cx / 8; pt[1].y = 7 * cy / 8;
             pt[2].x = 4 * cx / 8; pt[2].y = 5 * cy / 8;
             Triangle(lpdis->hDC, pt);

             pt[0].x = 1 * cx / 8; pt[0].y = 5 * cy / 8;
             pt[1].x = 1 * cx / 8; pt[1].y = 3 * cy / 8;
             pt[2].x = 3 * cx / 8; pt[2].y = 4 * cy / 8;
             Triangle(lpdis->hDC, pt);
             break;

        //--------------------------------------------------------------//
        // "Größer"  zeichnen                                           //
        //--------------------------------------------------------------//
        case IDC_LARGER:
             pt[0].x = 5 * cx / 8; pt[0].y = 3 * cy / 8;
             pt[1].x = 3 * cx / 8; pt[1].y = 3 * cy / 8;
```

```
                    pt[2].x = 4 * cx / 8; pt[2].y = 1 * cy / 8;
                    Triangle(lpdis->hDC, pt);

                    pt[0].x = 5 * cx / 8; pt[0].y = 5 * cy / 8;
                    pt[1].x = 5 * cx / 8; pt[1].y = 3 * cy / 8;
                    pt[2].x = 7 * cx / 8; pt[2].y = 4 * cy / 8;
                    Triangle(lpdis->hDC, pt);

                    pt[0].x = 3 * cx / 8; pt[0].y = 5 * cy / 8;
                    pt[1].x = 5 * cx / 8; pt[1].y = 5 * cy / 8;
                    pt[2].x = 4 * cx / 8; pt[2].y = 7 * cy / 8;
                    Triangle(lpdis->hDC, pt);

                    pt[0].x = 3 * cx / 8; pt[0].y = 3 * cy / 8;
                    pt[1].x = 3 * cx / 8; pt[1].y = 5 * cy / 8;
                    pt[2].x = 1 * cx / 8; pt[2].y = 4 * cy / 8;
                    Triangle(lpdis->hDC, pt);
                    break;
               }

               //-------------------------------------------------------------------//
               // Eigenes Element invertieren, falls Maustaste gedrückt gehalten  //
               //-------------------------------------------------------------------//
               if (lpdis->itemState & ODS_SELECTED)
                  InvertRect(lpdis->hDC, &lpdis->rcItem);

               //-------------------------------------------------------------------//
               // markieren, wenn Element den Eingabefokus hat                     //
               //-------------------------------------------------------------------//
               if (lpdis->itemState & ODS_FOCUS)
               {
                    lpdis->rcItem.left   += cx / 16;
                  lpdis->rcItem.top    += cy / 16;
                  lpdis->rcItem.right  -= cx / 16;
                  lpdis->rcItem.bottom -= cy / 16;
                  DrawFocusRect (lpdis->hDC, &lpdis->rcItem);
               }
               return (0);

          //-------------------------------------------------------------------//
          // Programm beenden                                                  //
          //-------------------------------------------------------------------//
          case WM_DESTROY :
               PostQuitMessage (0);
               return (0);
     }
     return DefWindowProc (hwnd, message, wParam, lParam);
}
```

8.17 Taschenrechner

```
//*************************************************************************//
//   P11.cpp – Taschenrechner                                              //
//*************************************************************************//
#include <windows.h>
#include <limits.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

//-------------------------------------------------------------------------//
// Defines aus dem Resource-Workshop einlesen, so daß sie hier verwendbar sind//
//-------------------------------------------------------------------------//
// Dies sind eigene Windowsmessages, der Bereich unterhalb von WM_USER wird   //
// von Windows selbst benötigt                                               //
//-------------------------------------------------------------------------//
#include "p110.rh"

//-------------------------------------------------------------------------
// Die folgenden Zeilen sind nur für den Borland C++ Builder interessant, um die
// von Hand erstellten Resourcen korrekt zu laden ...
//-------------------------------------------------------------------------
#ifdef __BORLANDC__
 #if __BORLANDC__ > 0x0502
   #include <condefs.h>
   USERC("p110.rc");
 #endif
#endif

//-------------------------------------------------------------------------
#define WMU_MYCOPY  WM_USER+1
#define WMU_MYPASTE WM_USER+2

//-------------------------------------------------------------------------//
// Prototyp der Handlerfunction, die die Messages der Client-Area verarbeitet //
//-------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

//-------------------------------------------------------------------------//
// Prototyp der Handlerfunction, die die Messagebox "About" verarbeitet       //
//-------------------------------------------------------------------------//
BOOL FAR PASCAL AboutProc (HWND hDlg, UINT message, WPARAM wParam, LONG lParam);
BOOL FAR PASCAL Dlg2Proc  (HWND hDlg, UINT message, WPARAM wParam, LONG lParam);

//-------------------------------------------------------------------------//
// globale Variablen                                                       //
//-------------------------------------------------------------------------//
HINSTANCE hInst;

//-------------------------------------------------------------------------//
// Alle Windowsprogramme beginnen mit "WinMain" statt mit "main"              //
//-------------------------------------------------------------------------//
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine,
                    int nCmdShow)
{
   static char szWindowclassname[] = "Calc";
   HWND        hWindow;
   MSG         aMessage;
   WNDCLASS    aWndclass;
   HBRUSH      myBrush;

   myBrush = CreateSolidBrush (RGB (100, 100, 255));

   if (!hPrevInstance)
   {
      aWndclass.style         = CS_HREDRAW | CS_VREDRAW;
      aWndclass.lpfnWndProc   = WndProc;
      aWndclass.cbClsExtra    = 0;
      //-----------------------------------------------------------------//
      // Gibt die Anzahl der Verwaltungsbytes an, die zusätzlich im Anschluß  //
      // an die Window-Instanz erzeugt werden sollen. Normalerweise ist       //
      // dieser Wert Null. Wird die WNDCLASS-Struktur aber wie hier dazu ver- //
      // wendet statt eines Fensters sofort einen Dialog mit dem Classname zu //
      // registrieren, so so muß der Wert auf DLGWINDOWEXTRA gesetzt werden   //
      //-----------------------------------------------------------------//
      aWndclass.cbWndExtra    = DLGWINDOWEXTRA;
      aWndclass.hInstance     = hInstance;
      //-----------------------------------------------------------------//
```

```
      // Zur Abwechslung holen wir uns einmal ein Icon von der Platte. Das    //
      // Icon heißt hier "P110.ICO"                                           //
      //-------------------------------------------------------------------------//
      aWndclass.hIcon         = LoadIcon (NULL, "P110");
      aWndclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
      aWndclass.hbrBackground = myBrush;// (HBRUSH)GetStockObject (GRAY_BRUSH);
      aWndclass.lpszMenuName  = MAKEINTRESOURCE(IDM_MENU1);
      aWndclass.lpszClassName = szWindowclassname;

      RegisterClass (&aWndclass);
   }

   //-------------------------------------------------------------------------//
   // Die zur Laufzeit erzeugten Dialogboxen benötigen für ihre eigene Ver-   //
   // waltung den Instanzenzähler. Dazu muß dieser global gepuffert werden    //
   //-------------------------------------------------------------------------//
   hInst = hInstance;

   //-------------------------------------------------------------------------//
   // Statt CreateWindow rufen wir hier Create Dialog auf - und zwar den      //
   // Dialog, der genauso heißt wie die Klasse                                //
   //-------------------------------------------------------------------------//
   hWindow = CreateDialog (hInstance, szWindowclassname, 0, NULL);

   if (hWindow)
   {
      ShowWindow (hWindow, nCmdShow);

      while (GetMessage (&aMessage, NULL, 0, 0))
      {
         TranslateMessage (&aMessage);
         DispatchMessage (&aMessage);
      }
   }
   else
   {
      MessageBeep (0);
   }

   //-------------------------------------------------------------------------//
   // Nicht vergessen den Brush zu zerstören...
   //-------------------------------------------------------------------------//
   DeleteObject (myBrush);

   return (aMessage.wParam);
}

//-------------------------------------------------------------------------//
// Ergebnisanzeige                                                         //
//-------------------------------------------------------------------------//
void ShowNumber (HWND hwnd, double fNumber)
{
   char szBuffer [50];

   //-------------------------------------------------------------------------//
   // Wir ändern einfach die Beschriftung des Buttons IDC_ERGEBNIS            //
   //-------------------------------------------------------------------------//
   sprintf (szBuffer, "%lf", fNumber);
   SetDlgItemText (hwnd, IDC_ERGEBNIS, szBuffer);
}

//-------------------------------------------------------------------------//
// Hier werden die Berechnungen durchgeführt, die den Buttons entsprechen  //
//-------------------------------------------------------------------------//
double CalcIt (double fFirstNum, int nOperation, double fSecNum)
{
   switch (nOperation)
   {
      case IDC_PLUS   : return (fFirstNum +  fSecNum);
      case IDC_MINUS  : return (fFirstNum -  fSecNum);
      default         : return (0.0);
   }
}

//-------------------------------------------------------------------------//
// Handlerfunction der Client-Area                                         //
//-------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   //-------------------------------------------------------------------------//
   // Unter 16-bit Windows muß der Funktionsaufruf für den Handler der Dialog-//
   // box "ABOUT" zuerst in einen FARPROC (32bit) Pointer übersetzt werden    //
   // Unter 32-Bit Windows ist dies nicht nötig, die Adresse bereits das      //
```

```
// richtige Format hat                                                    //
// FARPROC    lpAboutProc;                                                //
//------------------------------------------------------------------------//
static BOOL   bFirstNumber = TRUE;
static int    nLastOp      = -1;
HFONT         hFont, hoFont;
int           nOldBk;
COLORREF      nOldTxtCol;
BOOL          bShowResult  = FALSE;
static double fResult = 0.0, fFirstNum = 0.0, fSecNum = 0.0;
int           wID, nAdd;
HGLOBAL       hData;
LPSTR         pData;
char          szHelp [100] = "";
RECT          rect;
HDC           dc;
PAINTSTRUCT   ps;

switch (message)
{
   //------------------------------------------------------------------------//
   // Dies ist die Auswertung einer selbstdefinierten Message, die aus dem //
   // Command heraus geschickt wird. Aufgbe dieser Message ist es, den     //
   // gerade in der Anzeige befindlichen Wert in die Zwischenablage zu     //
   // schreiben                                                            //
   //------------------------------------------------------------------------//
   case WMU_MYCOPY:
        //--------------------------------------------------------------//
        // da die Zwischenablage allen Programmen gemeinsam zugänglich ist //
        // gehört sie nicht zu einem Programm, sondern wird von Windows in //
        // einem eigenen Datensegment verwaltet. Ein Segment der benötig-  //
        // ten Größe versuchen wir zunächst von Windows zu erhalten        //
        //--------------------------------------------------------------//
        hData = GlobalAlloc (GMEM_MOVEABLE, 100L);
        if (!hData) return (0);

        //--------------------------------------------------------------//
        // Bevor wir mit dem Zwischenablage-Speicher arbeiten können, muß  //
        // man das Handle in einen Pointer umwandeln.                   //
        //--------------------------------------------------------------//
        pData = (LPSTR)GlobalLock (hData);
        if (!pData)
        {
           GlobalFree (hData);
           return (0);
        }
        GlobalUnlock (hData);

        //--------------------------------------------------------------//
        // Da wir den Zeiger auf den Speicherbereich von GlobalLock er-    //
        // halten haben, besorgen wir uns jetzt den Text, den wir dort     //
        // hineinschreiben wollen                                       //
        //--------------------------------------------------------------//
        GetDlgItemText (hwnd, IDC_ERGEBNIS, pData, 45);

        //--------------------------------------------------------------//
        // jetzt öffnen wir das Clipboard. Dadurch ist es für alle anderen //
        // Anwendungen nicht mehr zugänglich                            //
        //--------------------------------------------------------------//
        if (OpenClipboard (hwnd))
        {
           //-----------------------------------------------------------//
           // Den bisherigen Inhalt lassen wir löschen und schreiben dafür //
           // unseren Text hinein. Auf gar keinen Fall dürfen wir ver-     //
           // gessen das CLIPBOARD wieder zu schließen, da es sonst erst   //
           // nach Beendigung unseres Programms wieder zugänglich wäre und //
           // andere Programme ggf. blockiert                              //
           //-----------------------------------------------------------//
           EmptyClipboard ();
           SetClipboardData (CF_TEXT, hData);
           CloseClipboard ();

           //-----------------------------------------------------------//
           // Weil man das Ergebnis nicht sehen kann, machen wir uns selbst//
           // zu Testzwecken eine kleine Mitteilung                        //
           //-----------------------------------------------------------//
           MessageBox (hwnd, "kopiert...", "CLIPBOARD", MB_OK |
                       MB_APPLMODAL | MB_ICONINFORMATION);
        }
        //--------------------------------------------------------------//
        // Sollte das ganze nicht geklappt haben, geben wir das Segment    //
        // wieder frei                                                  //
        //--------------------------------------------------------------//
```

```
     else
        GlobalFree(hData);
     return 0;

//---------------------------------------------------------------------//
// Dies ist die Auswertung einer selbstdefinierten Message, die aus dem //
// Command heraus geschickt wird                                       //
//---------------------------------------------------------------------//
case WMU_MYPASTE:
     //---------------------------------------------------------------//
     // Wir dürfen nur einfügen, wenn auch ein Text im CLIPBOARD ge-    //
     // speichert ist                                                 //
     //---------------------------------------------------------------//
     if (!IsClipboardFormatAvailable (CF_TEXT))
     {
        MessageBox (hwnd, "Kein Textformat", "CLIPBOARD", MB_OK |
                    MB_APPLMODAL | MB_ICONINFORMATION);
        return (0);
     }

     //---------------------------------------------------------------//
     // jetzt öffnen wir das Clipboard. Dadurch ist es für alle anderen //
     // Anwendungen nicht mehr zugänglich                             //
     //---------------------------------------------------------------//
     if (OpenClipboard (hwnd))
     {
        //------------------------------------------------------------//
        // Jetzt holen wir uns die Daten aus dem Clipboard zurück       //
        // Schließen nicht vergessen, siehe oben !                     //
        //------------------------------------------------------------//
        hData = GetClipboardData (CF_TEXT);
        lstrcpy (szHelp, (LPSTR)hData);
        CloseClipboard ();

        //------------------------------------------------------------//
        // Den gerade gelesenen Text zeigen wir uns zu Testzwecken an   //
        //------------------------------------------------------------//
        MessageBox (hwnd, szHelp, "CLIPBOARD gelesen", MB_OK |
                    MB_APPLMODAL | MB_ICONINFORMATION);
     }

     //---------------------------------------------------------------//
     // Den Text wandeln wir wieder in eine Zahl um. Nicht-Ziffern      //
     // gehen natürlich verloren                                      //
     //---------------------------------------------------------------//
     if (bFirstNumber)
     {
        fFirstNum = atof (szHelp);
        sprintf (szHelp, "%lf", fFirstNum);
     }
     else
     {
        fSecNum = atof (szHelp);
        sprintf (szHelp, "%lf", fFirstNum);
     }
     //---------------------------------------------------------------//
     // Das Ergebnis müssen wir noch anzeigen lassen und das wär's dann //
     //---------------------------------------------------------------//
     SetDlgItemText (hwnd, IDC_ERGEBNIS, szHelp);
     return 0;

//---------------------------------------------------------------------//
// Messages der einzelnen Dialogelemente (Controls)                    //
//---------------------------------------------------------------------//
case WM_COMMAND :
     nAdd = -1;
     //---------------------------------------------------------------//
     // Tastatureingabefokus setzen                                   //
     //---------------------------------------------------------------//
     SetFocus (hwnd);
     wID = LOWORD(wParam);
     switch (wID)
     {
        //------------------------------------------------------------//
        // Programm über Menü beenden, dafür senden wir uns selbst eine //
        // DESTROY-Message                                             //
        //------------------------------------------------------------//
        case IDM_MYEXIT:
             SendMessage (hwnd, WM_DESTROY, 0, 0);
             return (0);

        //------------------------------------------------------------//
        // Menüauswahl: Aus dem Clipboard einfügen                     //
```

```
//-------------------------------------------------------------//
case IDM_MYPASTE:
     SendMessage (hwnd, WMU_MYPASTE, 0, 0);
     return (0);

//-------------------------------------------------------------//
// Menüauswahl: in das CLIPBOARD kopieren                      //
//-------------------------------------------------------------//
case IDC_ERGEBNIS:
case IDM_MYCOPY:
     SendMessage (hwnd, WMU_MYCOPY, 0, 0);
     return (0);

//-------------------------------------------------------------//
// Menüauswahl: Windowshilfe aufrufen                          //
//-------------------------------------------------------------//
case IDM_MYHELP :
     WinHelp (hwnd, "c:\\windows\\help\\windows.hlp",
              HELP_CONTENTS, 0L);
     return 0;

//-------------------------------------------------------------//
// Menüauswahl: Unterdialog "About" / "Über" aufrufen          //
//-------------------------------------------------------------//
case IDM_MYABOUT:
     //---------------------------------------------------------//
     // Für 16-Bit-Windows muß die nachfolgende Zeile durch     //
     // diesen Dreizeiler ersetzt werden, um die 16-Bit-Adresse //
     // in eine eindeutige 32-Bit-Adressen umzusetzen           //
     // lpAboutProc = MakeProcInstance (AboutProc, hInst);      //
     //---------------------------------------------------------//
      // DialogBox (hInst, MAKEINTRESOURCE(IDD_ABOUT), hwnd,     //
     //            (DLGPROC)lpAboutProc);                       //
     // FreeProcInstance (lpAboutProc);                         //
     //---------------------------------------------------------//
     DialogBox (hInst, MAKEINTRESOURCE(IDD_ABOUT), hwnd,
                (DLGPROC)AboutProc);
     return 0;

case CM_DLG2:
     DialogBox (hInst, MAKEINTRESOURCE(DLG2), hwnd,
                (DLGPROC)Dlg2Proc);
     return 0;

//-------------------------------------------------------------//
// Einer der Zahlenbuttons wurde gedrückt                      //
//-------------------------------------------------------------//
case IDC_0     : nAdd = 0; break;
case IDC_1     : nAdd = 1; break;
case IDC_2     : nAdd = 2; break;
case IDC_3     : nAdd = 3; break;
case IDC_4     : nAdd = 4; break;
case IDC_5     : nAdd = 5; break;
case IDC_6     : nAdd = 6; break;
case IDC_7     : nAdd = 7; break;
case IDC_8     : nAdd = 8; break;
case IDC_9     : nAdd = 9; break;

//-------------------------------------------------------------//
// Programm in den Grundzustand bringen                        //
//-------------------------------------------------------------//
case IDC_CLEAR :
     bFirstNumber = TRUE;
     nLastOp      = -1;
     fFirstNum    = 0.0,
     fSecNum      = 0.0;
     nAdd         = 0;
     break;

//-------------------------------------------------------------//
// Einer der Operatorbuttons wurde gedrückt                    //
//-------------------------------------------------------------//
default :
     if (bFirstNumber) bFirstNumber = FALSE;
     else
     {
        bShowResult = TRUE;
        if (wID == IDC_EQUALS)
           fResult = CalcIt (fFirstNum, nLastOp, fSecNum);
        else
           fResult = CalcIt (fFirstNum, wID, fSecNum);
     }
     nLastOp = wID;
```

```
                  break;
            }

            //-------------------------------------------------------------//
            // Eingaben der Zahlen/Operatoren auswerten und anzeigen       //
            //-------------------------------------------------------------//
            if (bShowResult)
            {
               ShowNumber (hwnd, fResult);
               fFirstNum = fResult;
               fResult = 0.0;
               fSecNum = 0.0;
               bFirstNumber = FALSE;
            }
            else
            {
               if ((bFirstNumber) && (nAdd != -1))
               {
                  fFirstNum = fFirstNum * 10 + nAdd;
                  ShowNumber (hwnd, fFirstNum);
               }
               if ((!bFirstNumber) && (nAdd != -1))
               {
                  fSecNum = fSecNum * 10 + nAdd;
                  ShowNumber (hwnd, fSecNum);
               }
            }
            return 0;

      //-------------------------------------------------------------------//
      // Zeichnen der zusätzlichen Elemente                                //
      //-------------------------------------------------------------------//
      case WM_PAINT:
            //-------------------------------------------------------------//
            // Rechteck der Client-Area ermitteln                          //
            //-------------------------------------------------------------//
            GetClientRect (hwnd, &rect);
            rect.left += 5;
            rect.top  += 10;
            rect.right -= 5;
            rect.bottom = rect.top + 30;

            //-------------------------------------------------------------//
            // Einen True-Type-Font in gewünschter Ausprägung erzeugen     //
            //-------------------------------------------------------------//
            hFont  = CreateFont (30, 0, 0, 0, FW_THIN, TRUE, FALSE, FALSE,
                                 ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                                 CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                                 DEFAULT_PITCH, "Braggadocio");

            //-------------------------------------------------------------//
            // Ermittlung des Device-Contextes und Auswahl der "Mal-Utensilien"//
            // dabei die vorherigen Zustände merken, um sie hinterher sauber   //
            // wiederherzustellen                                          //
            //-------------------------------------------------------------//
            dc         = BeginPaint (hwnd, &ps);
            hoFont     = (HFONT)SelectObject (dc, hFont);
            nOldBk     = GetBkMode (dc);
            nOldTxtCol = GetTextColor (dc);
            SetBkMode (dc, TRANSPARENT);
            SetTextColor (dc, RGB (255,255,0));
            DrawText (dc, "Calculator", 10, &rect, 17);
            SetTextColor (dc, nOldTxtCol);
            SetBkMode    (dc, nOldBk);
            SelectObject (dc, hoFont);
            DeleteObject (hFont);
            EndPaint (hwnd, &ps);
            return (DefWindowProc (hwnd, message, wParam, lParam));

      //-------------------------------------------------------------------//
      // Beenden des Programms                                             //
      //-------------------------------------------------------------------//
      case WM_DESTROY :
            PostQuitMessage (0);
            return 0;
   }
   return DefWindowProc (hwnd, message, wParam, lParam);
}

//-----------------------------------------------------------------------//
// Handlerfunction der About-Box                                         //
//-----------------------------------------------------------------------//
#pragma argsused
```

```
BOOL FAR PASCAL AboutProc (HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
   switch (message)
   {
      //------------------------------------------------------------------------//
      // Initialisierung der Box (falls nötig)                                  //
      //------------------------------------------------------------------------//
      case WM_INITDIALOG:
           break;

      //------------------------------------------------------------------------//
      // Hier den einzigen Button abfragen und dann den Dialog beenden          //
      //------------------------------------------------------------------------//
      case WM_COMMAND:
           switch (wParam)
           {
              case IDOK:
                   EndDialog (hDlg, TRUE);
                   break;
           }
           break;
   }
   return (FALSE);
}

//------------------------------------------------------------------------------//
// Handlerfunction der About-Box                                                //
//------------------------------------------------------------------------------//
#pragma argsused
BOOL FAR PASCAL Dlg2Proc (HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
   char szBuffer [200];

   switch (message)
   {
      //------------------------------------------------------------------------//
      // Initialisierung der Box (falls nötig)                                  //
      //------------------------------------------------------------------------//
      case WM_INITDIALOG:
           GetPrivateProfileString ("Register",
              "Reg", "Nix", (LPSTR)szBuffer, 200,
              "E:\\27275\\my.ini");

           SendDlgItemMessage (hDlg, IDC_EDIT2, EM_LIMITTEXT, 15, 0);
           SetWindowText (GetDlgItem (hDlg, IDC_EDIT2),
                          (LPSTR)szBuffer);
           break;

      //------------------------------------------------------------------------//
      // Hier den einzigen Button abfragen und dann den Dialog beenden          //
      //------------------------------------------------------------------------//
      case WM_COMMAND:
           switch (wParam)
           {
              case IDOK:
                   GetWindowText(GetDlgItem (hDlg, IDC_EDIT2),
                    (LPSTR)szBuffer, 200);
                   WritePrivateProfileString ("Register",
                    "Reg", (LPSTR)szBuffer,
                    "E:\\27275\\my.ini");
                   EndDialog (hDlg, TRUE);
                   break;
           }
           break;
   }
   return (FALSE);
}
```

8.17.1 Resource-Header

```
//************************************************************************//
//   P11.rh – Taschenrechner                                              //
//************************************************************************//
#define DLGMAIN 11111

#define CM_DLG2   1
#define CM_POPUPEINTRAG2   62532
#define CM_EINTRAG2   62720
#define CM_EINTRAG1   62568
#define IDC_INSULT   101
#define IDC_BUTTON1   101
#define IDC_EDIT2   102
```

```
#define IDC_EDIT1   101
#define DLG2   3000
#define IDD_ABOUT   2

#define IDC_GROUPBOX1 501

#define IDM_MENU1     1000
#define IDM_MYABOUT 1001
#define IDM_MYHELP  1002
#define IDM_MYEXIT  1003
#define IDM_MYCOPY  1004
#define IDM_MYPASTE 1005

#define IDI_P11   1

#define IDC_0         100
#define IDC_1         101
#define IDC_2         102
#define IDC_3         103
#define IDC_4         104
#define IDC_5         105
#define IDC_6         106
#define IDC_7         107
#define IDC_8         108
#define IDC_9         109

#define IDC_PLUS      120
#define IDC_MINUS      121
#define IDC_EQUALS   130
#define IDC_CLEAR      140
#define IDC_ERGEBNIS   200
```

8.17.2 Resource-Datei

```
//************************************************************************//
//   P11.rc – Taschenrechner                                              //
//************************************************************************//

#include "p11.rh"

Calc DIALOG 0, 0, 126, 160
STYLE  DS_3DLOOK  |  WS_OVERLAPPED  |  WS_VISIBLE  |  WS_CAPTION  |  WS_SYSMENU  |
WS_MINIMIZEBOX
Class "Calc"
FONT 8, "MS Sans Serif"
{
 CONTROL "1", IDC_1, "button", BS_PUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 8, 88, 16, 14
 CONTROL "2", IDC_2, "button", BS_PUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 28, 88, 16, 14
 CONTROL "3", IDC_3, "button", BS_PUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 48, 88, 16, 14
 CONTROL "4", IDC_4, "button", BS_PUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 8, 68, 16, 14
 CONTROL "5", IDC_5, "button", BS_PUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 28, 68, 16, 14
 CONTROL "6", IDC_6, "button", BS_PUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 48, 68, 16, 14
 CONTROL "7", IDC_7, "button", BS_PUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 8, 48, 16, 14
 CONTROL "8", IDC_8, "button", BS_PUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 28, 48, 16, 14
 CONTROL "9", IDC_9, "button", BS_PUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 48, 48, 16, 14
 CONTROL "0", IDC_0, "button", BS_PUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 8, 108, 16, 14
 CONTROL "+", IDC_PLUS, "button", BS_PUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 80, 88, 16, 14
 CONTROL "-", IDC_MINUS, "button", BS_PUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE
| WS_TABSTOP, 100, 68, 16, 14
 CONTROL "=", IDC_EQUALS, "button", BS_PUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE
| WS_TABSTOP, 80, 108, 36, 14
 CONTROL  "C/CE",  IDC_CLEAR,  "button",  BS_PUSHBUTTON  |  BS_CENTER  |  WS_CHILD  |
WS_VISIBLE | WS_TABSTOP, 80, 48, 36, 14
 CONTROL  "0.0000",  IDC_ERGEBNIS,  "button",  BS_PUSHBUTTON  |  BS_CENTER  |  WS_CHILD  |
WS_VISIBLE | WS_TABSTOP, 4, 28, 116, 14
 CONTROL "Frame1", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 4, 44, 64,
84
 CONTROL "Frame2", -1, "static", SS_ETCHEDFRAME | WS_CHILD | WS_VISIBLE, 76, 44, 44,
84
}
```

```
IDI_P11 ICON
{
 '00 00 01 00 01 00 20 20 10 00 00 00 00 00 E8 02'
 '00 00 16 00 00 00 28 00 00 00 20 00 00 00 40 00'
 '00 00 01 00 04 00 00 00 00 00 80 02 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 80 00 00 80 00 00 00 80 80 00 80 00'
 '00 00 80 00 80 00 80 80 00 00 C0 C0 C0 00 80 80'
 '80 00 00 00 FF 00 00 FF 00 00 00 FF FF 00 FF 00'
 '00 00 FF 00 FF 00 FF FF 00 00 FF FF FF 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 0F FF'
 'FF FF FF FF FF F0 0F FF FF FF FF FF FF F0 00 77'
 '77 77 77 77 77 F0 00 77 77 77 77 77 77 F0 00 70'
 '87 77 77 78 07 F0 00 77 77 77 77 78 07 F0 00 78'
 '08 77 77 80 87 F0 00 77 77 77 77 80 87 F0 00 77'
 '80 87 78 08 77 F0 00 77 77 77 78 08 77 F0 00 77'
 '78 08 80 87 77 F0 00 77 77 77 80 87 77 F0 00 77'
 '77 80 08 77 77 F0 00 77 77 78 08 77 77 F0 00 77'
 '77 80 08 77 77 F0 00 77 77 80 87 77 77 F0 00 77'
 '78 08 80 87 77 F0 00 77 78 08 77 77 77 F0 00 77'
 '80 87 78 08 77 F0 00 77 80 87 77 77 77 F0 00 78'
 '08 77 77 80 87 F0 00 78 08 77 77 77 77 F0 00 70'
 '87 77 77 78 07 F0 00 70 87 77 77 77 77 F0 00 77'
 '77 77 77 77 77 F0 00 77 77 77 77 77 77 F0 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 0F FF'
 'FF FF FF FF FF F0 0F FF FF FF FF FF FF F0 00 77'
 '77 77 77 77 77 F0 00 77 77 77 77 77 77 F0 00 77'
 '77 70 07 77 77 F0 00 77 77 77 77 77 77 F0 00 77'
 '77 70 07 77 77 F0 00 77 77 77 77 77 77 F0 00 77'
 '77 70 07 77 77 F0 00 77 77 77 77 77 77 F0 00 77'
 '77 70 07 77 77 F0 00 77 77 77 77 77 77 F0 00 70'
 '00 00 00 00 07 F0 00 70 00 00 00 00 07 F0 00 70'
 '00 00 00 00 07 F0 00 70 00 00 00 00 07 F0 00 77'
 '77 70 07 77 77 F0 00 77 77 77 77 77 77 F0 00 77'
 '77 70 07 77 77 F0 00 77 77 77 77 77 77 F0 00 77'
 '77 70 07 77 77 F0 00 77 77 77 77 77 77 F0 00 77'
 '77 70 07 77 77 F0 00 77 77 77 77 77 77 F0 00 77'
 '77 77 77 77 77 F0 00 77 77 77 77 77 77 F0 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF'
 'FF FF 80 01 80 01 80 01 80 01 80 01 80 01 80 01'
 '80 01 80 01 80 01 80 01 80 01 80 01 80 01 80 01'
 '80 01 80 01 80 01 80 01 80 01 80 01 80 01 80 01'
 '80 01 80 01 80 01 80 01 FF FF FF FF FF FF'
 'FF FF 80 01 80 01 80 01 80 01 80 01 80 01 80 01'
 '80 01 80 01 80 01 80 01 80 01 80 01 80 01 80 01'
 '80 01 80 01 80 01 80 01 80 01 80 01 80 01 80 01'
 '80 01 80 01 80 01 80 01 FF FF FF FF'
}

IDM_MENU1 MENU
{
 POPUP "&Datei"
 {
  MENUITEM "&Beenden", IDM_MYEXIT
 }

 POPUP "&Pop-Up"
 {
  MENUITEM "&Dialog 2", CM_DLG2
 }

 POPUP "&Editieren"
 {
  MENUITEM "&Kopieren", IDM_MYCOPY
  MENUITEM "&Einfügen", IDM_MYPASTE
 }

 POPUP "&Hilfe"
 {
  MENUITEM "Ü&ber", IDM_MYABOUT
  MENUITEM SEPARATOR
  MENUITEM "&Hilfe", IDM_MYHELP
 }

}

IDD_ABOUT DIALOG 23, 20, 80, 64
STYLE DS_MODALFRAME | DS_3DLOOK | DS_CONTEXTHELP | WS_POPUP | WS_VISIBLE | WS_CAPTION
| WS_SYSMENU
CAPTION "About"
```

```
FONT 8, "MS Sans Serif"
{
 CONTROL "OK", IDOK, "BUTTON", BS_PUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 16, 43, 50, 14
 CONTROL "Copyright", IDC_GROUPBOX1, "button", BS_GROUPBOX | WS_CHILD | WS_VISIBLE |
WS_GROUP, 8, 4, 64, 34
 CONTROL "H. Gorski 1997", -1, "static", SS_CENTER | WS_CHILD | WS_VISIBLE, 16, 17,
48, 13
}

DLG2 DIALOG 0, 0, 228, 60
STYLE DS_MODALFRAME | DS_3DLOOK | DS_CONTEXTHELP | WS_POPUP | WS_VISIBLE | WS_CAPTION
| WS_SYSMENU
CAPTION ""
FONT 8, "MS Sans Serif"
{
 CONTROL "OK", IDOK, "BUTTON", BS_PUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 76, 34, 50, 14
 CONTROL "Registrierungsinfo", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 4, 9,
60, 8
 CONTROL "Textfeld2", IDC_EDIT2, "edit", ES_LEFT | WS_CHILD | WS_VISIBLE | WS_BORDER
| WS_TABSTOP, 68, 9, 140, 12
}
```

8.17.3 Definition-Datei

```
//************************************************************************//
//   P11.def – Taschenrechner                                             //
//************************************************************************//

;----------------------
; Modul-Definition P11
;----------------------

NAME           P11
DESCRIPTION    'Rechner'
EXETYPE        WINDOWS
STUB           'WINSTUB.EXE'
CODE           PRELOAD MOVEABLE DISCARDABLE
DATA           PRELOAD MOVEABLE MULTIPLE
HEAPSIZE       1000000
STACKSIZE      1000000
```

8.18 Drucken

```
//*************************************************************************//
//   P13.cpp - einfache Druckausgabe                                       //
//*************************************************************************//
#include <windows.h>
#include <string.h>
#include <stdio.h>

//-------------------------------------------------------------------------//
// Prototypen                                                              //
//-------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);
BOOL PrintMyPage  (HWND);
HDC  GetPrinterDC (void);
void fnWMPaint    (HDC, int, int);

//-------------------------------------------------------------------------//
// globale Variablen                                                       //
//-------------------------------------------------------------------------//
HINSTANCE hInst;
char      szWindowclassname[] = "P130";
short     cxClient, cyClient;

//-------------------------------------------------------------------------//
// Alle Windowsprogramme beginnen mit "WinMain" statt mit "main"           //
//-------------------------------------------------------------------------//
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine,
                    int nCmdShow)
{
   HWND        hWindow;
   MSG         aMessage;
   WNDCLASS    aWndclass;

   if (!hPrevInstance)
   {
      aWndclass.style         = CS_HREDRAW | CS_VREDRAW;
      aWndclass.lpfnWndProc   = WndProc;
      aWndclass.cbClsExtra    = 0;
      aWndclass.cbWndExtra    = 0;
      aWndclass.hInstance     = hInstance;
      aWndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
      aWndclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
      aWndclass.hbrBackground = (HBRUSH)GetStockObject (WHITE_BRUSH);
      aWndclass.lpszMenuName  = NULL;
      aWndclass.lpszClassName = szWindowclassname;

      RegisterClass (&aWndclass);
   }

   hInst = hInstance;

   hWindow = CreateWindow (szWindowclassname, "Druck-Demo",
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL);

   ShowWindow   (hWindow, nCmdShow);
   UpdateWindow (hWindow);

   while (GetMessage (&aMessage, NULL, 0, 0))
   {
      TranslateMessage (&aMessage);
      DispatchMessage  (&aMessage);
   }
   return (aMessage.wParam);
}

//-------------------------------------------------------------------------//
// Device-Context des Druckers (Treiber, Auflösung etc.) ermitteln         //
//-------------------------------------------------------------------------//
HDC GetPrinterDC (void)
{
   static char szPrinter [80];
   char        *szDevice, *szDriver, *szOutput;

   //----------------------------------------------------------------------//
   // Daten aus der WIN.INI-Datei lesen                                    //
   //----------------------------------------------------------------------//
```

```
   GetProfileString ("windows", "device", ",,,", szPrinter, 80);

   //------------------------------------------------------------------------//
   // Wenn die Daten korrekt aus der INI-Datei gelesen werden konnten, dann  //
   // den entsprechenden Device-Context erzeugen                             //
   //------------------------------------------------------------------------//
   if (NULL != (szDevice = strtok (szPrinter, "," )) &&
       NULL != (szDriver = strtok (NULL,      ", )) &&
       NULL != (szOutput = strtok (NULL,      ", )))
   {
      return CreateDC (szDriver, szDevice, szOutput, NULL);
   }
   return (0);
}

//------------------------------------------------------------------------//
// Aufrufe an da GDI auf der Drucker schicken                             //
//------------------------------------------------------------------------//
void fnWMPaint (HDC hdcPrn, int cxPage, int cyPage)
{
   static char szTextStr [] = "Hello, Printer!";
   HFONT         hFont, hoFont;

   //------------------------------------------------------------------------//
   // Statt auf dem Bildschirm zeichnen wir einfach auf dem Drucker          //
   //------------------------------------------------------------------------//
   Rectangle (hdcPrn, 0, 0, cxPage, cyPage);

   MoveToEx (hdcPrn, 0, 0, NULL);
   LineTo   (hdcPrn, cxPage, cyPage);
   MoveToEx (hdcPrn, cxPage, 0, NULL);
   LineTo   (hdcPrn, 0, cyPage);

   //------------------------------------------------------------------------//
   // Die aktuellen Einstellungen für den Druckerkontext sichern             //
   //------------------------------------------------------------------------//
   SaveDC (hdcPrn);

   //------------------------------------------------------------------------//
   // Skalierungsverfahren für die Druckausgabe ändern                       //
   //------------------------------------------------------------------------//
   SetMapMode (hdcPrn, MM_ISOTROPIC);

   //------------------------------------------------------------------------//
   // "Fenstergröße" für die Ausgabe festlegen                               //
   //------------------------------------------------------------------------//
   SetWindowExtEx   (hdcPrn, 1000, 1000, NULL);
   SetViewportExtEx (hdcPrn, cxPage / 2, -cyPage / 2, NULL);
   SetViewportOrgEx (hdcPrn, cxPage / 2,  cyPage / 2, NULL);

   //------------------------------------------------------------------------//
   // Ausgaben                                                               //
   //------------------------------------------------------------------------//
   Ellipse (hdcPrn, -500, 500, 500, -500);
   SetTextAlign (hdcPrn, TA_BASELINE | TA_CENTER);

   //------------------------------------------------------------------------//
   // Einen True-Type-Font in gewünschter Ausprägung erzeugen                //
   //------------------------------------------------------------------------//
   hFont  = CreateFont (120, 0, 0, 0, FW_THIN, TRUE, FALSE, FALSE,
                        ANSI_CHARSET, OUT_DEFAULT_PRECIS,
                        CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                        DEFAULT_PITCH, "Braggadocio");
   hoFont = (HFONT)SelectObject (hdcPrn, hFont);
   TextOut (hdcPrn, 0, 0, szTextStr, strlen (szTextStr));
   SelectObject (hdcPrn, hoFont);
   DeleteObject (hFont);

   //------------------------------------------------------------------------//
   // Einstellungen des Device-Contextes wiederherstellen                    //
   //------------------------------------------------------------------------//
   RestoreDC (hdcPrn, -1);
}

//------------------------------------------------------------------------//
// Handlerfunktion der Client-Area                                        //
//------------------------------------------------------------------------//
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   HDC          hdc;
   HMENU        hMenu;
   PAINTSTRUCT  ps;
```

```
   switch (message)
   {
      //------------------------------------------------------------------------//
      // Beim Erzeugen der Client-Area (Initialisierung)                        //
      //------------------------------------------------------------------------//
      case WM_CREATE:
           hMenu = GetSystemMenu (hwnd, FALSE);
           AppendMenu (hMenu, MF_SEPARATOR, 0, NULL);
           AppendMenu (hMenu, 0, 1, "&Drucken");
           return (0);

      //------------------------------------------------------------------------//
      // Bei Veränderung der Größe des Fensters                                 //
      //------------------------------------------------------------------------//
      case WM_SIZE:
           cxClient = LOWORD (lParam);
           cyClient = HIWORD (lParam);
           return (0);

      //------------------------------------------------------------------------//
      // Bei Auswahl im System-Menü                                             //
      //------------------------------------------------------------------------//
      case WM_SYSCOMMAND:
           if (wParam == 1)
           {
              if (PrintMyPage (hwnd))
              {
                 MessageBox (hwnd, "Fehler beim Drucken der Seite!",
                             szWindowclassname, MB_OK | MB_ICONEXCLAMATION);
              }
              return (0);
           }
           break;

      //------------------------------------------------------------------------//
      // Beim Neuzeichnen des Fensters                                          //
      //------------------------------------------------------------------------//
      case WM_PAINT:
           //-------------------------------------------------------------------//
           // Aufruf der Mal-Funktion "fnWMPaint" mit dem Display-Context       //
           // mit Angabe der Bildschirmauflösung                                //
           //-------------------------------------------------------------------//
           hdc = BeginPaint (hwnd, &ps);
           fnWMPaint (hdc, cxClient, cyClient);
           EndPaint (hwnd, &ps);
           return (0);

      //------------------------------------------------------------------------//
      // Beim Verlassen des Programms                                           //
      //------------------------------------------------------------------------//
      case WM_DESTROY:
           PostQuitMessage (0);
           return (0);
   }
   return DefWindowProc (hwnd, message, wParam, lParam);
}

//---------------------------------------------------------------------------//
// Ausgelagerte "Malfunktion" für den Drucker, ruft genau wie WM_PAINT die   //
// Funktion fnWMPaint auf                                                    //
//---------------------------------------------------------------------------//
#pragma argsused
BOOL PrintMyPage (HWND hwnd)
{
   HDC hdcPrn;
   DOCINFO     di;
   int         xPage, yPage;

   //-------------------------------------------------------------------------//
   // Device-Context des Druckers holen, wenn keiner verfügbar ist, dann mit  //
   // Fehler zurück                                                           //
   //-------------------------------------------------------------------------//
   hdcPrn = GetPrinterDC ();
   if (NULL == hdcPrn) return TRUE ;

   //-------------------------------------------------------------------------//
   // Auflösung des Druckers ermitteln                                        //
   //-------------------------------------------------------------------------//
   xPage = GetDeviceCaps (hdcPrn, HORZRES) ;
   yPage = GetDeviceCaps (hdcPrn, VERTRES) ;

   //-------------------------------------------------------------------------//
   // DocInfo-Struktur für die Drucker-Queue ausfüllen                        //
```

```
   //------------------------------------------------------------------------//
   ZeroMemory (&di, sizeof (DOCINFO));
   di.cbSize = sizeof (DOCINFO);
   di.lpszDocName = "TestDoc";

   //------------------------------------------------------------------------//
   // Druckvorgang mit neuer Seite starten                                   //
   //------------------------------------------------------------------------//
   StartDoc (hdcPrn, &di);
   StartPage (hdcPrn);

   //------------------------------------------------------------------------//
   // Hier mach wir das gleiche, wie auf dem Bildschirm...                   //
   //------------------------------------------------------------------------//
   fnWMPaint (hdcPrn, xPage, yPage);

   //------------------------------------------------------------------------//
   // Seite und Druckvorgang beenden, Device-Context wieder freigeben        //
   //------------------------------------------------------------------------//
   EndPage (hdcPrn);
   EndDoc  (hdcPrn);
   DeleteDC(hdcPrn);

   return (0);
}
```

8.19 DLL Demo

Das folgende Programm besteht aus mehreren teilprojekten und demonstriert das dynamische Einbinden von DLLs.

8.19.1 DLL Demo – Hauptprogramm

```
//#############################################################################
// P140.CPP
//#############################################################################

#include <windows.h>
#include <dir.h>
#include <dos.h>
#include <stdio.h>
#include <string.h>

#include "P140_Plugin.h"

//=============================================================================
// Defines aus dem Resource-Workshop einlesen, so daß sie hier verwendbar sind =
//=============================================================================
#include "p140.rh"

//-----------------------------------------------------------------------------
// Die folgenden Zeilen sind nur für den Borland C++ Builder interessant, um die
// von Hand erstellten Resourcen korrekt zu laden ...
//-----------------------------------------------------------------------------
#ifdef __BORLANDC__
 #if __BORLANDC__ > 0x0502
   #include <condefs.h>
   USERC("p140.rc");
//-----------------------------------------------------------------------------
 #endif
#endif

#define BEARBEITET   0
#define UNBEARBEITET 1

//=============================================================================
// Prototypen                                                                 =
//=============================================================================
LRESULT CALLBACK WndProc (HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam);

LRESULT fnCreate         (HWND hwnd, LPCREATESTRUCT lpCS);
LRESULT fnShowWindow     (HWND hwnd, BOOL fShow, int fnStatus);
LRESULT fnSize           (HWND hwnd, WPARAM fwSizeType, WORD wWidth, WORD wHeight);
LRESULT fnCommand        (HWND hwnd, WORD wNotifyCode, WORD wID, HWND hwndCtl);
LRESULT fnDestroy        (HWND hwnd);

LRESULT fnCommand_Button1  (HWND hwnd);
LRESULT fnCommand_ListBox1 (HWND hwnd);

//=============================================================================
// globale Variablen                                                          =
//=============================================================================
HINSTANCE hInst;
HWND      hWindow;

//=============================================================================
// Hauptprogramm                                                              =
//=============================================================================
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine,
                    int nCmdShow)
{
   static char szWindowclassname[] = "Pdll";
   MSG         aMessage;
   WNDCLASSEX  aWindowclass;

   //---------------------------------------------------------------------------
   // Die zur Laufzeit erzeugten Dialogboxen benötigen für ihre eigene Ver-    -
   // waltung den Instanzenzähler. Dazu muß dieser global gepuffert werden     -
   //---------------------------------------------------------------------------
   hInst = hInstance;

   if (!hPrevInstance)
   {
      aWindowclass.cbSize        = sizeof (aWindowclass);
```

```
      aWindowclass.style         = CS_HREDRAW | CS_VREDRAW;
      aWindowclass.lpfnWndProc   = WndProc;
      aWindowclass.cbClsExtra    = 0;
      aWindowclass.cbWndExtra    = DLGWINDOWEXTRA;
      aWindowclass.hInstance     = hInst;
      aWindowclass.hIcon         = LoadIcon (hInst, MAKEINTRESOURCE(IDI_INFO));
      aWindowclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
      aWindowclass.hbrBackground = (HBRUSH)GetStockObject (LTGRAY_BRUSH);
      aWindowclass.lpszMenuName  = NULL;
      aWindowclass.lpszClassName = szWindowclassname;
       aWindowclass.hIconSm       = LoadIcon (hInst, MAKEINTRESOURCE(IDI_INFO));

      RegisterClassEx (&aWindowclass);
   }

   //---------------------------------------------------------------------------
   // Client-Area-Erzeugung
   //---------------------------------------------------------------------------
   hWindow = CreateWindowEx (
                NULL,              // Extended Windowstyles
                szWindowclassname,   // mit RegisterClassEx registrierter Name
                "SelectDll",         // Text in der Titelzeile
                WS_OVERLAPPEDWINDOW, // Windowstyles
                100,                 // Horizontale Position
                100,                 // Vertikale Position
                140,                 // Horizontale Ausdehnung (Breite)
                120,                 // Vertikale Ausdehnung   (Höhe)
                NULL,                // Übergeordnetes Fenster
                NULL,                // Menü-Handle
                hInstance,           // Instancehandle von Windows
                NULL);               // Pointer auf Zusatzdaten

   if (hWindow)
   {
      ShowWindow (hWindow, nCmdShow);
      UpdateWindow (hWindow);

      while (GetMessage (&aMessage, NULL, 0, 0))
      {
         TranslateMessage (&aMessage);
         DispatchMessage (&aMessage);
      }
   }
   else
   {
      MessageBeep (0);
   }
   return (aMessage.wParam);
}

//===========================================================================
// Handlerfunction der Client-Area                                          =
//===========================================================================
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  LRESULT RC = TRUE; // TRUE = MEssage muß noch bearbeitet werden

   switch (message)
   {
      case WM_CREATE:     RC = fnCreate (hwnd, (LPCREATESTRUCT) lParam);
                          break;
      case WM_SHOWWINDOW: RC = fnShowWindow (hwnd, (BOOL)wParam, (int)lParam);
                          break;
      case  WM_COMMAND:       RC  =  fnCommand  (hwnd,  HIWORD(wParam),  LOWORD(wParam),
(HWND) lParam);      // handle of control
                          break;
      case WM_DESTROY:    RC = fnDestroy (hwnd);
                          break;
      case WM_SIZE:       RC = fnSize (hwnd, wParam, LOWORD(lParam), HIWORD(lParam));
                          break;
   }
   if (RC) RC = DefWindowProc (hwnd, message, wParam, lParam);
   return RC;
}

//===========================================================================
// WM_COMMAND Message                                                       =
//===========================================================================
#pragma argsused
LRESULT fnCommand (HWND hwnd, WORD wNotifyCode, WORD wID, HWND hwndCtl)
{
   LRESULT RC = UNBEARBEITET;
```

```
   switch (wID)
   {
      case IDC_BUTTON1:
           RC = fnCommand_Button1 (hwnd);
           break;
   }
   return RC; // nicht verarbeitet
}

//============================================================================
// WM_CREATE Message                                                        =
//============================================================================
#pragma argsused
LRESULT fnCreate (HWND hwnd, LPCREATESTRUCT lpCS)
{
   //--------------------------------------------------------------------------
   // hier Initialisierung der Dialogelemente auf der Client-Area:           -
   // xy-Koordinaten und Größen unnötig, da unter WM_SIZE abgehandelt        -
   //--------------------------------------------------------------------------
   CreateWindow ("listbox", "",
                 LBS_STANDARD | WS_CHILD | WS_VISIBLE | WS_TABSTOP,
                 0, 0, 0, 0,
                 hwnd, (HMENU)IDC_LISTBOX1, (HINSTANCE)lpCS->hInstance, NULL);

   CreateWindow ("button", "Aufrufen",
                 BS_PUSHBUTTON | BS_CENTER | WS_CHILD | WS_VISIBLE | WS_TABSTOP,
                 0, 0, 0, 0,
                 hwnd, (HMENU)IDC_BUTTON1, (HINSTANCE)lpCS->hInstance, NULL);

   return BEARBEITET;
}

//============================================================================
// WM_DESTROY Message                                                       =
//============================================================================
#pragma argsused
LRESULT fnDestroy (HWND hwnd)
{
   PostQuitMessage (0);
   return BEARBEITET;
}

//============================================================================
// WM_SHOWWINDOW Message                                                    =
//============================================================================
#pragma argsused
LRESULT fnShowWindow (HWND hwnd, BOOL fShow, int fnStatus)
{
   fnCommand_ListBox1 (hwnd);
   return BEARBEITET;
}

//============================================================================
// WM_SIZE-Message                                                          =
//============================================================================
#pragma argsused
LRESULT fnSize (HWND hwnd, WPARAM fwSizeType, WORD wWidth, WORD wHeight)
{
   RECT aR;

   GetClientRect (hwnd, &aR);
   MoveWindow    (GetDlgItem (hwnd, IDC_LISTBOX1),
                  aR.left+2, aR.top+2, aR.right-2, aR.bottom-22, TRUE);
   MoveWindow    (GetDlgItem (hwnd, IDC_BUTTON1),
                  aR.left+2, aR.bottom-20, aR.right-2, 18, TRUE);
   return BEARBEITET;
}

//============================================================================
// Aufruf einer Funktion in der ausgewählten DLL                            -
//============================================================================
#pragma argsused
LRESULT fnCommand_Button1 (HWND hwnd)
{
   char      szDLLPfad [MAXPATH]; // Namensspeicher für die DLL incl. Pfad     -
   char      szFileLB  [MAXPATH]; // Namensspeicher für die DLL                -
   HINSTANCE hInstDLL;            // Instanzen-Handle für die DLL              -
   BOOL      bRet = TRUE;         // Returnwert                                -
   LRESULT   nIdx = LB_ERR;       // Index des in Listbox selekt. Eintrags     -
   char     *pszHelp;             // Hilfspointer f. Zeichenkettenmanipulation -
   HWND      hLB;                 // Pufferspeicher für Listbox-Handle         -

   PlugIn *(FAR *lpfn) (void);  // Ptr auf eine in DLL enthaltener Funktion  -
```

```
PlugIn *test = NULL;         // Pointer um Returnwert aufzunehmen         -

//---------------------------------------------------------------------------
// Handle der Listbox ermitteln                                            -
//---------------------------------------------------------------------------
hLB = GetDlgItem (hwnd, IDC_LISTBOX1);

//---------------------------------------------------------------------------
// Namen der DLL aus der Listbox ermitteln und kompletten Pfadnamen bilden  -
// dazu zunächsteinmal den Startpfad des Hauptprogramms ermitteln           -
//---------------------------------------------------------------------------
strcpy (szDLLPfad, _argv[0]);
pszHelp = strrchr (szDLLPfad, '\\');
if (pszHelp)
{
   pszHelp++;
   *pszHelp = '\0';
}

//---------------------------------------------------------------------------
// Namen der selektierten DLL holen und mit Pfad zusammenbauen              -
//---------------------------------------------------------------------------
nIdx = SendMessage (hLB, LB_GETCURSEL, (WPARAM)0, (LPARAM)0);
if (LB_ERR == nIdx)
{
   bRet = FALSE;
     MessageBox (NULL, "Kein Name in der Listbox gewählt", "SELECTDLL",
               MB_OK|MB_TASKMODAL|MB_ICONSTOP);
}
else
{
   SendMessage (hLB, LB_GETTEXT, (WPARAM)nIdx, (LPARAM)szFileLB);
   strcat (szDLLPfad, szFileLB);
}

//---------------------------------------------------------------------------
// Wenn der DLL-Pfad erfolgreich gebildet werden konnte, dann DLL laden     -
//---------------------------------------------------------------------------
if (bRet)
{
   //------------------------------------------------------------------------
   // DLL dynamisch nachladen                                               -
   //------------------------------------------------------------------------
   hInstDLL = LoadLibrary (szDLLPfad);

   //------------------------------------------------------------------------
   // Konnte DLL geladen werden ?                                           -
   //------------------------------------------------------------------------
   if (!hInstDLL)
   {
      bRet = FALSE;
      MessageBox (NULL, szDLLPfad, "DLL konnte nicht geladen werden",
                  MB_OK|MB_TASKMODAL|MB_ICONSTOP);
   }
   else
   {
      //---------------------------------------------------------------------
      // DLL konnte geladen werden. Jetzt die Adresse d. gewünschten Funktion
      // holen. Diese muß dazu in der EXPORTS-Section der DEF-Datei der DLL -
      // vermerkt sein.                                                     -
      //---------------------------------------------------------------------
      (FARPROC)lpfn = GetProcAddress (hInstDLL, "_GetDLLPlugInObject");

      //---------------------------------------------------------------------
      // Konnte Funktionsadresse geladen werden ?                           -
      //---------------------------------------------------------------------
      if (NULL == lpfn)
      {
         bRet = FALSE;
         MessageBox (NULL, szDLLPfad, "Funktion konnte nicht geladen werden",
                     MB_OK|MB_TASKMODAL|MB_ICONSTOP);
      }
      else
      {
         //------------------------------------------------------------------
         // Aufruf der geladenen Funktionaddresse. Die Funktion liefert     -
         // den Zeiger auf eine in der DLL vorhandene Klasse zurück         -
         // ist der Aufruf nicht erfolgreich, so wird NULL zurückgegeben    -
         //------------------------------------------------------------------
         test = (*lpfn) ();
         if (test)
         {
            test->ShowName();
```

```
            }
         }

         //---------------------------------------------------------------------
         // Wenn die DLL nicht mehr gebraucht wird, wieder entladen            -
         //---------------------------------------------------------------------
         FreeLibrary (hInstDLL);
      }
   }

   return (bRet);
}

//============================================================================
// Füllen einer Listbox mit Dateinamen gemäß Namensmuster                     -
//============================================================================
#pragma argsused
LRESULT fnCommand_ListBox1 (HWND hwnd)
{
   char          szDLLPfad [MAXPATH]; // Namensspeicher für die DLL incl. Pfad -
   char         *pszHelp;
   struct ffblk  rFileBlock;
   int           bFailed;
   LRESULT       nAnzahl, RC = FALSE;
   HWND          hLB;

   //---------------------------------------------------------------------------
   // Handle der Listbox ermitteln                                            -
   //---------------------------------------------------------------------------
   hLB = GetDlgItem (hwnd, IDC_LISTBOX1);

   //---------------------------------------------------------------------------
   // Namensmuster der DLLs für die Listbox ermitteln und in Listbox stellen   -
   //---------------------------------------------------------------------------
   strcpy (szDLLPfad, _argv[0]);
   pszHelp = strrchr (szDLLPfad, '\\');
   if (pszHelp)
   {
      pszHelp++;
      *pszHelp = '\0';
      lstrcat (szDLLPfad, "P140_*.dll");
   }

   //---------------------------------------------------------------------------
   // gemäß Namensmuster der DLL das Verzeichnis durchsuchen                  -
   //---------------------------------------------------------------------------
   bFailed = findfirst (szDLLPfad, &rFileBlock, 0);
   while (!bFailed)
   {
      SendMessage (hLB, LB_ADDSTRING, (WPARAM)0, (LPARAM)rFileBlock.ff_name);
      bFailed = findnext (&rFileBlock);
   }

   //---------------------------------------------------------------------------
   // Anzahl der gefundenen DLLs prüfen                                       -
   //---------------------------------------------------------------------------
   nAnzahl = SendMessage (hLB, LB_GETCOUNT, (WPARAM)0, (LPARAM)0);

   if ((nAnzahl > 0) && (LB_ERR != nAnzahl))
   {
      // ersten in Listbox selektieren
      SendMessage (hLB, LB_SETCURSEL, (WPARAM)0, (LPARAM)0);
      RC = TRUE;
   }
   return RC;
}
```

8.19.2 DLL Demo – Hauptprogramm Resource

```
//############################################################################
// P140.RC
//############################################################################

#include "P140.rh"

IDI_INFO ICON
{
 '00 00 01 00 01 00 20 20 10 00 00 00 00 00 E8 02'
 '00 00 16 00 00 00 28 00 00 00 20 00 00 00 40 00'
 '00 00 01 00 04 00 00 00 00 00 80 02 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 80 00 00 80 00 00 00 80 80 00 80 00'
```

```
 '00 00 80 00 80 00 80 80 00 00 C0 C0 C0 00 80 80'
 '80 00 00 00 FF 00 00 FF 00 00 00 FF FF 00 FF 00'
 '00 00 FF 00 FF 00 FF FF 00 00 FF FF FF 00 7F FF'
 'FF FF FF FF FF FF FF FF FF FF FF FF FF 87 88'
 '88 88 88 88 88 88 88 88 88 88 88 88 88 8F 8F 77'
 '77 77 77 77 77 77 77 77 77 77 77 77 77 8F 8F 77'
 '77 77 77 77 77 77 77 77 77 77 77 77 77 8F 8F 77'
 '77 77 77 77 99 99 99 99 77 77 77 77 77 8F 8F 77'
 '77 77 77 99 99 99 99 99 99 77 77 77 77 8F 8F 77'
 '77 77 79 99 99 99 99 99 99 97 77 77 77 8F 8F 77'
 '77 77 99 99 99 94 44 44 49 99 99 77 77 77 8F 8F 77'
 '77 79 99 94 44 44 44 44 49 99 97 77 77 8F 8F 77'
 '77 99 99 94 44 44 44 44 49 99 99 77 77 8F 8F 77'
 '79 99 99 99 44 44 44 44 99 99 99 97 77 8F 8F 77'
 '79 99 44 99 94 44 44 49 99 44 99 97 77 8F 8F 77'
 '99 99 44 49 99 44 44 99 94 44 99 99 77 8F 8F 77'
 '99 94 44 44 99 94 49 99 44 44 49 99 77 8F 8F 77'
 '99 94 44 44 49 99 99 94 44 44 49 99 77 8F 8F 77'
 '99 94 44 44 44 99 99 44 44 44 49 99 77 8F 8F 77'
 '99 94 44 44 44 99 99 44 44 44 49 99 77 8F 8F 77'
 '99 94 44 44 49 99 99 94 44 44 49 99 77 8F 8F 77'
 '99 94 44 44 99 94 49 99 44 44 49 99 77 8F 8F 77'
 '99 99 44 49 99 44 44 99 94 44 99 99 77 8F 8F 77'
 '79 99 44 99 94 44 44 49 99 44 99 97 77 8F 8F 77'
 '79 99 99 99 44 44 44 44 99 99 99 97 77 8F 8F 77'
 '77 99 99 94 44 44 44 44 49 99 99 77 77 8F 8F 77'
 '77 79 99 94 44 44 44 44 49 99 97 77 77 8F 8F 77'
 '77 77 99 99 94 44 44 49 99 99 77 77 77 8F 8F 77'
 '77 77 79 99 99 99 99 99 99 97 77 77 77 8F 8F 77'
 '77 77 77 99 99 99 99 99 99 77 77 77 77 8F 8F 77'
 '77 77 77 77 99 99 99 99 77 77 77 77 77 8F 8F 77'
 '77 77 77 77 77 77 77 77 77 77 77 77 77 8F 8F 77'
 '77 77 77 77 77 77 77 77 77 77 77 77 77 8F 8F FF'
 'FF FF FF FF FF FF FF FF FF FF FF FF FF 7F 88 88'
 '88 88 88 88 88 88 88 88 88 88 88 88 88 87 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
 '00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00'
}

IDD_ABOUT DIALOG 24, 21, 188, 138
STYLE DS_SYSMODAL | WS_POPUP | WS_VISIBLE | WS_CAPTION
CLASS "BorDlg_Gray"
CAPTION "Remover ... V 2.20"
FONT 8, "MS Sans Serif"
{
 CONTROL "(C) Copyright 1998 by Heiko Gorski, Norderstedt, Germany\n\ne-mail:
Gorski@T-Online.de", -1, "STATIC", SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 42,
18, 132, 57
 CONTROL "&OK", IDOK, "BUTTON", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP,
78, 108, 60, 17
 CONTROL "Text", IDS_T1, "STATIC", SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 60,
81, 96, 8
 CONTROL "Text", IDS_T2, "STATIC", SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 60,
93, 96, 8
 CONTROL 1, -1, "STATIC", SS_ICON | WS_CHILD | WS_VISIBLE, 9, 15, 18, 20
 CONTROL "", IDC_GROUPBOX3, "BUTTON", BS_GROUPBOX | WS_CHILD | WS_VISIBLE, 33, 6,
147, 123
}
```

8.19.3 DLL Demo – Hauptprogramm Resource Header

```
//########################################################################################
// P140.RH
//########################################################################################

#define IDC_LISTBOX1   101
#define IDC_BUTTON1   102
#define IDC_BORSTATIC1   103
#define IDD_ABOUT   2
#define IDI_INFO   1

#define IDS_T1          602
#define IDS_T2          603
#define IDC_GROUPBOX3   504
#define ST_MODULETITLE   505
#define ST_ENTRYTITLE   506
```

8.19.4 DLL Demo –DLL gemeinsamer Header

```
#ifndef _P140_PLUGIN_H_
#define _P140_PLUGIN_H_

class PlugIn
{
   public:
      virtual void ShowName (void) = 0;
};

#endif
```

8.19.5 DLL Demo – erste DLL Header

```
#ifndef _P140DLL1_PLUGIN_H_
#define _P140DLL1_PLUGIN_H_

#include "P140_PlugIn.h"

#ifdef EXPORT_P140DLL1
   #define P140DLL1 __declspec(dllexport)
#else
   #define P140DLL1 __declspec(dllimport)
#endif

class P140DLL1 DLL1_PlugIn : public PlugIn
{
   public:
        virtual void ShowName (void);
};

#endif
```

8.19.6 DLL Demo – zweite DLL Header

```
#ifndef _P140DLL2_PLUGIN_H_
#define _P140DLL2_PLUGIN_H_

#include "P140_PlugIn.h"

#ifdef EXPORT_P140DLL2
   #define P140DLL2 __declspec(dllexport)
#else
   #define P140DLL2 __declspec(dllimport)
#endif

class P140DLL2 DLL2_PlugIn : public PlugIn
{
   public:
        virtual void ShowName (void);
};

#endif
```

8.19.7 DLL Demo – dritte DLL Header

```
#ifndef _P140DLL3_PLUGIN_H_
#define _P140DLL3_PLUGIN_H_

#include "P140_PlugIn.h"

#ifdef EXPORT_P140DLL3
   #define P140DLL3 __declspec(dllexport)
#else
   #define P140DLL3 __declspec(dllimport)
#endif

class P140DLL3 DLL3_PlugIn : public PlugIn
{
   public:
        virtual void ShowName (void);
};

#endif
```

8.19.8 DLL Demo – erste DLL

```
#include <windows.h>
#include <strstrea.h>

//============================================================================
// FUNKTIONSEXPORT DER DLL                                                   =
//============================================================================
#define EXPORT_P140DLL1
#include "P140DLL1_Plugin.h"

//============================================================================
// GLOBALE VARIABLEN DER DLL                                                 =
//============================================================================
#define ALLOCANZAHL 100
HINSTANCE   hLibInst;
DLL1_PlugIn myDllObject;
int         *x [ALLOCANZAHL];

//============================================================================
// PROTOTYPEN                                                                =
//============================================================================
void InitDLL (void);

//============================================================================
// DLL HAUPTPROGRAMM                                                         =
//============================================================================
#pragma argsused
int WINAPI DllEntryPoint (HINSTANCE hinst, unsigned long reason, void*)
{
   switch (reason)
   {
      case DLL_PROCESS_ATTACH: // Aufruf durch LoadLibrary
                               hLibInst = hinst; // Handle der DLL sichern
                               InitDLL ();
                               break;
      case DLL_PROCESS_DETACH: // Aufruf durch FreeLibrary
                               break;
      case DLL_THREAD_ATTACH:  break;
      case DLL_THREAD_DETACH:  break;
   }
   return 1;
}

//============================================================================
// INITDALL                                                                  =
//============================================================================
#pragma argsused
void InitDLL (void)
{
   strstream szInfo;
   int i    = 0;
   int ende = 0;

   while (i<ALLOCANZAHL && !ende)
   {
      x[i] = new int [30000];
      if (!x[i]) ende = 1;
      i++;
   }

   szInfo << i << " x 60kB angelegt";
   MessageBox (NULL, szInfo.str(), "dyn. Allokation", MB_OK);
}

//----------------------------------------------------------------------------
// GETDLLPLUGINOBJECT                                                        -
//----------------------------------------------------------------------------
extern "C" PlugIn * FAR _export GetDLLPlugInObject (void)
{
   int i=0;
   bool bMsg = false;

   for (i=0; i<ALLOCANZAHL; i++)
   {
      if (x[i])
      {
         delete [] x[i];
          x[i] = NULL;
         bMsg = true;
      }
   }
```

```
     if (bMsg)
   {
      MessageBox (NULL, "alles freigegeben", "x", MB_OK);
   }

   return (&myDllObject);
}

//---------------------------------------------------------------------------
// METHODEN DES DLL-Objektes                                                -
//---------------------------------------------------------------------------
void DLL1_PlugIn::ShowName (void)
{
      MessageBox (NULL, "P140_DLL_1", "Aufruf des Objekts", MB_OK);
}
```

8.19.9 DLL Demo – zweite DLL

```
#include <windows.h>

//===========================================================================
// FUNKTIONSEXPORT DER DLL                                                  =
//===========================================================================
#define EXPORT_P140DLL2
#include "P140DLL2_Plugin.h"

//===========================================================================
// GLOBALE VARIABLEN DER DLL                                                =
//===========================================================================
HINSTANCE   hLibInst;
DLL2_PlugIn myDllObject;

//===========================================================================
// PROTOTYPEN                                                               =
//===========================================================================
void InitDLL (void);

//===========================================================================
// DLL HAUPTPROGRAMM                                                        =
//===========================================================================
#pragma argsused
int WINAPI DllEntryPoint (HINSTANCE hinst, unsigned long reason, void*)
{
   switch (reason)
   {
      case DLL_PROCESS_ATTACH: // Aufruf durch LoadLibrary
                               hLibInst = hinst; // Handle der DLL sichern
                               InitDLL ();
                               break;
      case DLL_PROCESS_DETACH: // Aufruf durch FreeLibrary
                               break;
      case DLL_THREAD_ATTACH:  break;
      case DLL_THREAD_DETACH:  break;
   }
   return 1;
}

//===========================================================================
// INITDALL                                                                 =
//===========================================================================
#pragma argsused
void InitDLL (void)
{
   MessageBox (NULL, "InitDLL", "P140_DLL_2", MB_OK);
}

//---------------------------------------------------------------------------
// GETDLLPLUGINOBJECT                                                       -
//---------------------------------------------------------------------------
extern "C" PlugIn * FAR _export GetDLLPlugInObject (void)
{
   return (&myDllObject);
}

//---------------------------------------------------------------------------
// METHODEN DES DLL-Objektes                                                -
//---------------------------------------------------------------------------
void DLL2_PlugIn::ShowName (void)
{
      MessageBox (NULL, "P140_DLL_2", "Aufruf des Objekts", MB_OK);
}
```

8.19.10 DLL Demo – dritte DLL

```
#include <windows.h>

//=============================================================================
// FUNKTIONSEXPORT DER DLL                                                    =
//=============================================================================
#define EXPORT_P140DLL3
#include "P140DLL3_Plugin.h"

//=============================================================================
// GLOBALE VARIABLEN DER DLL                                                  =
//=============================================================================
HINSTANCE   hLibInst;
DLL3_PlugIn myDllObject;

//=============================================================================
// PROTOTYPEN                                                                 =
//=============================================================================
void InitDLL (void);

//=============================================================================
// DLL HAUPTPROGRAMM                                                          =
//=============================================================================
#pragma argsused
int WINAPI DllEntryPoint (HINSTANCE hinst, unsigned long reason, void*)
{
   switch (reason)
   {
      case DLL_PROCESS_ATTACH: // Aufruf durch LoadLibrary
                               hLibInst = hinst; // Handle der DLL sichern
                               InitDLL ();
                               break;
      case DLL_PROCESS_DETACH: // Aufruf durch FreeLibrary
                               break;
      case DLL_THREAD_ATTACH:  break;
      case DLL_THREAD_DETACH:  break;
   }
   return 1;
}

//=============================================================================
// INITDALL                                                                   =
//=============================================================================
#pragma argsused
void InitDLL (void)
{
}

//-----------------------------------------------------------------------------
// GETDLLPLUGINOBJECT                                                         -
//-----------------------------------------------------------------------------
extern "C" PlugIn * FAR _export GetDLLPlugInObject (void)
{
   return (&myDllObject);
}

//-----------------------------------------------------------------------------
// METHODEN DES DLL-Objektes                                                  -
//-----------------------------------------------------------------------------
void DLL3_PlugIn::ShowName (void)
{
      MessageBox (NULL, "P140_DLL_3", "Aufruf des Objekts", MB_OK);
}
```