

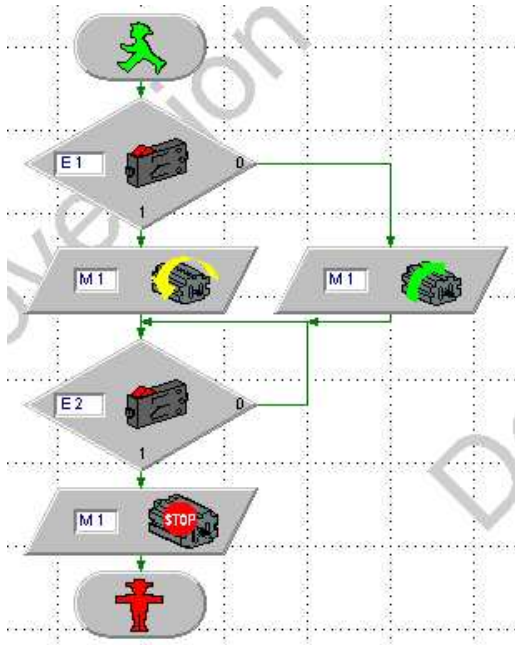
C++ Kurs

**Fachhochschule Pforzheim
Hochschule für Gestaltung, Technik und Wirtschaft
Studiengänge
Elektrotechnik / Informationstechnik
Technische Informatik
Tiefenbronner Str. 65
75175 Pforzheim**

**Tel.: 07231 28-6065
Fax: 07231 28-6060**

Methoden der Programmierung

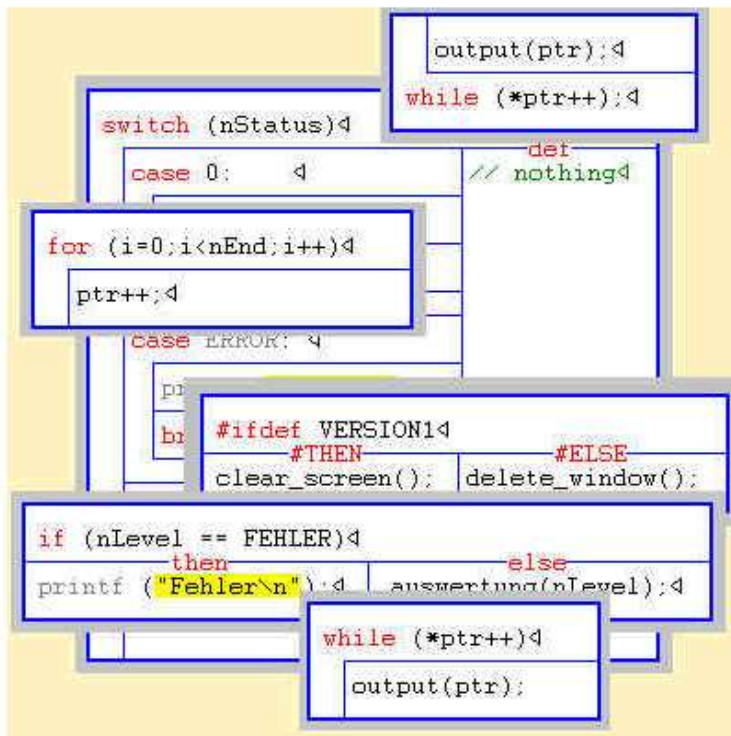
Prozedurale Programmierung



Ein Programm besteht aus einer Folge von auszuführenden Anweisungen, die auf einen Satz von Daten zugreifen.

Entwurfsverfahren: Flußdiagramm

Strukturierte Programmierung



Die Anweisungen werden systematisch in Prozeduren organisiert. Aufgaben werden in kleinere Teilaufgaben zerlegt, die eigenständig sind und sich leichter überblicken lassen. Insbesondere sind aber Sprünge z.B. goto verboten.

Entwurfsverfahren:
 Struktogramm (Nasi-Schneiderman Diagramm)

Erfolgreicher Lösungsansatz bei komplexen Aufgaben speziell bei begrenztem Hauptspeicher.

Problem: Durch die Trennung von Daten und Prozeduren, die Daten manipulieren wird die Programmierung bei zunehmenden Datenmengen unübersichtlicher.

Objektorientierte Programmierung (OOP)

Ein Objekt verfügt über Eigenschaften und Verhaltensweisen, sie werden in einer Klasse definiert. Die Trennung zwischen Daten und Prozeduren ist somit aufgehoben.

4 Säulen:

Kapselung:

Eigenschaften und Verhaltensweisen werden in einem Objekt zusammengefaßt (gekapselt).

Verbergen von Daten:

Nur die für den Nutzer wichtigen Daten und Funktionen sind zugänglich, alle anderen können verborgen werden.

Vererbung und Wiederverwendbarkeit

Es ist möglich ein neues Objekt zu definieren, das eine Erweiterung eines schon bestehenden Objektes darstellt.

Polymorphie

-Vielgestaltig-

Eine Verhaltensweise kann ihre Wirkungsweise ändern – abhängig von den äußeren Ereignissen.

Elemente der Programmiersprache

Anweisungsblock

Alle Anweisungen die in einem Block mit geschweiften Klammern zusammengefasst sind nennt man Anweisungsblock. Innerhalb eines Blockes definierte Variablen verlieren am Blockende ihre Gültigkeit.

```
{  
    cout<<"Dies ";  
    cout<<"ist ein ";  
    cout<<"Anweisungsblock"<<endl;  
}
```

Funktionen

In Funktionen werden Anweisungen zusammengefasst, die aus einem anderen Programmteil aufgerufen werden können.

Die Funktion main() stellt einen Sonderfall dar, da diese Funktion beim Programmstart aufgerufen wird. Jedes Programm darf nur eine main() Funktion enthalten.

Funktionen geben entweder einen Wert oder void (also nichts) zurück. Einer Funktion können Parameter übergeben werden. Der tatsächlich übergebene Wert wird Argument genannt.

Ein- und Ausgabe cin und cout

cout

Funktion zur Ausgabe von Zeichen, Zeichenketten und Werten auf die Standardausgabe.

```
cout<<Variable<<endl; //endl bewirkt einen Zeilenumbruch
```

cin

Funktion zum Einlesen von der Standardeingabe

```
cin<<Variable;
```

Beispiel Funktionen, cout:

```
#include <iostream.h>

void DemoFunction();           //Prototyp der Funktion
int Summe(int, int);

int main()
{
    cout<<"Start main"<<endl;
    DemoFunction();           //Aufruf der Funktion
    cout<<"Wieder in main"<<endl;
    cout<<"2+3="<<Summe(2,3)<<endl;
    return 0;
}

void DemoFunction()           //Funktion mit Funktionsrumpf
{
    cout<<"\tDemoFunction"<<endl; //\t erzeugt einen TAB
}

int Summe (int a, int b)      //Argumente a und b
{
    int c;
    c=a+b;
    return c;                 //Wert zurueckgeben
}
```

Datentypen

einfache Datentypen

Die von einem Programm verwendeten Daten müssen im Speicher (RAM) des Computers abgelegt werden. Der Speicher ist in Byte eingeteilt. Jede Speicherstelle (Byte) erhält eine eigene aufsteigende Adresse. In der Antike des Programmierens konnte nur über die Adresse (absolute Adressierung) auf den Speicher zugegriffen werden. Moderne Sprachen stellen Datentypen zur Verfügung mit deren Hilfe Variablen angelegt werden können. Variablen erleichtern den Zugriff auf den Speicher erheblich.

Je nach Daten- und Rechnertyp belegen die Variablen unterschiedlich viel Platz im Speicher.

Datentyp	Kennung	Größe	Beschreibung
char	c, ch	1	Zeichenwerte -128 bis 127
int	i, n	4	+2.147.483.648
float	f, fl	4	1,2e-38 bis 3,4e38
double	d, dbl	8	2,2e-308 bis 1,8e308
bool	b	1	true oder false

Es ist üblich vor Variablennamen mit einer Kennung zu beginnen. Der eigentliche Name wird groß geschrieben.

Achtung: Kommazahlen werden im Programmcode mit . getrennt also 1.3 geschrieben, dies gilt auch für die Eingabe über die Konsole.

Datenfelder

Es ist möglich mehrer Variablen zu einem Datenfeld zusammenzufassen.

```
int iVektor[10];
```

vereinbart ein eindimensionales Feld (Vektor) vom Datentyp int der Länge 10 , d.h. einen Block von 10 aufeinanderfolgenden benannten Objekten

Zugriff: `iVektor[0]=100;`

String: `char strWort[6];`

vereinbart ein Feld vom Datentyp char der Länge 6 . Es ist geeignet, eine Zeichenkette der maximalen Länge 5 aufzunehmen, da das Element `strWort[5]` zur Aufnahme des Nullzeichens '\0'(also numerisch 0) dient, welches eine Zeichenkette abschließt.

Mit der Funktion `sizeof(Variable Typ)` kann die Größe festgestellt werden.

Beispiel Deklaration von Variablen:

```
char strWort[]="ABC";  
double dZahl;  
char cBuchstabe; //Zuweisung cBuchstabe='a';  
int iSchleifeTest1;
```

Organisation im Speicher

Adresse	RAM	Variablenname
....		
..100		
..101		int iSchleife;
..102		
..103		
..104		
..105		char cBuchstabe;
..106		double dZahl;
..107		
..108		
..109		
..110		
..111		
..112		
..113		
..114	A	char strWort[]="ABC"
..115	B	
..116	C	
..117	0	
..118		
....		

Mathematische Operatoren

Grundrechenarten

+	Addition	$2 + 3 = 7$
-	Subtraktion	$7 - 3 = 4$
*	Multiplikation	$9 * 3 = 27$
/	Division	$9 / 3 = 3$
%	Modulo(nur bei Ganzzahl)	$10 \% 3 = 1$
=	Zuweisungsoperator	

Es gelten die mathematischen Vorrang - und Komma - Regeln.

Achtung beim Rechnen mit ganzen Zahlen.

Ist z.B. eine int Zahl an einer Division beteiligt, so wird diese als Ganzzahldivision ausgeführt.

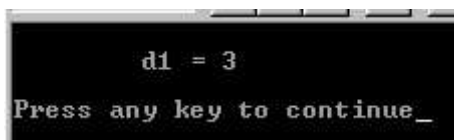
```
#include <iostream.h>

int main()
{
    int i1;
    double d1;

    i1=10;
    d1= i1/3;

    cout<<"\n\td1 = "<<d1<<endl<<endl;
    return 0;
}
```

Das erwartete Ergebnis ist 3.33333333 aber die Rechnung 10/3 wird als Ganzzahldivision durchgeführt – es kommt also 3 heraus.

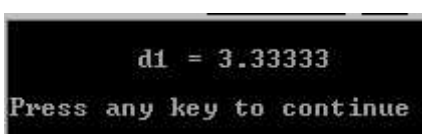


```
d1 = 3
Press any key to continue_
```

Die Rechnung wird erst durch eine Typkonvertierung richtig.

```
d1= (double)i1/3;
```

i1 wird in eine Zahl vom Typ double gewandelt und erst dann wird das Ergebnis berechnet.



```
d1 = 3.33333
Press any key to continue
```

Inkrementieren und Dekrementieren

i1++	i1=i1+1;	
i1--	i1=i1-1;	
i1 += 2;	i1=i1+2;	
i1-=2;	i1=i1-2;	

Präfix: i1=1; i2=++i1; i1 wird um 1 erhöht und dann i2 zugewiesen i2=2 i1=2

Postfix: i1=1; i2=i1++; i1 wird i2 zugewiesen und dann erhöht i2=1 i1=2

Vergleichsoperatoren

Der Datentyp bool enthält die Werte wahr (true) und falsch (false).

Jeder Ausdruck lässt sich als Wahrheitswert auswerten. Nur Ausdrücke die numerisch 0 sind werden als false bewertet alles andere gilt als true (auch -5).

Operator	Symbol	Beispiel	Rückgabewert
Gleich	==	100==50	false
		50==50	true
Ungleich	!=	100!=0	true
		50!=50	false
Größer als	>	100>50	true
		50>100	false
Größer oder gleich	>=	100>=50	true
		50>=50	true
Kleiner	<	100<50	false
		50<100	true
Kleiner oder gleich	<=	100<=50	false
		50<=50	true

Bei Überprüfungen von Gleitkommazahlen kann == aus Gründen der numerischen Ungenauigkeit zu Fehlern führen.

Programmverzweigungen

Die if-Anweisung

Mit der if-Anweisung kann eine Bedingung überprüft werden, um im Programm zu unterschiedlichen Teilen des Codes zu verzweigen.

If-Anweisungen können verschachtelt werden.

if

Der in { } gesetzte Anweisungsblock wird ausgeführt, wenn die Bedingung erfüllt also true ist.

else

Der in { } gesetzte Anweisungsblock wird ausgeführt, wenn die Bedingung nicht erfüllt also false ist. Der else Zweig kann auch weggelassen werden, wenn er nicht erforderlich ist.

```
#include <iostream.h>

int main()
{
    double d1;

    cout<<"Bitte eine Zahl eingeben: ";
    cin>>d1;

    if (d1>=0)
    {
        cout<<d1<<" ist positiv oder 0\n";
    }
    else
    {
        cout<<d1<<" ist negativ\n";
    }
    return 0;
}
```

Soll jeweils nur eine Anweisung nach einer if oder else Verzweigung ausgeführt werden, so kann auf die Klammern { } verzichtet werden.

```
#include <iostream.h>

int main()
{
    double d1;

    cout<<"Bitte eine Zahl eingeben: ";
    cin>>d1;

    if (d1>=0) cout<<d1<<" ist positiv oder 0\n";
    else      cout<<d1<<" ist negativ\n";
    return 0;
}
```

Logische Operatoren

Operator	Symbol	Beschreibung
AND	&&	UND-Verknüpfung
OR		ODER-Verknüpfung
NOT	!	NICHT-Anweisung

X1	X2	X1 && X2	X1 X2	Y= !X2
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	FALSE	TRUE	
FALSE	FALSE	FALSE	FALSE	

Reihenfolge der Ausführung : ! && ||

Beispiel if-Anweisung:

```
#include <iostream.h>

int main()
{
    char c1;

    cout<<"Bitte ein Zeichen eingeben: ";
    cin>>c1;

    if (c1 >= '0' && c1 <= '9')
        cout<<c1<<" ist eine Ziffer\n";
    else
        if ( c1 >= 'a' && c1<= 'z' || c1 == 'A' && c1<= 'Z')
            cout<<c1<<" ist ein Buchstabe des Alphabets\n";
        else
            cout<<c1<<" ist ein Sonderzeichen\n";

    return 0;
}
```

Die switch Anweisung

Mit einer switch Anweisung kann ein Wert ausgewertet werden, es lassen sich aber beliebig viele Verzweigungen in Abhängigkeit von mehreren Werten aufbauen.

switch(Ausdruck)	Definiert den zu prüfenden Ausdruck.
case Wert:	Vergleicht den Ausdruck mit dem Wert und führt die darauf folgenden Anweisungen durch.
break;	Beendet die switch Anweisung.
default:	Gilt für alle bisher nicht vorhandenen Fälle.

Beispiel switch-Anweisung:

```
#include <iostream.h>

int main()
{
    char c1;

    cout<<"Bitte einen Buchstaben eingeben: ";

    cin>>c1;

    switch(c1)
    {
        case 'a':
        case 'b':
        case 'c':
            cout<<" es war a, b oder c\n";
            break;
        case 'A':
        case 'B':
        case 'C':
            cout<<"es war A, B oder C\n";
            break;
        default:
            cout<<"es war ein d - z oder ein D - Y\n";
    }
    return 0;
}
```

Schleifen

Viele Programmieraufgaben lassen sich durch eine wiederholte Ausführung eines Programmteiles lösen.

Es wird zwischen kopf- und fußgesteuerten Schleifen (Wiederholungen) unterschieden.

Kopfgesteuerte Schleifen

Die Bedingung wird zum Beginn der Schleife geprüft. Die while-Schleife wird also solange wiederholt wie die Bedingung TRUE bleibt. Ist die Bedingung von Anfang an FALSE, so werden die Anweisungen in der while-Schleife nicht ausgeführt.

`break;` Die Schleife wird sofort verlassen und das Programm wird nach der Schleife weiter ausgeführt.

`continue;` //Sprung zurück zum Beginn der Schleife

while - Schleife

```
while(Bedingung)
{
    Anweisungen
}
```

Beispiel while-Schleife:

```
#include <iostream.h>

int main()
{
    int iCounter;

    cout<<"Anzahl Ausgabe:";
    cin>>iCounter;

    while(iCounter>0)
    {
        cout<<iCounter<<"\tx Hallo Welt\n";
        iCounter--;
    }
    return 0;
}
```

for - Schleife

Sonderform der kopfgesteuerten Schleife

Die Kopfzeile faßt die drei Schritte Initialisierung, Test und Inkrementierung in einer Anweisung zusammen.

```
for(Startzustand; Bedingung; Veraenderung)
{
    Anweisungen
}
```

Beispiel for - Schleife:

```
#include <iostream.h>

int main()
{
    int iCounter;

    cout<<"Anzahl Ausgabe:";
    cin>>iCounter;

    for(int i=1; i<=iCounter; i++)
    {
        cout<<i<<"\tx Hallo Welt\n";
    }
    return 0;
}
```

Fußgesteuerte Schleifen

Bei der while-Schleife kann es vorkommen, daß die Schleife nicht ausgeführt, da die Bedingung zu Beginn der Schleife geprüft wird.

Die do...while-Schleife wird einmal ausgeführt. Vor der Wiederholung wird die Bedingung geprüft und wenn diese wahr ist wird die Schleife wiederholt.

do while - Schleife

```
do
{
    Anweisungen
} while (Bedingung);
```

Beispiel do while -Schleife:

```
#include <iostream.h>

int main()
{
    char c;
    int i;

    cout<<"weiter mit bel.Taste - Ende mit e\n";
    i=0;

    do
    {
        i++;
        cout<<"Schleife Nr."<<i<<"  Ende ? (e)";
        cin>>c;
    } while(c!='e');
    return 0;
}
```

Direkter Speicherzugriff

Zeiger

Jede Variable belegt im Computer Speicherplatz. Der Prozessor greift über eine Adresse auf die Variable zu. Mit dieser Adresse kann **direkt** auf den Speicherinhalt zugegriffen werden. Um eine Adresse verwenden zu können muß sie ebenfalls im Speicher abgelegt werden. Die Variable die eine solche Adresse enthält wird **Zeiger** genannt.

* legt einen Zeiger an:

```
int *iZeiger;    //legt einen Zeiger auf einen Int-Wert an.
```

Der Zeiger zeigt noch nicht auf einen gültigen Speicherbereich.
Es gibt zwei Möglichkeiten dies zu ändern:

& - Operator

& übergibt die Adresse einer Variablen

```
int *iZeiger;  
int i;  
  
    iZeiger=&i;    //iZeiger zeigt auf den gleichen Speicherplatz der  
                  //von der Variable i belegt wird.
```

new - Operator

Der Operator new reserviert den angeforderten Speicherbereich zur Laufzeit und übergibt die Adresse.

```
iZeiger = new(int); //Speicherbereich für einen Int-Wert  
                  //reservieren und die Adresse an iZeiger  
                  //übergeben.
```

delete - Operator

delete gibt den angeforderten Speicher wieder frei.

```
delete iZeiger;
```

Zeiger zum Schreiben und Lesen verwenden

Werte in den Speicher schreiben:

```
int a;  
a=5;  
*iZeiger=3; //oder auch *iZeiger=a;
```

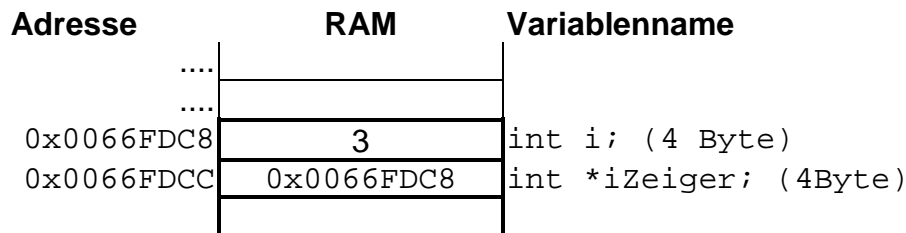
Werte aus dem Speicher lesen:

```
a=*iZeiger;
```

Beispiel: einfache Variablendeklaration:

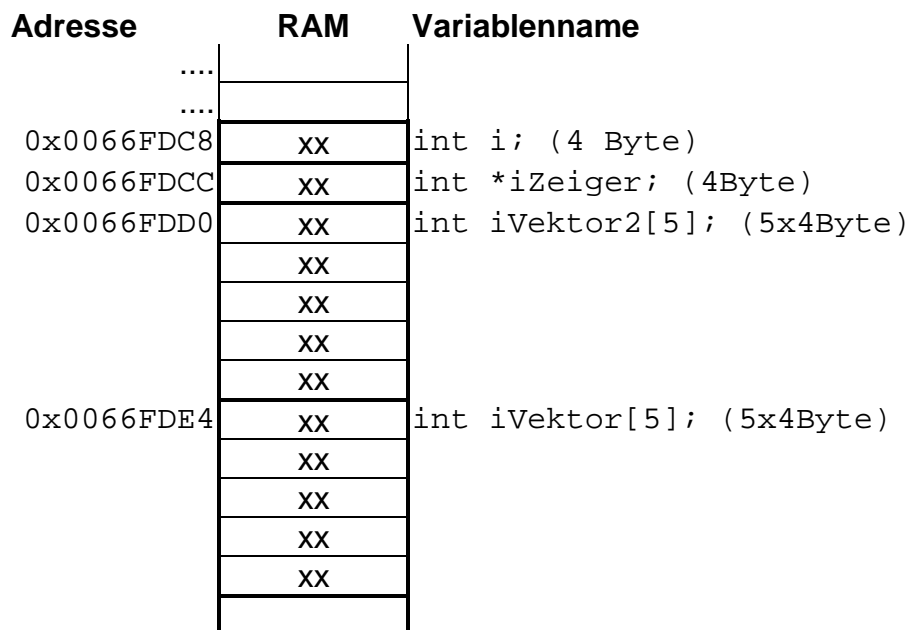
```
int *iZeiger;
int i;

iZeiger=&i;
*iZeiger=3;
```



Beispiel: Variablendeklaration von Datenfeldern:

```
int iVektor[5];
int iVektor2[5];
int *iZeiger;
int i;
```



Speicher nach Ausführung der folgenden Anweisungen:

```

:
for(i=0; i<5; i++) iVektor[i]=i;
for(i=0; i<5; i++) iVektor2[i]=20+i;

:
iZeiger=&iVektor2[0]; //oder auch iZeiger=iVektor2
:

```

Adresse	RAM	Variablenname
....		
....		
0x0066FDC8	5	int i; (4 Byte)
0x0066FDCC	0x0066FDD0	int *iZeiger; (4Byte)
0x0066FDD0	20	int iVektor2[5]; (5x4Byte)
	21	
	22	
	23	
	24	
0x0066FDE4	0	int iVektor[5]; (5x4Byte)
	1	
	2	
	3	
	4	

Beispiel: Programm mit Zeigern

```
#include <iostream.h>

int main()
{
    int iVektor[5];
    int iVektor2[5];
    int *iZeiger;
    int i;

    for(i=0; i<5; i++) iVektor[i]=i;
    for(i=0; i<5; i++) iVektor2[i]=20+i;

    for(i=0; i<5; i++)
        cout<<&iVektor[i]<<" : "<<iVektor[i]<<endl;

    cout<<"++++++\n";

    iZeiger=&iVektor2[0]; //oder auch iZeiger=iVektor2

    for(i=0; i<10; i++)
    {
        cout<<iZeiger<<" : "<<*iZeiger<<endl;
        iZeiger=iZeiger+1; //erhoeht um 1 Intspeicherbereich also um 4
    }

    cout<<"++++++\n";
    cout<<"iVektor:\t"<<&iVektor<<"\t size:"<<sizeof(iVektor)<<endl;
    cout<<"iVektor2:\t"<<&iVektor2<<"\t size:"<<sizeof(iVektor2)<<endl;
    cout<<"iZeiger:\t"<<&iZeiger<<"\t size:"<<sizeof(iZeiger)<<endl;
    cout<<"i:\t\t"<<&i<<"\t size:"<<sizeof(i)<<endl;
    return 0;
}
```

Ausgabe Konsole:

```
0x0066FDE4: 0
0x0066FDE8: 1
0x0066FDEC: 2
0x0066FDF0: 3
0x0066FDF4: 4
++++++
0x0066FDD0: 20
0x0066FDD4: 21
0x0066FDD8: 22
0x0066FDDC: 23
0x0066FDE0: 24
0x0066FDE4: 0
0x0066FDE8: 1
0x0066FDEC: 2
0x0066FDF0: 3
0x0066FDF4: 4
++++++
iVektor:      0x0066FDE4      size:20
iVektor2:    0x0066FDD0      size:20
iZeiger:     0x0066FDCC      size:4
i:           0x0066FDC8      size:4
Press any key to continue
```

Zeiger als Übergabewert an Funktionen

Soll eine Funktion eine Variable des aufrufenden Programmteiles verändern, so muß mit Zeigern gearbeitet werden.

```
#include <iostream.h>

void eingabe(int *pInt);

int main()
{
    int i=0;
    eingabe(&i);
    cout<<"in main: "<<i<<endl;
    return 0;
}

void eingabe(int *pInt)
{
    cout<<"in eingabe: Bitte Int-Wert eingeben:";
    cin>>*pInt;
}
```

Sonderform Call by Reference

&Variablenname wird eine Sonderform erlaubt.

```
void eingabe(int &pInt);
```

`pInt` wird wie ein Zeiger verwendet, allerdings in der Funktion ohne den `*` Operator.

```
#include <iostream.h>

void eingabe(int &pInt);

int main()
{
    int i;
    eingabe(i);
    cout<<"in main: "<<i<<endl;
    return 0;
}

void eingabe(int &pInt)
{
    cout<<"in eingabe: Bitte Int-Wert eingeben:";
    cin>>pInt;
}
```

Datenfelder werden in Funktionsaufrufen immer als Zeiger behandelt.

Strukturen

C/C++ bietet die Möglichkeit unterschiedliche Variablen zusammen zu fassen. Es kann eine benutzerdefinierte Datenstruktur angelegt werden.

```
struct Strukturname  
{  
    Strukturvariable  
    :  
};
```

Im Programm kann die Struktur wie eine Variable verwendet werden.

Variable:

```
Strukturname s1;  
s1.Strukturvariable=..
```

Zeiger:

```
Strukturname *s2;  
s1=new(Strukturname);  
s1->Strukturvariable=..  
delete s1;
```

Datenfeld:

```
Strukturname sFeld[n];  
sFeld[i].Strukturvariable=..
```

in einer Funktion kann auch gleich eine Variable dieser Struktur angelegt werden.

```
int main()  
{  
  
    struct Geschenkliste //Struktur nur innerhalb von main bekannt  
    {  
        char Name[40];  
        int DM;  
    }MeineGeschenke[20];  
  
    strcpy(MeineGeschenke[0].Name, "TanteElma");  
    MeineGeschenke[0].DM=100;  
    cout<<MeineGeschenke[0].Name<<" : " <<MeineGeschenke[0].DM<<endl;  
    return 0;  
}
```

Beispiel:

```
#include <iostream.h>
#include <string.h>      //Befehle zur Stringmanipulation z.B. strcpy

struct strPerson //benutzerdefinierte Struktur
{
    char Name[40];
    char Vorname[40];
    int  GebJahr;
};

int main()
{
    char c;
    int i;

    strPerson Person1;           //Variablendeklaration
    strPerson *Person2;         //als Zeiger
    strPerson Personen[20];     //als Datenfeld

    strcpy(Person1.Name,"Meier"); //strcpy kopiert Zeichen in String
    strcpy(Person1.Vorname,"Hans");//Zugriff auf Elemente
                                   //der Struktur mit .
    Person1.GebJahr=1982;

    Person2 = new(strPerson);    //Person2 ist bisher ein
                                   //nichtdefinierter Zeiger
                                   //new weist Speicherplatz zu

    strcpy(Person2->Name,"Mueller"); //Zugriff auf Elemente
                                   //der Struktur mit ->
    strcpy(Person2->Vorname,"Klaus"); //mit -> Zeigeroperator
    Person2->GebJahr=1977;

    i=0;
    do
    {
        cout<<"Bitte Person Nr."<<i+1<<" eingeben"<<endl;
        cout<<"Name : ";
        cin>>Personen[i].Name;
        cout<<"Vorname : ";
        cin>>Personen[i].Vorname;
        cout<<"Geburtsjahr : ";
        cin>>Personen[i].GebJahr;

        cout<<"ende mit e:";
        cin>>c;
        i++;
    }while(c!='e');

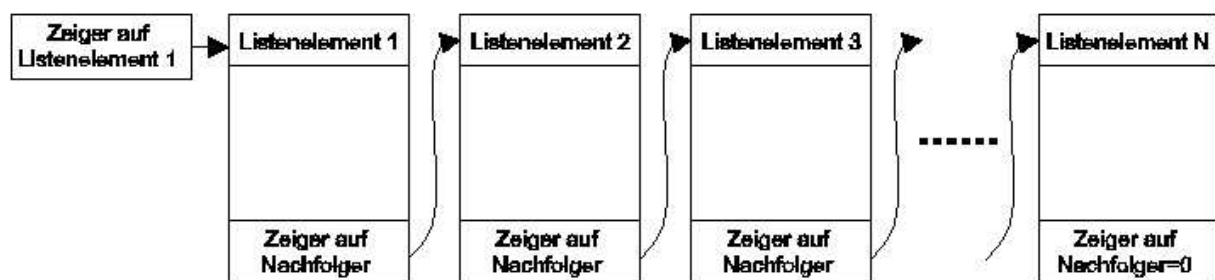
    cout<<Person1.Name<<" , "<<Person1.Vorname<<"
        , "<<Person1.GebJahr<<endl;
    cout<<Person2->Name<<" , "<<Person2->Vorname<<"
        , "<<Person2->GebJahr<<endl;

    delete (Person2);

    for(int i2=0; i2<i; i2++)
    {
        cout<<Personen[i2].Name<<" , "<<Personen[i2].Vorname<<"
            , "<<Personen[i2].GebJahr<<endl;
    }
    return 0;
}
```

Verkettete Listen (einfach linear)

In vielen Anwendungsfällen wird erst zur Laufzeit klar wieviel Speicherplätze für eine Liste benötigt werden. Der Speicher kann dynamisch angefordert, und über einen Zeiger an das vorherige Listenelement angehängt werden.



Struktur mit Zeiger auf eben diese Struktur

```
struct liste
{
    char Name[30];
    char Vorname[30];
    //Zeiger auf naechstes Listenlement
    liste *pNListe;
};
```

Anlegen der Zeiger auf erstes Listenlement und auf das aktuelle Listenelement

```
liste *pStartListe;
liste *pAktuellesElement;
```

Speicheranforderung für das erste Listenelement

```
pStartListe=new(liste);
pAktuellesElement=pStartListe;
//0 - noch keine neue Liste angehaengt
pAktuellesElement->pNListe=NULL;
```

Neues Listenelement anhängen

```
pAktuellesElement->pNListe=new(liste);
pAktuellesElement=pAktuellesElement->pNListe;
pAktuellesElement->pNListe=NULL;
```

Liste durcharbeiten

```
pAktuellesElement=pStartListe;
pAktuellesElement=pAktuellesElement->pNListe;
```

Beispielprogramm

```
#include <iostream.h>

struct liste
{
    char Name[30];
    char Vorname[30];
    //Zeiger auf naechstes Listenlement
    liste *pNListe;
};

int main()
{
    char c;
    int i;
    liste *pStartListe;
    liste *pAktuellesElement;

    pStartListe=new(liste);
    pAktuellesElement=pStartListe;
    //0 - noch keine neue Liste angehaengt
    pAktuellesElement->pNListe=NULL;

    i=0;
    do
    {
        i++;
        cout<<"Nr.: "<<i<<" Name Vorname ";
        cin>>pAktuellesElement->Name;
        cin>>pAktuellesElement->Vorname;

        pAktuellesElement->pNListe=new(liste);
        pAktuellesElement=pAktuellesElement->pNListe;
        pAktuellesElement->pNListe=NULL;

        cout<<"\tneue Eingabe mit 'n' beenden mit 'e': ";
        cin>>c;

    }while(c!='e');

    cout<<"Liste Ausgeben#####"<<endl;

    pAktuellesElement=pStartListe;
    i=0;
    while(    pAktuellesElement->pNListe!=NULL)
    {
        i++;
        cout<<"Adr: "<<pAktuellesElement;
        cout<<" Nr.: "<<i<<"\t"<<pAktuellesElement->Vorname<<" ,
            "<<pAktuellesElement->Name<<endl;
        pAktuellesElement=pAktuellesElement->pNListe;
    }

    liste *pDeleteElement;
    pAktuellesElement=pStartListe;
    i=0;
    while(    pAktuellesElement->pNListe!=NULL)
    {
        pDeleteElement=pAktuellesElement;
        pAktuellesElement=pAktuellesElement->pNListe;
        delete(pDeleteElement);
    }
    delete(pAktuellesElement);
    return 0;
}
```

Module

Um bei größeren Projekten den Überblick zu behalten, ist es üblich den Quellcode modular zu gestalten, d.h. Softwareteile werden in unterschiedlichen Dateien verwaltet.

Main.cpp

```
#include <iostream.h>
#include "pytagoras.h"

int main()
{
    double a,b,c;

    cout<<"Bitte a eingeben: ";
    cin>>a;
    cout<<"Bitte b eingeben: ";
    cin>>b;

    c=pytagoras(a,b);
    cout<<"c="<<c<<endl;
    return 0;
}
```

pytagoras.h

```
#if !defined pytagoras_h
#define pytagoras_h

double pytagoras(double a,double b);

#endif
```

pytagoras.cpp

```
#include <math.h>
#include "pytagoras.h"

double pytagoras(double a,double b)
{
    return sqrt(a*a+b*b);
}
```

Klassen

Eine Klasse stellt die Erweiterung der bisher bekannten Struktur `struct` dar.

In einer Klasse werden Variablen (auch Eigenschaften genannt) und Funktionen (auch Verhaltensweise genannt) vereint, es entsteht ein sogenanntes Objekt.

Mit den Methoden der objektorientierten Programmierung sollen real existierende Objekte als Einheit abgebildet werden.

Die Trennung von Daten und Funktionen wird aufgehoben.

Die Umsetzung eines Objektes in C++ erfolgt durch die Programmierung einer Klasse (`class`).

Das Objekt enthält Eigenschaften die durch Membervariablen beschrieben werden und Verhaltensweisen die als Methoden realisiert werden.

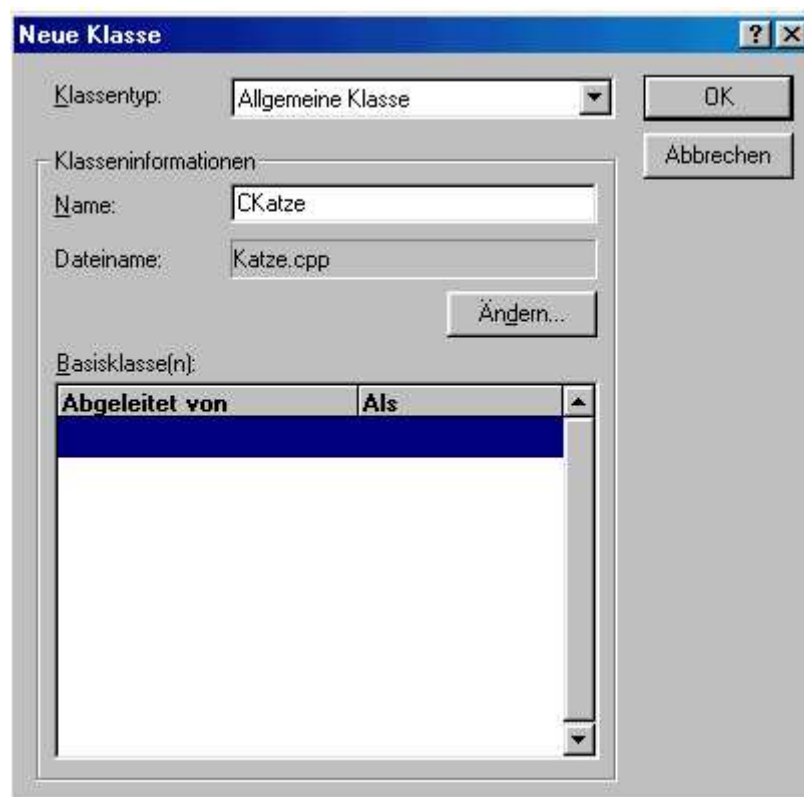
In einer Klasse können alle Methoden direkt auf alle Membervariablen der Klasse zugreifen.

Die im Programm angelegte Klasse nennt man Instanz.

In einer Klasse kann die Instanz einer andere Klasse angelegt werden.

<code>class CKatze</code>	Legt den Namen der Klasse fest
<code>{...</code>	
<code>private:</code>	Zugriff nur innerhalb der Klasse erlaubt
<code>public:</code>	Zugriff von außerhalb möglich
<code>CKatze();</code>	Konstruktor wird bei der Initialisierung automatisch aufgerufen.
<code>virtual ~CKatze();</code>	Destruktor wird aufgerufen, wenn die Instanz der Klasse ihre Gültigkeit verliert.
<code>CKatze::Methodenname</code>	Zuordnung der Methode zu einer Klasse in der cpp Datei
<code>void main()</code>	Instanz der Klasse CKatze anlegen
<code>{</code>	
<code> CKatze flecki;</code>	
<code> :</code>	
<code> :</code>	

Anlegen einer neuen Klasse mit VC 6.0



Die Dateien Katze.cpp und Katze.h werden automatisch erzeugt und dem Projekt hinzugefügt.

Beispiel CKatze

```
// Katze.h: Schnittstelle für die Klasse CKatze.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#if !defined(AFX_KATZE_H__CDD0D680_0B3B_11D6_B1EC_00105AC323DC__INCLUDED_)
#define AFX_KATZE_H__CDD0D680_0B3B_11D6_B1EC_00105AC323DC__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CKatze
{
private:

    double dGewicht; //neu in gramm

public:

    int iGeburtsjahr; //neu

    CKatze();
    CKatze(int Jahr); //neu
    virtual ~CKatze();

    double getGewicht(); //neu
    void Gewichtsanderung(double dAenderung); //neu

};

#endif //
!defined(AFX_KATZE_H__CDD0D680_0B3B_11D6_B1EC_00105AC323DC__INCLUDED_)

// Katze.cpp: Implementierung der Klasse CKatze.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include "Katze.h"

CKatze::CKatze()
{
    dGewicht=300;
    iGeburtsjahr=0;
}

//neu
CKatze::CKatze(int Jahr)
{
    dGewicht=300;
    iGeburtsjahr=Jahr;
}

CKatze::~~CKatze()
{
}

//neu
double CKatze::getGewicht()
{
    return dGewicht;
}

//neu
void CKatze::Gewichtsanderung(double dAenderung)
{
    dGewicht=dGewicht+dAenderung;
}
```

```
//main.cpp

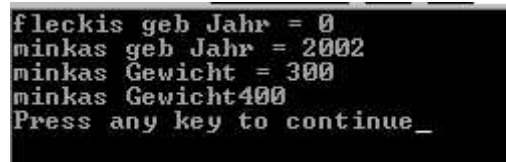
#include <iostream.h>
#include "katze.h"

int main()
{
    CKatze flecki;
    CKatze minka(2002);

    cout<<"fleckis geb Jahr = "<<flecki.iGeburtsjahr<<endl;
    cout<<"minkas geb Jahr = "<<minka.iGeburtsjahr<<endl;

    cout<<"minkas Gewicht = "<<minka.getGewicht()<<endl;
    minka.Gewichtsaenderung(100);
    cout<<"minkas Gewicht"<<minka.getGewicht()<<endl;

    return 0;
}
```



```
fleckis geb Jahr = 0
minkas geb Jahr = 2002
minkas Gewicht = 300
minkas Gewicht400
Press any key to continue_
```

this Pointer

Wie für Variablen und Strukturen wird für jede Instanz einer Klasse Speicherbereich belegt. Nicht in allen Fällen ist der Name der Instanz bekannt. Aus diesem Grund verfügt jede Klasse über eine „Funktion“ this, welche die Adresse der Instanz zurück gibt.

Beispiel:

myclass.h

```
//Doppelteinbindung Verhindern
#ifndef my_class_h
#define myclass_h
//Klasse anlegen
class CMyClass
{
//nur innerhalb der Klasse verfuegbar
private:
    int i;
//ausserhalb verfuegbar
public:
    //Konstruktor wird bei Anlegen der Instanz ausgefuehrt
    CMyClass();
    //Destruktor wird ausgefuehrt wenn die Instanz ihre
    //Gueltigkeit verliert
    ~CMyClass();
    //Methoden der Klasse
    void methode_myclass();
    void die_methode();
};
//Doppelteinbindung Verhindern
#endif
```

myclass.cpp

```
//verwendete Bibliotheken allgemein
#include <iostream.h>
//Header File der Klasse
#include "myclass.h"

//Konstruktor
CMyClass::CMyClass()
{
    i=0;
    cout<<"++Konstruktor CMyClass: "<<this<<endl;
}
//Destruktor
CMyClass::~CMyClass()
{
    cout<<"--Destruktor CMyClass: "<<this<<endl;
}
//Methode
void CMyClass::methode_myclass()
{
    i=1;
    cout<<" Methode CMyClass: "<<this<<endl;
}
void CMyClass::die_methode()
{
    cout<<" Die CMyClass: "<<this<<endl;
}
```

main.cpp

```
#include <iostream.h>
#include "myclass.h"

int main()
{
    //Programmcode                                //Ausgabe Bildschirm

    //Instanz von CMyClass anlegen
    CMyClass meineKlasse;                        //++Konstruktor CMyClass: 0x0065FDE8

    //Zeiger vom Typ CMyClass
    CMyClass *pointerMeineKlasse;

    //Aufruf einer Methode
    meineKlasse.methode_myclass();              //Methode CMyClass: 0x0065FDE8

    //Speicherbereich fuer CMyClass
    //belegen und Adresse an pointer
    //übergeben
    pointerMeineKlasse = new CMyClass;          //++Konstruktor CMyClass: 0x00780440

    //pointer Anzeigen
    cout<<"pointerMeineKlasse: "<<pointerMeineKlasse<<endl;
                                                //pointerMeineKlasse: 0x00780440

    //Aufruf methode mit pointer
    pointerMeineKlasse->methode_myclass();      //Methode CMyClass: 0x00780440

    //Speicherplatz freigeben
    delete pointerMeineKlasse;                  //--Destruktor CMyClass: 0x00780440

    cout<<"Programmende"<<endl;                //Programmende

                                                //--Destruktor CMyClass: 0x0065FDE8
                                                //Press any key to continue
}
}
```

Vererbung

Die modulare Programmierung ermöglicht die Wiederverwendbarkeit von Code. Eine weitere Möglichkeit ist in einer Objekt Orientierten Programmiersprache die Vererbung. Eine neue Klasse erbt die Eigenschaften (Membervariablen) und Verhaltensweisen (Methoden) einer anderen Klasse. Man spricht von der Basisklasse und der abgeleiteten Klasse. Eine Katze ist z.B. eine Spezialisierung eines allgemeinen Tieres.

```
//Basisklasse
class CTier
{
    :
    :
}

//abgeleitete Klasse
class CKatze: public CTier
{
    :
    :
}
```

In der abgeleiteten Klasse kann nur auf Membervariablen und Methoden zugegriffen werden, die als public oder protected deklariert sind.

Der Aufruf der Konstruktoren erfolgt in der Reihenfolge der Ableitungen.
Konstruktor Basisklasse, Konstruktor abgeleitete Klasse.
Der Aufruf der Destruktoren erfolgt in umgekehrter Reihenfolge.
Destruktor abgeleitete Klasse, Destruktor Basisklasse.

Beispiel:

abgclass.h

```
//Doppelteinbindung Verhindern
#ifndef abgclass_h
#define abgclass_h

#include "myclass.h"

//Klasse anlegen
class CAbgClass :public CMyClass
{

//nur innerhalb der Klasse verfuegbar
private:
    int i;

//ausserhalb verfuegbar
public:

    //Konstruktor wird bei Anlegen der Instanz ausgeführt
    CAbgClass();

    //Destruktor wird ausgeführt wenn die Instanz ihre
    //Gültigkeit verliert
    ~CAbgClass();

    //Methoden der Klasse
    void methode_abgclass();
    void die_methode();
    void test();
};
//Doppelteinbindung Verhindern
#endif
```

abgclass.cpp

```
//verwendete Bibliotheken allgemein
#include <iostream.h>
//Header File der Klasse
#include "abgclass.h"

//Konstruktor
CAbgClass::CAbgClass()
{
    i=0;
    cout<<"++Konstruktor CAbgClass: "<<this<<endl;
}
//Destruktor
CAbgClass::~~CAbgClass()
{
    cout<<"--Destruktor CAbgClass: "<<this<<endl;
}
//Methode
void CAbgClass::methode_abgclass()
{
    i=1;
    cout<<" Methode CAbgClass: "<<this<<endl;
}
void CAbgClass::die_methode()
{
    cout<<" Die Methode CAbgClass: "<<this<<endl;
}
```

main.cpp

```
#include <iostream.h>
#include "myclass.h"
#include "abgclass.h"

int main()
{
    //Programmcode                               //Ausgabe Bildschirm

    //Instanz von CAbgClass anlegen
    CAbgClass abgeleiteteKlasse; //++Konstruktor CMyClass: 0x0065FDE4
                                //++Konstruktor CAbgClass: 0x0065FDE4

    //Aufruf einer Methode
    abgeleiteteKlasse.methode_abgclass();
                                //Methode CAbgClass: 0x0065FDE4
    abgeleiteteKlasse.methode_myclass();
//Methode CMyClass: 0x0065FDE4
    abgeleiteteKlasse.die_methode();
//Die Methode CAbgClass: 0x0065FDE4
    //Aufruf der Methode der Basisklasse
    abgeleiteteKlasse.CMyClass::die_methode();
                                //Die Methode CMyClass: 0x0065FDE4

    CAbgClass *zeigerAbgKlasse;
    zeigerAbgKlasse = new CAbgClass;
                                //++Konstruktor CMyClass: 0x00780420
                                //++Konstruktor CAbgClass: 0x00780420

    //Aufruf der Methode der Basisklasse mit einem Zeiger
    zeigerAbgKlasse->CMyClass::die_methode();
                                // Die Methode CMyClass: 0x00780420

    delete zeigerAbgKlasse;
                                //--Destruktor CAbgClass: 0x00780420
                                //--Destruktor CMyClass: 0x00780420

    cout<<"Programmende"<<endl; //Programmende
    return 0;

                                //--Destruktor CAbgClass: 0x0065FDE4
                                //--Destruktor CMyClass: 0x0065FDE4
                                //Press any key to continue
}
```

Windowsprogrammierung

Dialoganwendung

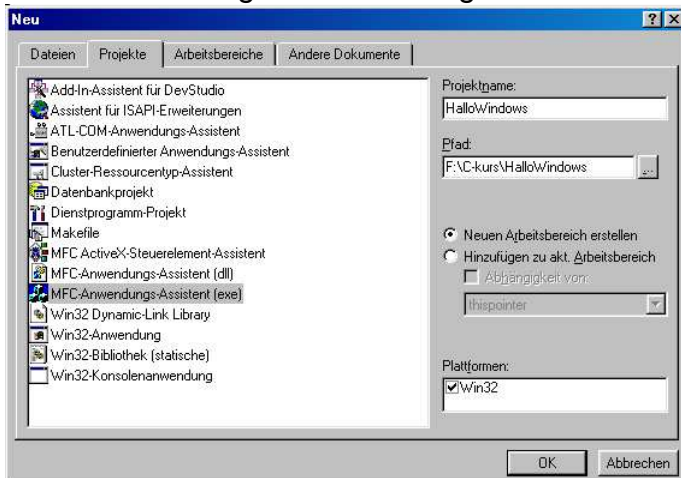
Ein Dialoganwendung eignet sich zum Einstieg in die Windowsprogrammierung am besten, da es sich um den einfachsten Typ ein Windowsanwendung handelt.



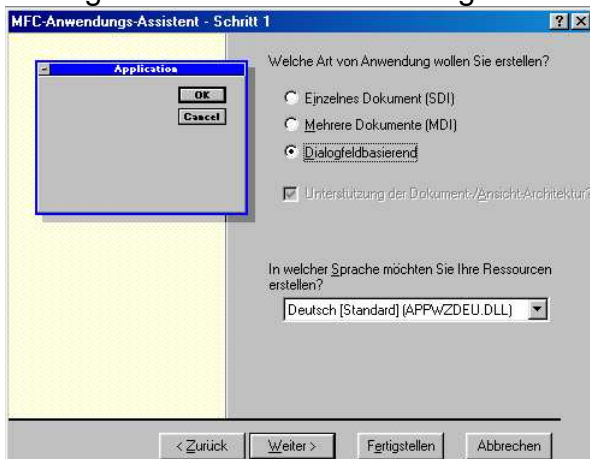
Bild: einfache Dialoganwendung

Erstellen einer Dialoganwendung

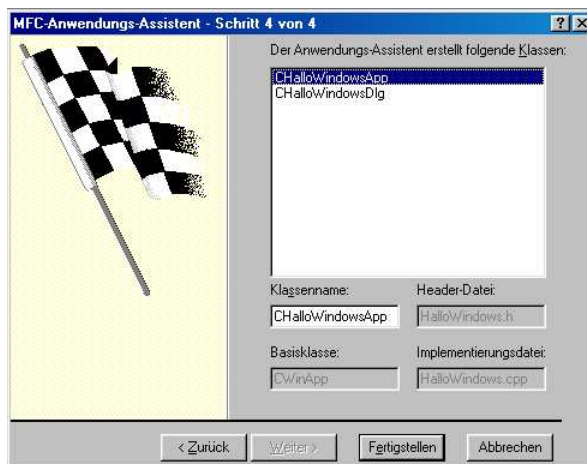
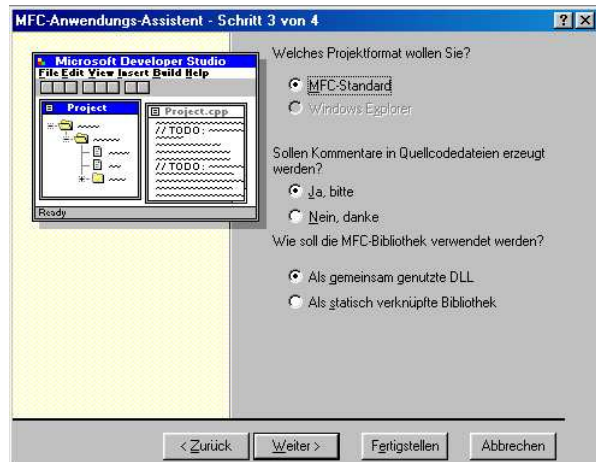
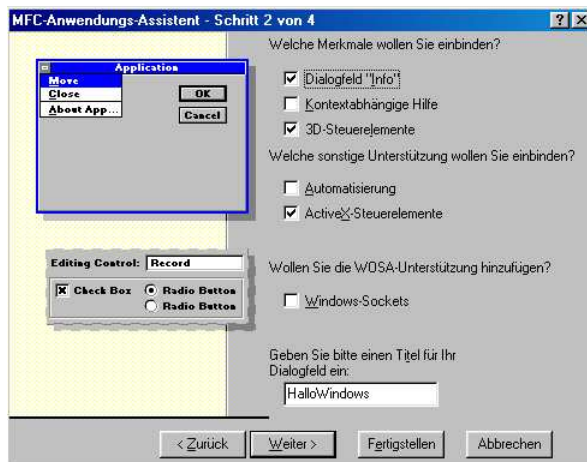
Wie bisher wird auch diesmal ein neues Projekt angelegt allerdings wird nun der MFC-Anwendungsassistent ausgewählt.



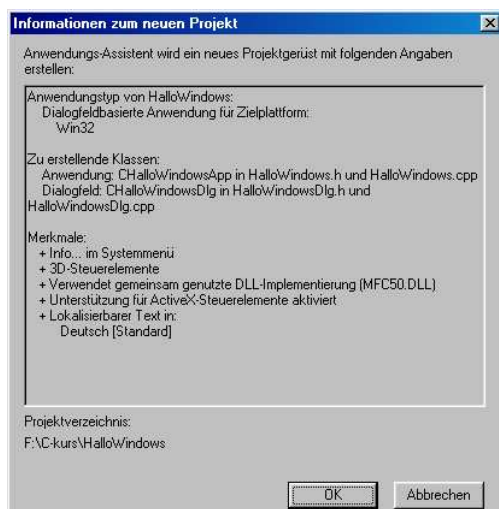
Im folgenden Fenster wird Dialogfeldbasierend als Art ausgewählt.



Die nächsten drei Fenster können mit ihren Voreinstellungen übernommen werden.



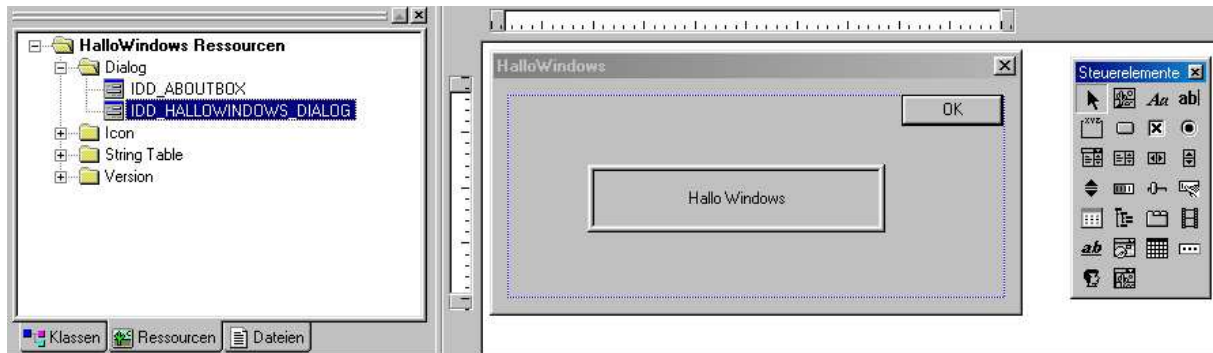
Beim letzten Fenster handelt es sich nur um ein Statusfenster das einfach bestätigt wird.



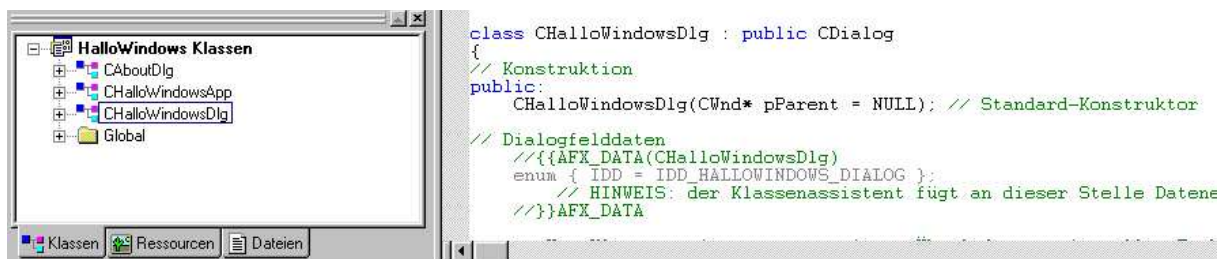
Ansichten im Arbeitsbereich

Die Ressourcen-Ansicht bietet die Übersicht über die im Projekt vorhandenen Ressourcen.

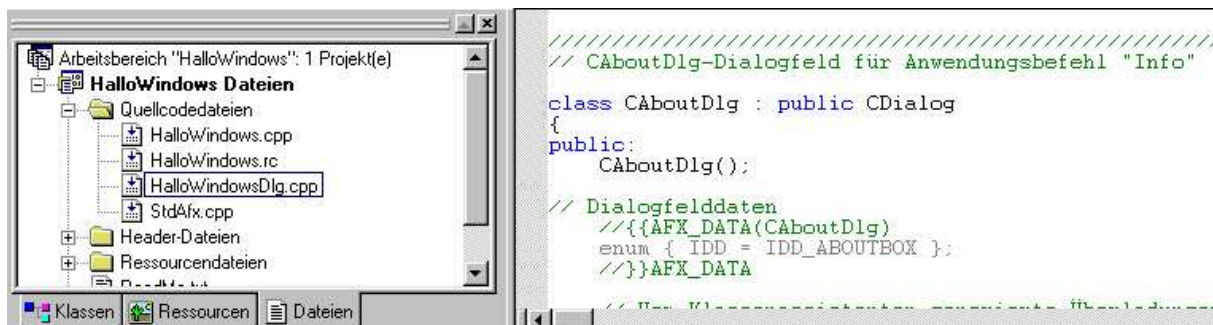
In dieser Ansicht findet auch die grafische Gestaltung der entsprechenden Anwendungen statt. Durch einen Doppelklick auf den Eintrag wird das gewünschte Element dargestellt.



Die Klassenansicht bietet eine Übersicht über die verwendeten Klassen. Mit einem Doppelklick auf den entsprechenden Eintrag wird der zugehörige Quellcode angezeigt.



Die Dateiansicht verschafft wie gewohnt den Überblick über die im Projekt verwendeten Dateien.



Elemente der Dialoganwendung

Button

Button in dem Fenster Dialoganwendungen auswählen und an der entsprechenden Stelle der Dialoganwendung positionieren.



Eigenschaften

Ein Klick mit der rechten Maustaste auf das Element Auswahl Eigenschaften öffnet das Eigenschaften Fenster.

Alle Buchstaben, die die ID benennen werden groß geschrieben IDC_ wird beibehalten.



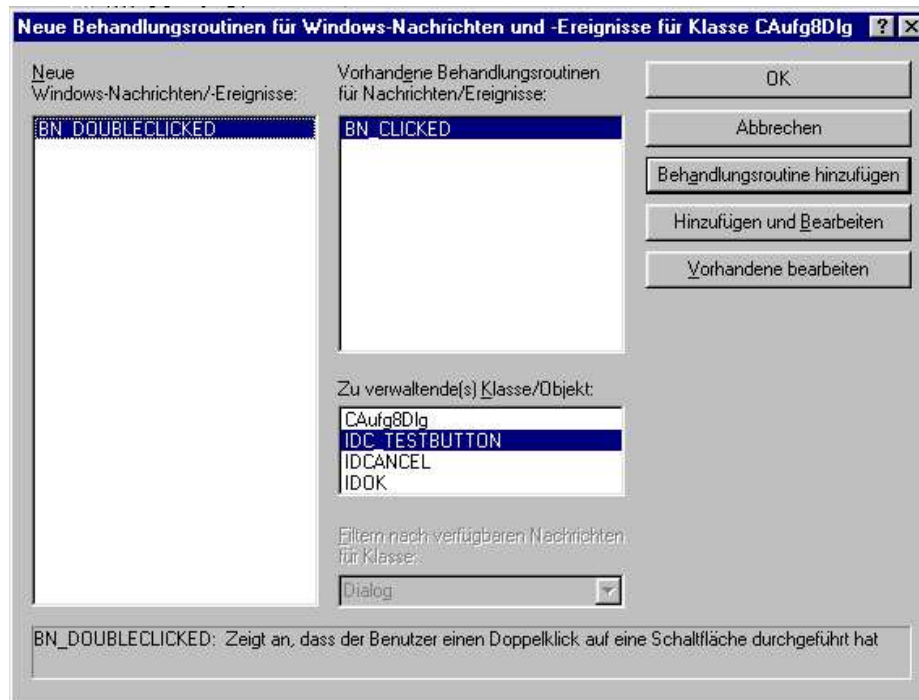
Klassenassistent

Klick mit der rechten Maustaste Auswahl Klassenassistenten.

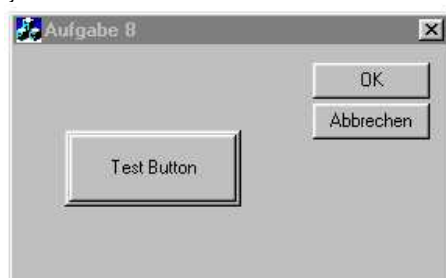
Im Klassenassistenten können Membervariablen oder Behandlungsroutinen zugewiesen werden.

Durch einen Doppelklick auf einen Button wird das Menü Behandlungsroutinen direkt aufgerufen.

Die Vorgehensweise bei Eingabefeldern Edit-Box ist ähnlich, allerdings ist es sinnvoll hier eine Memebervariable des gewünschten Datentyps anzulegen.



```
BEGIN_MESSAGE_MAP(CAufg8Dlg, CDialog)
    //{{AFX_MSG_MAP(CAufg8Dlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_TESTBUTTON, OnTestbutton)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
+++++
void CAufg8Dlg::OnTestbutton()
{
    // TODO: Code für die Behandlungsroutine der
    //Steuerelement-Benachrichtigung hier einfügen
    MessageBox(" Der Button Test wurde betätigt",MB_OK);
}
```



Entfernen des OK Buttons

Der Ok Button darf erst nach dem Anlegen der entsprechenden Behandlungsroutine entfernt werden (doppelklick auf den OK Button).

In der Methode OnOK() wird der Befehl CDialog::OnOK(); entfernt. Wird der OK-Button ohne diese Maßnahme entfernt so führt jedes betätigen der Return –Taste zur Beendigung des Programms.

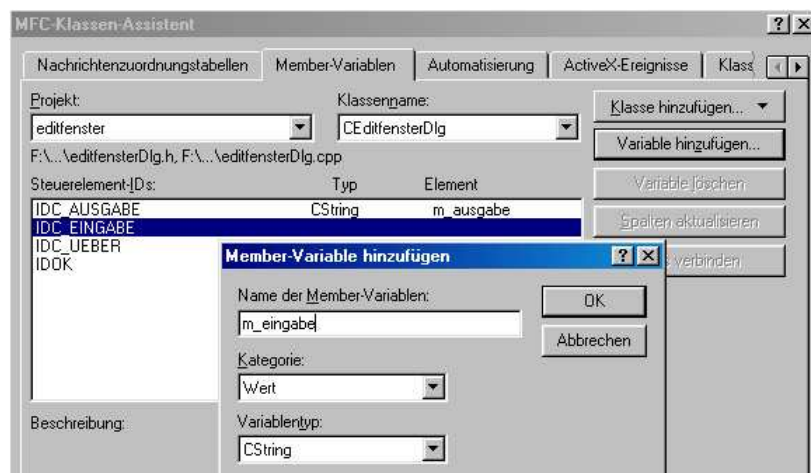
Der Aufruf des Befehl CDialog::OnOK(); in einer Behandlungsroutine führt zur Beendigung des Programms.

Eingabefeld

Das Eingabefeld wird in der Anwendung positioniert. Danach werden die IDs sinnvoll vergeben.



Im zweiten Schritt werden den Eingabefeldern Memebervariablen zugewiesen. Über die Memebervariablen kann auf die Eingabefelder zugegriffen werden. Bei Memebervariablen kann zwischen Wert und Control gewählt werden. Eine Controlvariable erlaubt auch den Zugriff auf die Eigenschaften z.B. erlaubt sie das deaktivieren des Eingabefeldes.



UpdateData

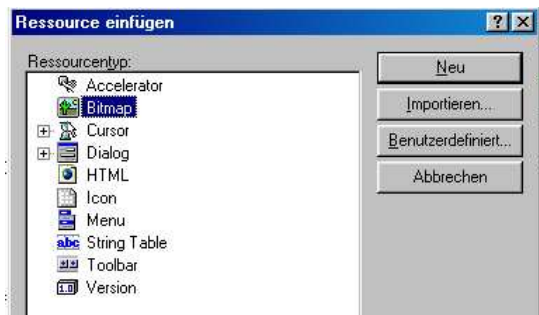
Die Darstellung z.B. in einem Eingabefeld ist unabhängig von dem Inhalt der entsprechenden Memebervariable. Erst durch den Aufruf der Funktion `UpdateData(true)` wird der Inhalt eines Eingabefeldes in die entsprechende Memebervariable kopiert. Soll der Inhalt einer Memebervariable dargestellt werden so ist ebenfalls die Funktion `UpdateData(false)` diesmal aber mit dem Übergabewert `false` aufzurufen.

Beispiel:

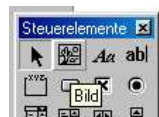
```
void CEditfensterDlg::OnUeber()
{
    UpdateData(true);
    m_ausgabe=m_eingabe;
    m_eingabe=" ";
    UpdateData(false);
}
```

Übung: Taschenrechner

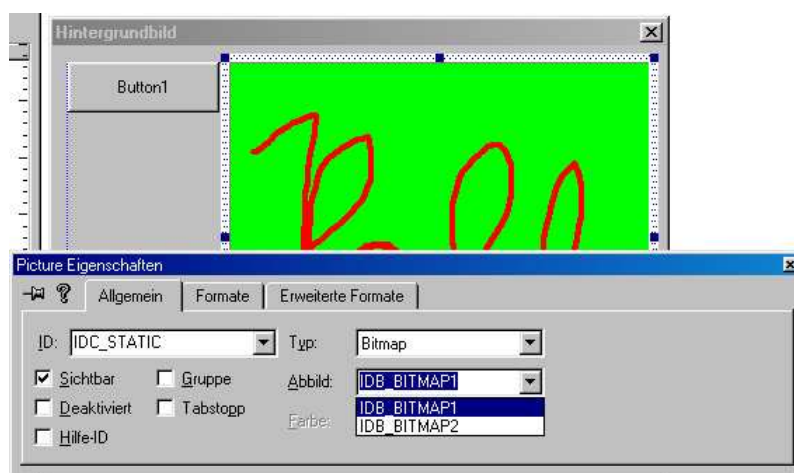
Statische Bilder als Ressource einbinden



Bitmap Neu: legt neue Bitmap an Editieren im ICON-Editor
Bitmap Importieren..: Import aus Datei *.bmp (256 Farben erforderlich)
Das entsprechende Bild steht nun als Ressource zur Verfügung und kann verwendet werden.



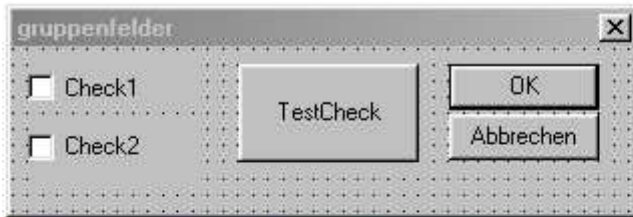
Steuerelement Bild auf Dialog positionieren



Picture Eigenschaften

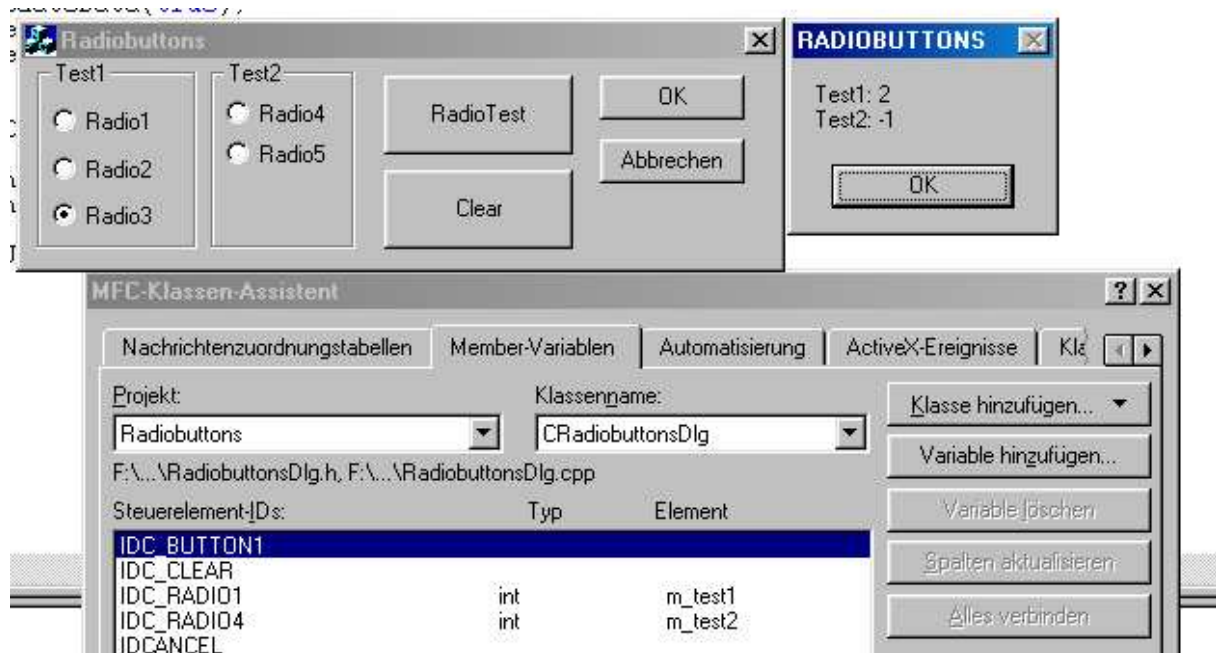
Typ: Bitmap
Abbild: Auswahl der Ressource

Kontrollfeld

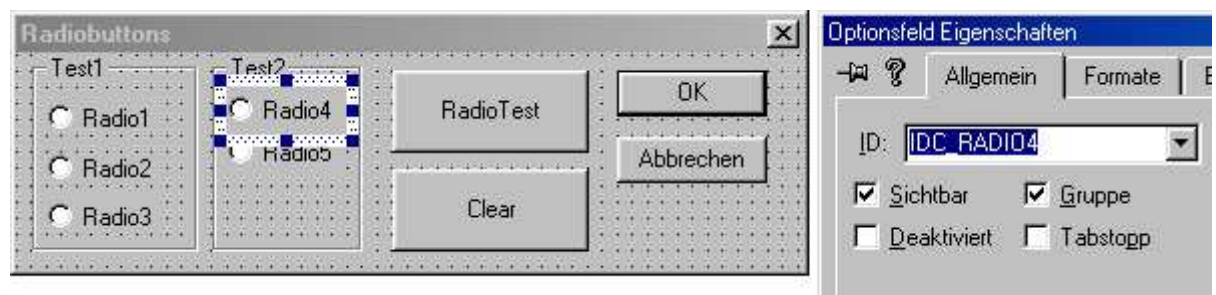


```
void CGruppenfelderDlg::OnTestcheck()
{
    CString help;
    UpdateData(true);
    help.Format("Check1: %d\nCheck2: %d", m_check1, m_check2);
    MessageBox(help);
}
```

Optionsfelder



Gruppieren durch Auswahl Gruppe

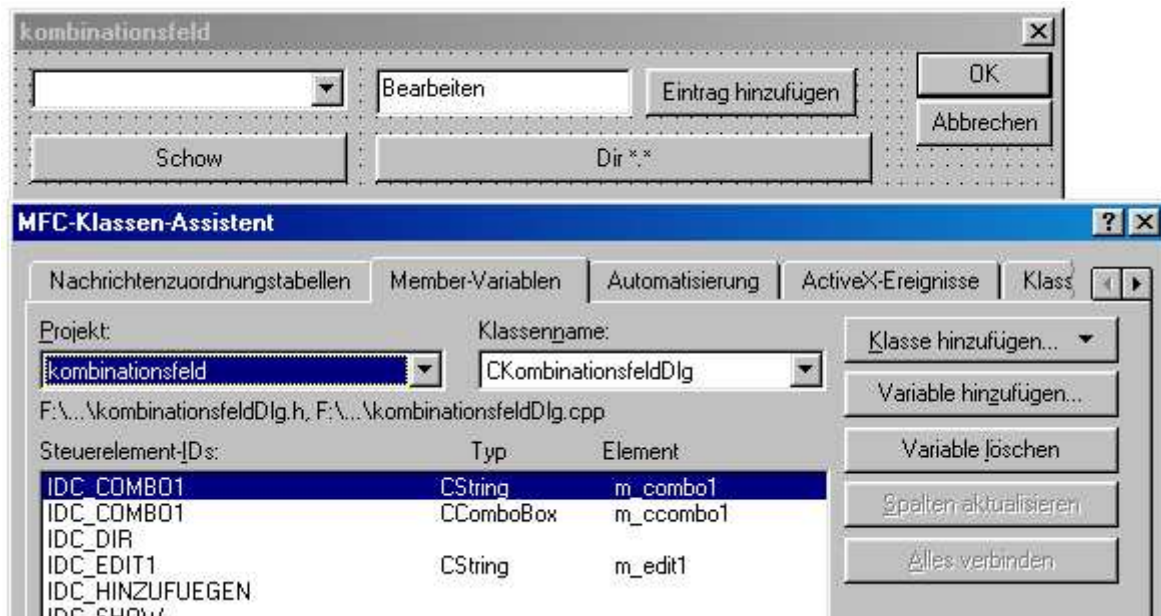


```
void CRadiobuttonsDlg::OnButton1()
{
    CString help;
    UpdateData(true);
    help.Format("Test1: %d\nTest2: %d", m_test1, m_test2);
    MessageBox(help);
}

void CRadiobuttonsDlg::OnClear()
{
    m_test1=-1;
    m_test2=-1;

    UpdateData(false);
}
```

Kombinationsfeld



```
void CKombinationsfeldDlg::OnHinzufuegen()
{
    UpdateData(true);
    m_ccombol.AddString(m_edit1);
}

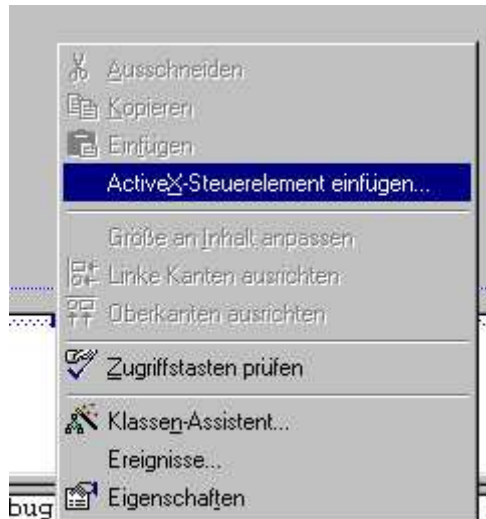
void CKombinationsfeldDlg::OnDir()
{
    m_ccombol.ResetContent();
    m_ccombol.Dir(0, "*. *");
}

void CKombinationsfeldDlg::OnShow()
{
    CString help;
    UpdateData(true);
    help="notepad "+m_combol;
    system(help);
}
```

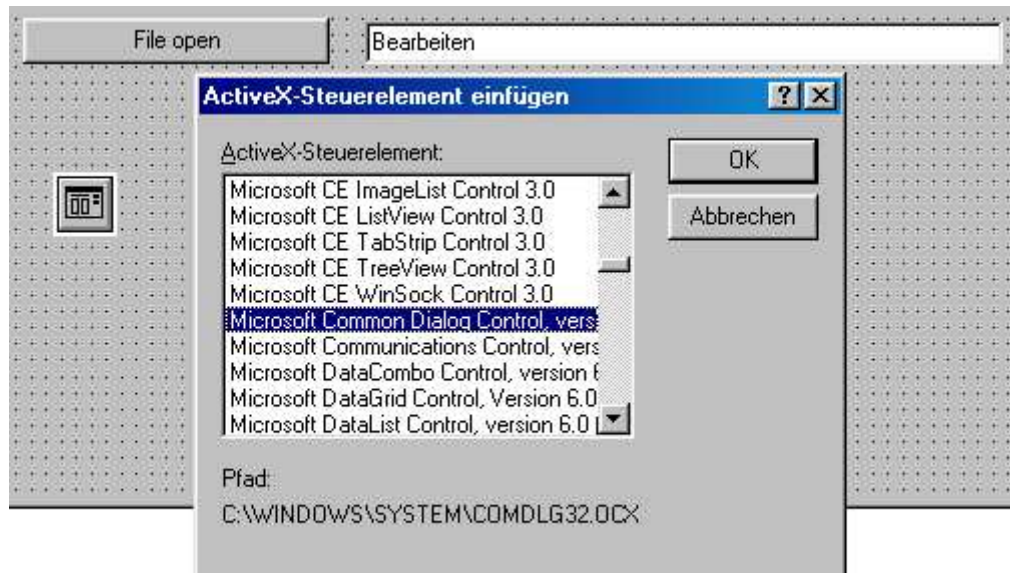
ActiveX - Steuerelemente

File Dialog

Einfügen: Klick rechte Maustaste auf Dialog,
Auswahl: ActiveX-Steuerelement einfügen..



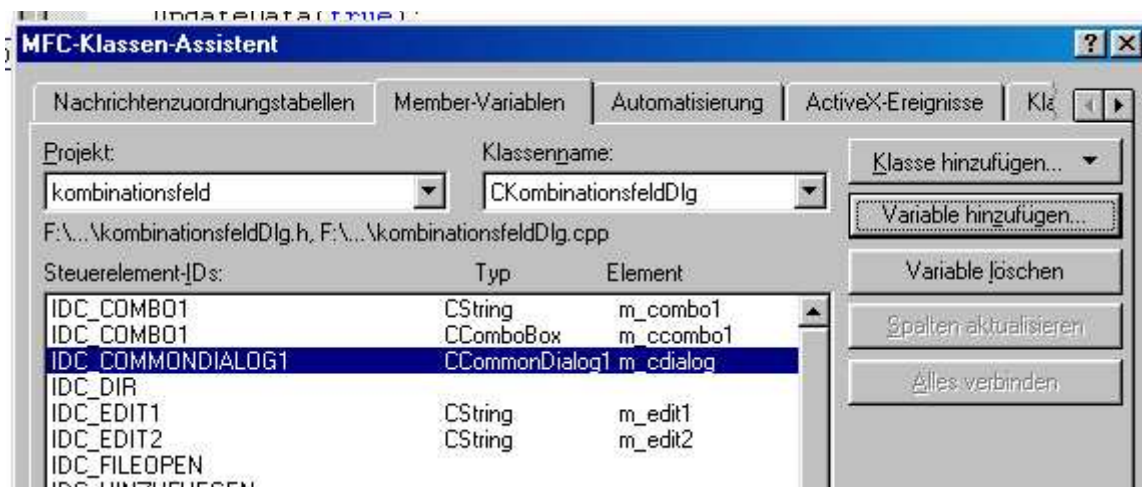
Auswahl: Microsoft Common Dialog Control



ID Zuweisen



Controlvariable zuweisen



Beispiel Code

```
void CKombinationsfeldDlg::OnFileopen()
{
    m_cdialog.SetInitDir("C:\\");
    m_cdialog.SetFileName("*.txt");
    m_cdialog.ShowOpen();
    m_edit2=m_cdialog.GetFileName();
    UpdateData(false);
}
```

Datei anzeigen:

```
ShellExecute(m_hWnd, "open", m_edit2, NULL, NULL, SW_SHOW );
```

oder

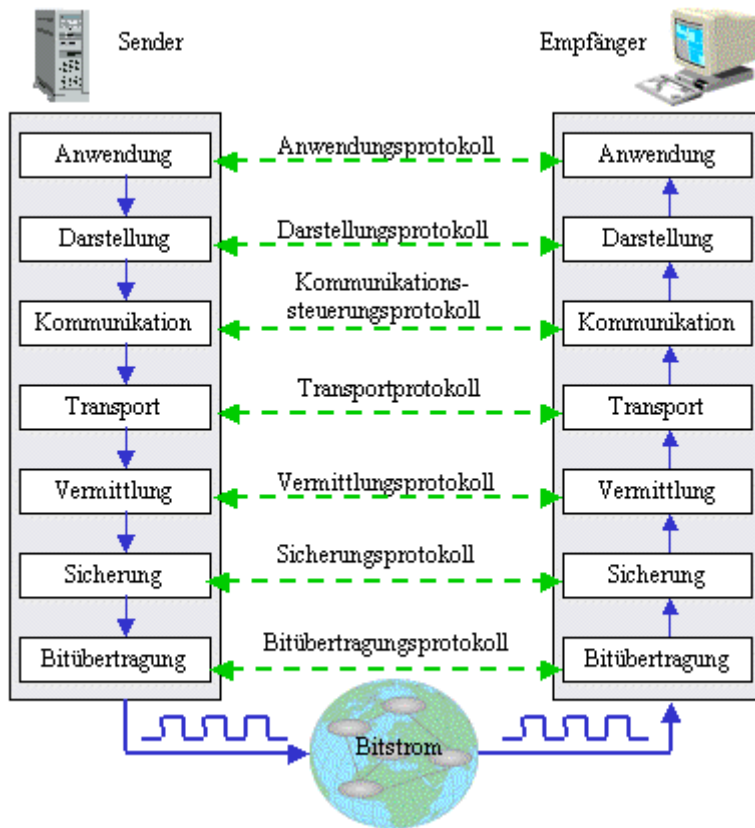
```
CString help;
help="notepad "+ m_cdialog.GetFileName();
system(help);
```

Anzeige in Micosoft Richttextbox Control:

```
m_rtf.SetFileName(m_datei);
m_rtf.Refresh();
```

Netzwerkprogrammierung

Schichtenmodell



Anwendung	File Transfer	Electronic Mail	Terminal Emulation	Usernet News	Gopher	WAIS	WWW	Domain Name Service	Archie
Darstellung	File Transfer Protocol (FTP)	Simple Mail Transfer Protocol (SMTP)	Telnet Protocol (Telnet)	Network News Transfer Protocol (NNTP)	Internet Gopher Protocol	Z39.50	Hyper Text Transfer Protocol (http)	Domain Name System (DNS)	Prospero Protocol
Kommunikation									
Transport	Transmission Control Protocol (TCP)							User Datagram Protocol	
Vermittlung	Address Resolution Protocol (ARP)		Internet Protocol (IP)				Internet Control Message Protocol (ICMP)		
Sicherung	Ethernet, Token Ring, DQDB, FDDI								
Bitübertragung	Übertragungsmedium Doppelader, Koaxkabel, Lichtwellenleiter, drahtlose Übertragung								

Quelle: <http://www.wi1.wiso.uni-goettingen.de/pa/reco/kompetenz/schichtenmodell/1.htm>

Beispiel UDP Daytime

Instanz der Klasse CSocket zur Verwaltung eines Winsockets

```
CSocket mySocket;
```

Anlegen eines Sockets Typ Datagram verwendeter Port 5000

```
if (!mySocket.Create(5000,SOCK_DGRAM,NULL)) MessageBox("Create Error");
```

Senden eines Paketes an Port 13 des angegebenen Daytime-Servers.

```
char outbuffer[]="Hallo";  
mySocket.SendTo(&outbuffer, sizeof(outbuffer),13,"ptbtimel.ptb.de",0);
```

Warten auf Daten max. 100 Byte. Die tatsächliche Anzahl der empfangenen Bytes steht in rec.

```
int rec;  
char inbuffer[100];  
rec=mySocket.Receive(&inbuffer, sizeof(inbuffer),0);
```

Socket schließen

```
mySocket.Close();
```

Empfangene Daten in CString abspeichern

```
CString myDate;  
inbuffer[rec]=0;  
myDate=inbuffer;
```

Auszug aus dem Datenverkehr

13	IP-141.47.45.4	IP-141.47.70.1	79	11:04:51.501000	UDP DNS	ptbtimel.ptb.de.
14	IP-141.47.70.1	IP-141.47.45.4	304	11:04:51.502000	UDP DNS	ptbtimel.ptb.de.
15	IP-141.47.45.4	IP-192.53.103.103	64	11:04:51.507000	IP UDP	
16	00:e0:18:c1:79:0e	03:00:00:00:00:01	65	11:04:51.543000	NB AdNQ	
17	IP-192.53.103.103	IP-141.47.45.4	75	11:04:51.551000	IP UDP	

Paket 13 (Auszug)

```
Source IP Address: 141.47.45.4  
Dest. IP Address: 141.47.70.1  
No Internet Datagram Options  
UDP - User Datagram Protocol  
Source Port: 1167  
Destination Port: 53 Domain Name Server  
Length: 41  
Checksum: 0xa3f1  
DNS - Domain Name System Protocol  
Identification: 0x0001  
Parameter: 0x0100  
Request  
Standard Query  
Recursion Desired  
Number of Questions: 1  
Number of Answers: 0  
Number of Authority: 0  
Number of Additional: 0  
Query Domain Name: ptbtimel.ptb.de
```

Paket 14 (Auszug)

Answer Type: 1 *Host Address*
Answer Class: 1 *Internet*
Time to Live: 13277
Resource Data Length: 4
Resource Data: 192.53.103.103

Paket 15

Flags: 0x00
Status: 0x02 *Truncated*
Packet Length: 64 **Slice Length:** 48
Timestamp: 11:04:51.507000 04/11/2002

Ethernet Header

Destination: 00:01:64:10:bd:ec
Source: 00:10:5a:c3:23:dc
Protocol Type: 08-00 *IP*

IP Header - Internet Protocol Datagram

Version: 4
Header Length: 5
Precedence: 0
Type of Service: %0000
Unused: %0
Total Length: 34
Identifier: 11530
Fragmentation Flags: %000
Fragment Offset: 0
Time To Live: 128
IP Type: 0x11 *UDP*
Header Checksum: 0x2bf1
Source IP Address: 141.47.45.4
Dest. IP Address: 192.53.103.103

No Internet Datagram Options

UDP - User Datagram Protocol

Source Port: 5000
Destination Port: 13 *Daytime*
Length: 14
Checksum: 0xe69e
UDP Data Area: No more data.

Extra bytes (Padding):

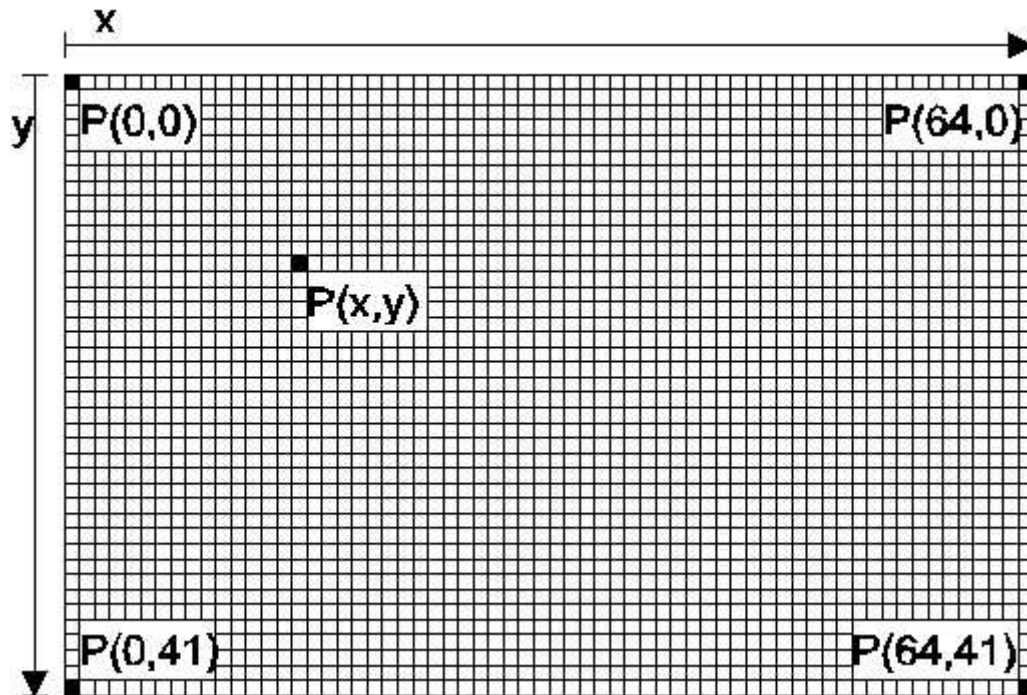
Hallo. 48 61 6c 6c 6f 00
Packet is too short for further decode.
Bytes short: 16

Paket 17

```
Flags:          0x00
  Status:       0x00
  Packet Length: 75
  Timestamp:    11:04:51.551000 04/11/2002
Ethernet Header
  Destination:  00:10:5a:c3:23:dc
  Source:       00:01:64:10:bd:ec
  Protocol Type: 08-00  IP
IP Header - Internet Protocol Datagram
  Version:      4
  Header Length: 5
  Precedence:   0
  Type of Service: %0000
  Unused:      %0
  Total Length: 57
  Identifier:   60158
  Fragmentation Flags: %000
  Fragment Offset: 0
  Time To Live: 47
  IP Type:     0x11  UDP
  Header Checksum: 0xbee5
  Source IP Address: 192.53.103.103
  Dest. IP Address: 141.47.45.4
  No Internet Datagram Options
UDP - User Datagram Protocol
  Source Port:  13  Daytime
  Destination Port: 5000
  Length:      37
  Checksum:    0x0522
  UDP Data Area:
  11 APR 2002 10:5      31 31 20 41 50 52 20 32 30 30 32 20 31 30 3a 35
  8:54 METDST..      38 3a 35 34 20 4d 45 54 44 53 54 0d 0a
Frame Check Sequence: 0xffffffff
```

Grafikprogrammierung

Koordinatensystem (Fenster oder gesamter Schirm)



Farbdarstellung RGB

RGB(0,0,0) – Schwarz
RGB(255,0,0) – Rot
RGB(0,255,0) – Grün
RGB(0,0,255) – Blau
RGB(255,255,255) – Weiss
RGB(f,f,f) – Graustufen (f (0..255))

Grafik Ausgabe mit MFC

```
CClientDC dc(this);
```

Anlegen einer Instanz die einen Device Context verwaltet, der in diesem Fall ein Client eines bestehenden Fensters ist.

```
dc.Rectangle(0,0,513,513);
```

Rechteck zeichnen linke obere Ecke nach rechte untere Ecke.

```
dc.SetPixel(x,y,RGB(255,0,0));
```

Rotes Pixel an P(x,y) setzen.

```
dc.MoveTo(x1,y1);
```

```
dc.LineTo(x2,y2);
```

Zeichenbeginn auf P1(x1,y1) setzen und Linie zu P2(x2,y2) ziehen.

```
CPoint P1,P2;  
  
P1.x=100;  
P1.y=100;  
P2.x=200;  
P2.y=200;  
dc.MoveTo(P1);  
dc.LineTo(P2);
```

Den Funktionen kann auch eine Instanz der Klasse CPunkt übergeben werden.

```
dc.SelectObject(CPen(PS_SOLID,1,RGB(255,0,0)));  
Verändern der Eigenschaften des Stiftes also der Linien.
```

```
dc.SelectObject(CBrush(RGB(255,0,0)));  
Füllfarbe ändern.
```

```
for (int i=0; i<=255; i++)  
    for (int j=0; j<=255; j++)  
        dc.SetPixel(j,i,RGB(j,j,j));
```

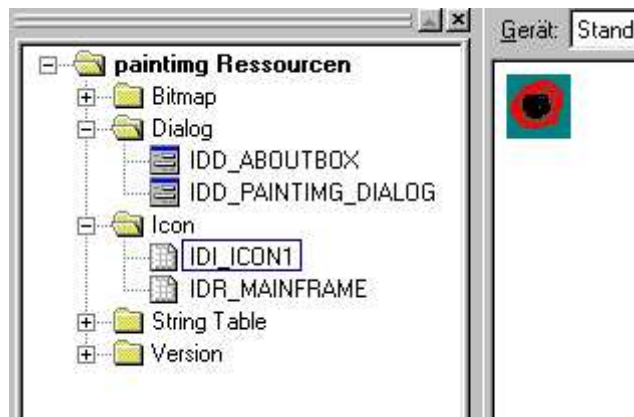
Fläche mit Grauverlauf füllen;

```
for (i=0; i<=255; i++)  
    for (j=0; j<=255; j++)  
        dc.SetPixel(i+1,j+1,RGB(j,i,0));
```

Fläche mit Farbverlauf füllen;

```
point.x-=16;  
point.y-=16;
```

```
dc.DrawIcon(point,AfxGetApp()->LoadIcon(IDI_ICON1) );  
Icon laden und zeichnen.
```



Die Maus

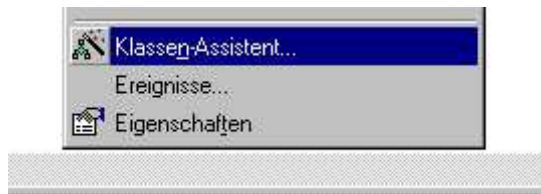
Jede Bewegung der Maus hat eine Nachricht zur Folge. Wenn die Nachricht an eine Anwendung(Fensterbereich) weitergeleitet werden muß, so geschieht dies automatisch.

Die wichtigsten Nachrichten:

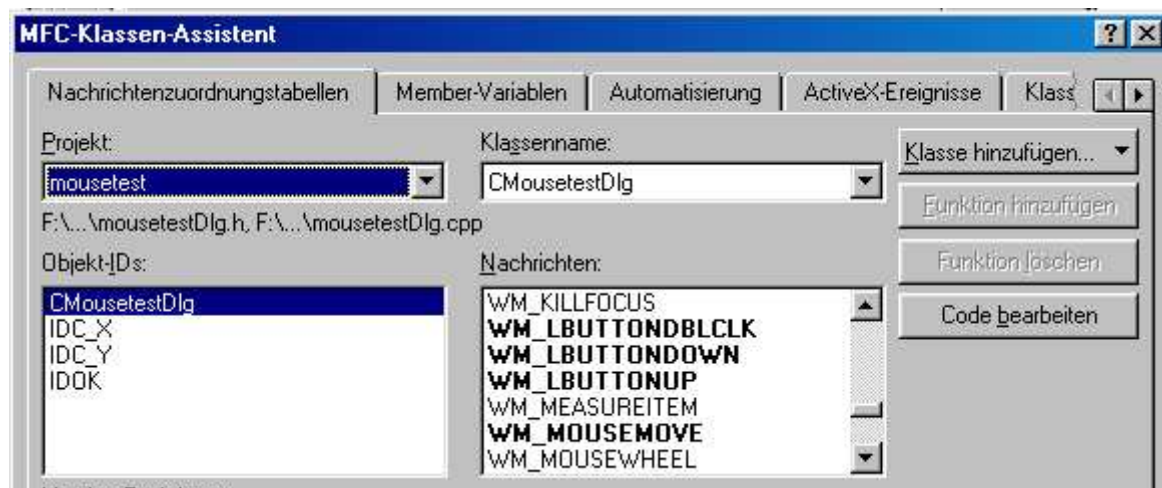
Bewegung:	ON_WM_MOUSEMOVE()
Linker Button gedrückt:	ON_WM_LBUTTONDOWN()
Linker Button 2x gedrückt:	ON_WM_LBUTTONDBLCLK()
Linker Button losgelassen:	ON_WM_LBUTTONUP()

Behandlungsroutinen einfügen:

Rechte Maustaste Klassen-Assistent aufrufen:



Objekt-ID des Dialoges auswählen. Es erscheinen die zugeordneten Nachrichten: Die gewünschte Nachricht wird ausgewählt und der Button Funktion hinzufügen wird gedrückt. Über Code Bearbeiten kann man sofort mit der Bearbeitung beginnen.



Beispiel Doppel Klick Funktion:

```
void CMousetestDlg::OnLButtonDblClk(UINT nFlags, CPoint point)
{
    CClientDC dc(this);
    dc.Ellipse(point.x,point.y,point.x+5,point.y+5);

    CDialog::OnLButtonDblClk(nFlags, point);
}
```

Zufallszahlen

C bietet eine Funktion um „zufällige“ Zahlenfolgen zu erzeugen.
Die Funktion rand() liefert einen ganzzahligen Wert zwischen 0 und RAND_MAX.

Um nicht immer die gleiche Zahlenfolge zu erhalten kann die Startzahl festgelegt werden.

Eine Möglichkeit ist die Eingabe der Uhrzeit:

```
srand( (unsigned)time( NULL ) );
```

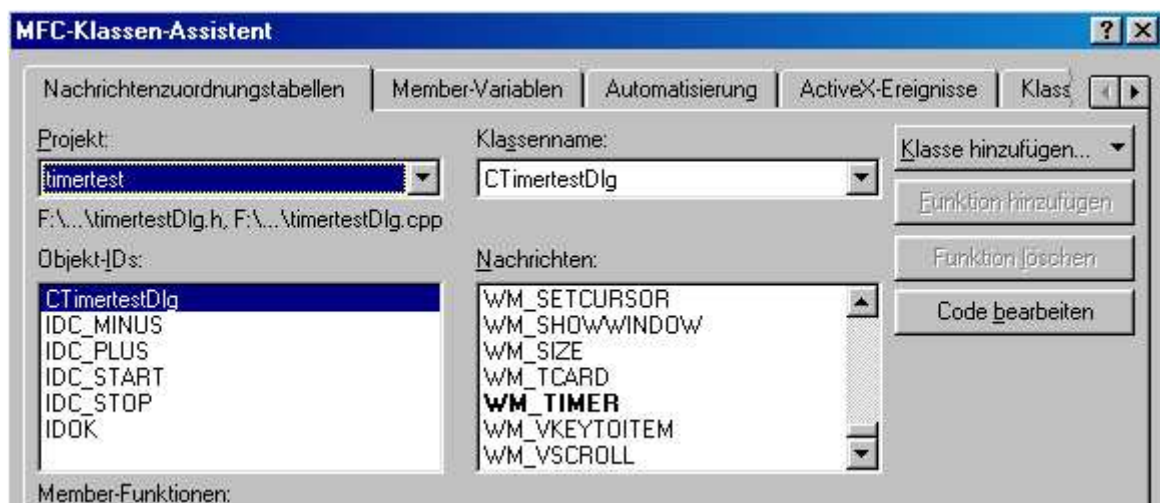
Zufallszahl auslesen:

```
i = rand();
```

Timer

Ein Timer wird durch einen Rückwärtszähler realisiert. Jedesmal wenn der der Vorgegebene Startwert bei Null angekommen ist wird eine WM_TIMER Nachricht geschickt und der Zähler beginnt wieder von vorne.

Für die WM_TIMER Nachricht kann mit Hilfe des Klassenassistenten eine Funktion hinzugefügt werden.



Timer starten oder neu initialisieren:

```
SetTimer(1,1000,NULL); // (ID, Zeit in ms, keine Call Back Funktion)
```

Timer anhalten

```
KillTimer(1); // (ID)
```

Beispiel Funktion:

```
void CTimertestDlg::OnTimer(UINT nIDEvent)
{
    MessageBeep(0xFFFFFFFF); //0xFFFFFFFF Klick auf Systemlautsprecher
    CDialog::OnTimer(nIDEvent);
}
```

Dateien schreiben und lesen

```
void CDateiDlg::OnSchreiben()
{
    CFile datei;
    if(!datei.Open("test.txt",CFile::modeCreate|CFile::modeNoTruncate|
        CFile::modeReadWrite,NULL))
    {
        MessageBox("geht nicht");
        return;
    }
    else
    {
        datei.SeekToEnd();

        UpdateData(true);
        datei.Write(m_eingabe, m_eingabe.GetLength());
        datei.Write("\r\n",2);
        datei.Close();
    }
}
```

```
void CDateiDlg::OnLesen()
{
    CFile datei;
    char cin;
    CString ein;

    if(!datei.Open("test.txt",CFile::modeRead))
    {
        MessageBox("geht nicht");
        return;
    }
    else
    {
        datei.SeekToBegin();
        m_clist1.ResetContent();
        cin=0;
        ein=" ";

        while(datei.Read(&cin,1))
        {
            if(cin!='\r')
                ein=ein+cin;
            else
            {
                datei.Read(&cin,1);
                m_clist1.AddString(ein);
                ein=" ";
            }
        }
    }
}
```