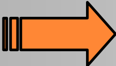
Zeiger

In C++ werden Zeiger (Pointer) sehr viel häufiger eingesetzt, als in anderen Programmiersprachen. Oft ist die Verwendung von Zeigern nicht unbedingt erforderlich, z.B. **um auf die Elemente eines Vektors zuzugreifen**, da dies auch mit Hilfe der bekannten Vektor-Schreibweise möglich ist.
Zeiger werden ferner eingesetzt, **um Argumente an eine Funktion zu übergeben**, wenn die Argumente von dieser Funktion verändert werden sollen (wie bei Referenz-Übergaben), wobei die Übergabe von Vektoren fast ausschließlich mit Hilfe von Zeigern erfolgt.
Die Bedeutung der Zeiger liegt jedoch in der Möglichkeit der Erzeugung von Datenstrukturen wie **verbundene Listen und Binärbäume**, sowie in der Möglichkeit der **interaktiven Speicherplatzanforderung**, was die Leistungsfähigkeit der Programmiersprache erheblich steigert.

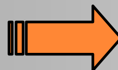
Copyright © 1998 Dagmar Walddobler

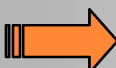
Zeiger

Einsatz von Zeigern:

 **- um auf die Elemente eines Vektors zuzugreifen**

- um veränderbare Argumente an eine Funktion zu übergeben

 **- bei dynamischer Speicherplatzverwaltung**

 **- um verkette Listen und Binärbäume zu erzeugen**

Copyright © 1998 Dagmar Walddobler

Zeiger

Jedes Byte im Speicher hat eine eindeutige Adresse, d.h. daß jede Variable bzw. Datenstruktur an einer bestimmten Adresse beginnt. Die Adresse einer Variablen kann man mit Hilfe des **Adressoperators &** bestimmen.

Beispiel:

printf ("Die Variable %d hat die Adresse %p", var, &var)

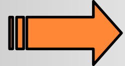
Das Formatsteuerzeichen **%p** gibt die Adresse der Variablen var (mit **&**var) aus.

Die Adressen lokaler (automatischer) Variablen erscheinen in absteigender Reihenfolge, weil sie im Stapelspeicher (Stack) angelegt werden, während die Adressen globaler (externer) Variablen aufsteigend sind, da sie im Heap angelegt werden.
Sinnvoll ist nun die Speicherung solcher Adressen (= Zeigerkonstante) in Variablen, den sog. Zeigervariablen.

Copyright © 1998 Dagmar Walddobler

Zeiger

Was ist ein Zeiger:



eine Variable, welche die Adresse von Variablen speichert !

Beispiel:

int zahl ;
int * p;
zahl = 758;
p = &zahl;

Variable zahl mit Inhalt 758

@e 8A2E | 758

Zeigervariable p beinhaltet die Adresse der Variablen zahl, nämlich 8A2E

8A2E

Variable zahl hat im Speicher die Adresse Hex **8A2E**

Copyright © 1998 Dagmar Walddobler

Zeiger

Jedes Byte im Speicher hat eine eindeutige Adresse, d.h. daß jede Variable bzw. Datenstruktur an einer bestimmten Adresse beginnt. Die Adresse einer Variablen kann man mit Hilfe des **Adressoperators &** bestimmen.

Beispiel:

printf ("Die Variable %d hat die Adresse %p", var, &var);

Das Formatsteuerzeichen **%p** gibt die Adresse der Variablen var (mit **&**var) aus.

Die Adressen lokaler (automatischer) Variablen erscheinen in absteigender Reihenfolge, weil sie im Stapelspeicher (Stack) angelegt werden, während die Adressen globaler (externer) Variablen aufsteigend sind, da sie im Heap angelegt werden.
Sinnvoll ist nun die Speicherung solcher Adressen (= Zeigerkonstante) in Variablen, den sog. Zeigervariablen.

Copyright © 1998 Dagmar Walddobler

Zeiger

Deklaration einer Zeigervariablen

datentyp * variablenname ;

Beispiele:

```
int *  pi ;
float * px, *py ;
```

Der **Stern** bedeutet **Zeiger auf** eine entsprechende Variable (im Beispiel ist ***pi*** eine Variable, die auf eine Integer-Variable zeigt und ***px*** sowie ***py*** Zeigervariablen, die Adressen von *float*-Variablen speichern.

Der Compiler muss wissen, auf welche Art von Daten eine Variable zeigt, um bestimmte Operationen mit dieser Zeigervariablen durchführen zu können. Eine Zeigervariable kann somit nur die Adresse einer Variablen vom angegebenen Datentyp speichern.

Man kann Zeiger auf jeden beliebigen Datentyp deklarieren.

Copyright © 1998 Dagmar Walddobler

Zeiger

Nach der lokalen Deklaration einer Zeigervariablen enthält diese irgendeinen zufälligen Wert, d.h. die zufällige Adresse von irgend einem Speicherplatz im Hauptspeicher, wodurch es zu einem Systemabsturz kommen kann, wenn man sie verwendet ohne sie vorher zu initialisieren.

Daher ist es wichtig, eine Variable wie folgt zu initialisieren:

```
int * intzeiger;
...
intzeiger = NULL;
```

NULL ist eine vordefinierte Adresskonstante und entspricht dem Wert 0.
Eine Zeigervariable mit dem Inhalt ***NULL*** zeigt auf keine gültige Speicherplatz-adresse.

Copyright © 1998 Dagmar Walddobler

Zeiger

Dereferenzieren eines Zeigers:

Man kann nun über einen Zeiger auf den Inhalt einer Variablen zugreifen, ohne ihren Namen zu kennen, mit Hilfe des Inhaltsoperators:

```
*intzeiger  = 10;
```

Wenn also vor der Zeigervariablen ein Stern steht, so ist damit nicht der Inhalt (=Adresse) gemeint, sondern der Wert der Variablen, auf die gezeigt wird. Man spricht hier von **indirekter Adressierung**. Über den Inhaltsoperator kann man so jede beliebige Operation mit der Variablen selbst ausführen (ohne ihren Namen zu kennen !), wie folgendes Beispiel zeigt :

Copyright © 1998 Dagmar Walddobler

Zeiger

```
void main( )
{
        int  a, b;
        int  * intzeiger;
        intzeiger    =  &a;
        *intzeiger  =  10;
        b   =  *intzeiger;
        printf (" Variable  a : %d   Variable  b : %d", a, b);
}
```

Welche Ausgabe erzeugt das Programm ?

Copyright © 1998 Dagmar Walddobler

Zeiger

Zeiger auf Vektorelemente:

Man kann nun auch mit der Zeigerschreibweise auf die Elemente eines Vektors zugreifen. Der Vektorname steht für die Anfangsadresse des Vektors . Das folgende Beispiel stellt den herkömmlichen Vektor-Zugriff und den Zugriff mit Hilfe eines Zeigers gegenüber :

Zugriff mit Vektor-Schreibweise :

```
void main()
{           int   feld[] = {14, 8, 12, 23, 45, 11};
            int  i;
            .
            .
            for ( i=0 ; i<6 ; i++)
            printf ( "%d .tes Feld-Element  : %d \n ",i, feld[i]);
            }
```

Copyright © 1998 Dagmar Walddobler

Zeiger

Zugriff mit Zeiger :

```
void main( )
{           int   feld[ ] = {14, 8, 12, 23, 45, 11} ;
            int  i;
            int *  intzeiger ;
            .
            .
            intzeiger = feld;
            for ( i=0 ; i<6 ; i++)
            printf ( "%d .tes Feld-Element  : %d \n “
                        , i , *intzeiger++ );
}
```

Copyright © 1998 Dagmar Walddobler

Zeiger

Alternativ könnte man auch den Feldnamen als Zeigerkonstante einsetzen:

```
printf ( "%d .tes Feld-Element  : %d \n ",i, *(feld + j ) );
```

Der Ausdruck ***(feld+j)*** bewirkt das folgende :

Für j = 4 beispielsweise wird die Adresse ***feld*** nicht um 4 (Byte), sondern um 8 (Byte) erhöht, da ***feld*** aus Integer-Variablen besteht und eine Integer-Variable 2 Byte lang ist. Der Compiler erkennt also selbständig bei Adress-Operationen an Hand des Datentyps um welchen Wert eine Adresse verändert werden muss . Mit dem Inhalt-Operator * wird der Inhalt des Elements ausgegeben.
Der Ausdruck ****(feld++)*** wäre nicht erlaubt, da ***feld*** eine Adresskonstante ist, die, wie jede andere Konstante, nicht verändert werden kann.

Copyright © 1998 Dagmar Walddobler

Zeiger

Zeiger als Funktionsparameter

Soll eine Funktion eine Variable des aufrufenden Programms verändern, so kann man sie als Referenz übergeben.
Alternativ dazu kann man einen Zeiger übergeben, der ihre Adresse beinhaltet.
In den folgenden Beispielen wird beides gegenübergestellt :

Copyright © 1998 Dagmar Walddobler

Zeiger

Beispiel eines Funktionsaufrufs mit Referenz-Übergabe :

```
void bogenmass (float&);      //  Prototyp: Funktion mit Referenz-Übergabe

void main( )
{           float  winkel;
            ...
            printf ("Geben Sie Gradmaß ein : ");
            scanf("%f",&winkel);
            bogenmass (winkel);
            printf("umgerechnet in Bogenmaß : %f",winkel);
}

void bogenmass (float& w)
{
            w    =  M_PI/180;
 }
```

Copyright © 1998 Dagmar Walddobler

Zeiger

Funktionsaufruf mit Zeigerübergabe:

```
void bogenmass (float * ) ;      //  Prototyp :   Funktion mit Zeigerübergabe

void main( )
{           float  winkel;
            ...
            printf ("Geben Sie Gradmaß ein : ");
            scanf("%f",&winkel);
            bogenmass (&winkel);      // Übergabe der Variablenadresse
            printf("umgerechnet in Bogenmaß : %f",winkel);
}

void bogenmass(float*  w)
{
            *w     =  M_PI / 180;
}
```

Copyright © 1998 Dagmar Walddobler

Zeiger

Die Funktion *bogenmass()* hat einen ***float*-Zeiger** als Argument:

bogenmass(float *) ;

daher muss beim Aufruf die Adresse der Variablen *winkel* übergeben werden:

bogenmass (&winkel);

Die Funktion *bogenmass* greift nun mit Hilfe des Inhalt-Operators auf die Variable zu:

***w = M_PI / 180;**

Copyright © 1998 Dagmar Walddobler

Zeiger

Übergabe von Vektoren mit Zeigerschreibweise

Es ist üblich bei der Übergabe eines Vektors an eine Funktion die Zeiger-Schreibweise zu verwenden, d.h. seine Anfangsadresse zu übergeben.

Das folgendes Beispiel soll dies demonstrieren :

Copyright © 1998 Dagmar Walddobler

Zeiger

```
void bogenmass(float*);      //  Prototyp :   Funktion mit Zeigerübergabe

void main()
{           float  winkel [ 6 ] ;
            ...
            for( int i=0 ; i<6 ; i++)
            {           printf ("Geben Sie die Winkel in Gradmaß ein : ");
                        scanf("%f",&winkel[i]);
            }

            bogenmass ( winkel );      //  Übergabe der Anfangsadresse
            for (i=0; i<6; i++)
            printf("%d.ter Winkel  :%8.1f\n",i+1, winkel[i] );
}
void bogenmass( float*  w)              // Funktionsdefinition
{
            for (int i=0 ; i<6 ; i++)
            *(w ++ )   =  M_PI / 180;
}
```

Copyright © 1998 Dagmar Walddobler

Zeiger

Beim Aufruf der Funktion *bogenmass* braucht der Adressoperator dem Vektornamen nicht vorangestellt zu werden, da dieser Name ja schon für die Anfangsadresse steht!
Um nun in der Funktion nacheinander auf die Elemente des Vektors zuzugreifen, wird bei jedem Schleifendurchgang die Zeigervariable um eine float-Einheit erhöht (*w*++).

Copyright © 1998 Dagmar Walddobler

Zeiger

Erlaubte Arithmetik mit Zeigern

Folgende arithmetische Operationen sind mit Zeigern **erlaubt** :

- Addition einer *int*-Konstanten bzw. *int*-Variablen
- Subtraktion einer *int*-Konstanten bzw. *int*-Variablen
- Subtraktion zweier Zeiger
- Vergleiche zweier Zeiger mit den bekannten Vergleichsoperatoren

nicht erlaubt sind :

- Addition, Division und Multiplikation von Zeigern,
- logische Operationen mit Zeigern.

Copyright © 1998 Dagmar Walddobler

Zeiger

Dynamische Speicherplatz-Verwaltung

Bei der Deklaration eines Vektors muss man dem Compiler die Größe des Vektors mit einer Konstanten mitteilen. Die Angabe einer variablen Feldgröße ist in C nicht erlaubt!
Dadurch wird sehr häufig Speicherplatz verschwendet, da man ja größtmöglichen Platz freihalten muss, der oft aber nicht ganz benutzt wird.

Die Programmiersprache C++ bietet jedoch die Möglichkeit, mit dem Operator ***new*** während des Programmlaufs Speicherplatz anzufordern, dann wenn er wirklich erforderlich ist.

(In C gibt es eine entsprechende Funktion: *alloc( )*)

Copyright © 1998 Dagmar Walddobler

Zeiger

Beispiel:

```
int *  intzeiger;
.
.
intzeiger = new  int [ 10 ] ;
```

Der Operator ***new*** liefert nun als Ergebnis einen Zeiger auf den Anfang eines Speicherbereiches, der 10 Variablen des Typs *int* aufnehmen kann.
Zu beachten ist hierbei, dass Zeigertyp und Variablentyp überein-stimmen müssen!

Copyright © 1998 Dagmar Walddobler

Zeiger

Speicherplatz, der mit ***new*** angefordert wurde, kann (und sollte auch) während des Programmlaufs wieder freigegeben werden!
Dazu verwendet man den Operator ***delete*** :

delete intzeiger;

In diesem Zusammenhang sei folgendes erwähnt :

Der Operator ***delete*** löscht nicht den Zeigerinhalt! Die Adresse bleibt also in der Zeigervariablen bestehen, sie hat aber keine Gültigkeit mehr, d.h. der Speicherplatz wird für andere Daten verwendet!
Man sollte daher nach einer Speicherplatz-Freigabe mit ***delete*** den entsprechenden Zeiger nicht mehr verwenden !

Copyright © 1998 Dagmar Walddobler

Zeiger

Mit Hilfe des Operators ***new*** kann man nun eine sog. **verkettete Liste** erstellen.
Dabei wird für ein Datenelement erst während der Laufzeit Speicher zur Verfügung gestellt und die Anfangsadresse dieses neuen Datenelements im zuletzt gespeicherten Datenelement vermerkt.
Die so angelegten Datenelemente enthalten daher eine Zeigervariable auf ein solches Element und werden über diesen Zeiger miteinander verbunden (verkettet).
Im Vergleich zu den Vektorelementen sind diese Datenstrukturen im Speicher nicht aufeinanderfolgend, sondern können überall verstreut sein !

Copyright © 1998 Dagmar Walddobler

Zeiger

Zeiger auf Strukturen

Bisher verwiesen die Zeiger nur auf vordefinierte Datentypen, man kann aber auch Zeiger auf selbst-definierte Datenstrukturen deklarieren.

zum Beispiel :

```
struct person
{       char  name[20];
        char  vorname[15];
        float  gehalt;
 }  mitarbeiter ;

person  *person_zeiger;
```

Copyright © 1998 Dagmar Walddobler

Zeiger

Man kann jetzt der definierten Zeigervariablen die Anfangsadresse der Strukturvariablen *mitarbeiter* zuordnen :

person_zeiger = &mitarbeiter;

Die Variable *person_zeiger* verweist nun auf den Anfang der Struktur ***mitarbeiter***.

Um nun auf den Inhalt einer Strukturkomponenten zugreifen zu können, verwendet man den Pfeil-Zugriff :

person_zeiger -> gehalt = 4580;

oder: ***scanf("%f", &person_zeiger -> gehalt);***

Ebenso möglich, aber äußerst unüblich ist die Schreibweise :

**(person_zeiger).gehalt = 4580;*

Copyright © 1998 Dagmar Walddobler

Zeiger

far-Zeiger

Ein ***near***-Zeiger (Voreinstellung) speichert eine 2-Byte Adresse, welche die Offset-Adresse, bezogen auf das aktuelle Datensegment (abhängig vom Speichermodell), darstellt.
Will man jedoch auf Adressen eines anderen Segments zugreifen, z.B. auf Systemvariablen, so muss man ***far***-Zeiger deklarieren, die eine 4-Byte Adresse speichern in der Form 0x0430F0BE.
Dabei stellen die ersten 4 Ziffern (0430) die Segment-Adresse und die letzten 4 Ziffern (F0BE) die Offset-Adresse dar.

Copyright © 1998 Dagmar Walddobler