LANDESBERUFSSCHULE 4 SALZBURG

Informatik

Zeiger (Pointer)

LBS 4

Dieses Skript dient als zusätzliche Lernunterlage für Informatik

Inhalt

Zeiger (Pointer):

Jeder Variable, die wir definieren wird

- ein Name
- ein Wert und
- eine RAM-Adresse zugewiesen.

Die RAM-Adresse gibt an, wo die Variable im Speicher zu finden ist.

Das RAM ist byteweise organisiert. Das erste Byte hat die RAM-Adresse 0. D.h. jede Variable im Speicher hat auch eine Adresse.

Bsp.:

int x = 5;

Adresse:	Name:	Wert:
$233112	x	5

Wir können in C Variablen verwenden, die ausschließlich zur Speicherung von RAM-Adressen dienen. Solche Variablen werden **Zeigervariablen** genannt.

Wenn Variablen zur Speicherung der Adresse einer anderen Variablen verwendet werden, spricht man von einer Zeigervariablen (kurz. Einem Zeiger).

Warum werden Zeiger verwendet?

Es gibt 2 Gründe, warum man Zeiger verwenden sollte:

- **Zugriff auf lokale Variablen gewähren.**
  Eine Funktion kann an die aufrufende Funktion mittels return nur einen Wert zurückgeben. Sollten mehrere Werte zurückgegeben werden, müsste man diese Variablen global definieren. Dies hat aber den gravierenden Nachteil, dass alle Funktionen nun auf die Variablen zugreifen können.

  Eine sehr gute Lösung des Problems ist die Methode des Call-By-Reference. Dabei übergibt man an die Funktion nicht die Werte der Variablen, sondern die Adressen der Variablen im Speicher. Also, wo die Variablen im Speicher liegen.

- **Die Größe von Arrays** muss bereits zur Übersetzungszeit angegeben werden. Dies kann sehr oft zu Problemen führen, weil entweder zu viel angegeben wird (Speicherverschwendung) oder zu wenig. In diesem Fall kann oftmals das ausgelieferte Programm seine Aufgabe nicht erfüllen.

  Zur Lösung dieser Problematik hat man die so genannte **Dynamische Speicherverwaltung** eingeführt. Dies bedeutet, dass erst zur Laufzeit des Programmes der tatsächlich benötigte Speicher angefordert werden muss. Die Funktionen zur dynamischen Speicherverwaltung liefern die Ram-Adresse des nun zur Verfügung stehenden Speichers.

Operatoren für Zeiger

Es gibt drei Operatoren für Zeiger:

- eine Zeigervariable definieren
- die Adresse eines Zeigers bestimmen
- einen Zeiger dereferenzieren

Zeiger deklarieren *

Bsp.:

```
int *a
float *c
double *b
```

Adressoperator &

Um einer Zeigervariablen eine Adresse zuzuweisen wird der Adressoperator benötigt.

Bsp.:

```
#include <stdio.h>

int main(){
   int x=5;
   int *ptr;       //Zeiger deklarieren
   ptr= &x;       //Adresse von x zuweisen
   printf("Die Adresse von ptr ist %p \n", ptr);     //Adresse der Variable x ausgeben
   printf("Die Adresse von x ist %p \n", &x);        //Adresse der Variable x ausgeben
   return 0;
}
```

Der Platzhalter %p wird für printf() zum Anzeigen der Adresse benötigt.

Dereferenzierungsoperator *

Man spricht von einer so genannten indirekten Adressierung, wenn man nicht direkt auf die Variable zugreift, sondern den Wert einer Zeigervariablen verwendet, um auf den Inhalt der Variablen zuzugreifen. (auch Dereferenzierung genannt)

```
#include <stdio.h>

int main(){
   int x=5;
   int *ptr;    // Zeigervariable

   ptr= &x;      // Zeigervariable erhält die Adresse der Variablen x
```

```
   printf("Die Adresse von x ist %p \n",ptr); //Adresse ausgeben
   printf("Die Variable x hat den Wert %d \n", *ptr); //dereferenzieren und ausgeben

   *ptr= 17;     // mittels der Zeigervariablen den Wert von x ändern

   printf("x hat den Wert %d \n", *ptr);
   printf("x hat den Wert %d \n", x); // Wert von x = 17

   return 0;
}
```

Bisher wurden die verschiedenen Zeigeroperatoren gezeigt. Als nächstes wird die Verwendung von Zeigern näher erklärt.

Wertübergabe von Parametern (Call by Value)

Wenn an Funktionen Variablen übergeben werden, so werden diese **kopiert**. Die aufgerufene Funktion arbeitet dann mit den Wertekopien (=Parametern). Beim Verlassen der Funktion werden alle lokalen Variablen (= Kopien der Variablen des Aufrufers) zerstört.

Als Beispiel soll eine Funktion zeigen (Swap), wie der Inhalt zweier Variablen austauscht wird. Solch eine Funktion wird häufig beim Sortieren von Daten benutzt.

Beispiel: Swap (funktioniert nicht)

```
// call by value
#inlcude <stdio.h>
void swap (int , int );

int main(){
        int a= 11;
        int b= 99;

        printf("\nVor dem Aufruf von swap(): a=%d und b=%d\n", a,b);
        swap (a,b);
        printf("\nNach dem Aufruf von swap(): a=%d und b=%d\n", a,b);

        return 0;
}

//soll den Wert der Parameter tauschen
void swap (int x, int y){
        int help; // Hilfsvariable
        help= x;
        x= y;
        y= help;
}
```

Wenn Sie diese Version ausprobieren, werden Sie sehen, dass der Wert von a und b nicht getauscht wurde.

Es wurden die Kopien von a und b (in der Funktion swap() mit x und y bezeichnet) ausgetauscht. Die Originale (a und b) blieben unberührt!

Problem: Wie schaffen wir es, die Originale –also Variablen, die dem Aufrufer gehören- auszutauschen?

Referenzübergabe von Parametern (Call-By-Reference)

Wenn Sie an die Funktion swap() die Adressen von a und b übergeben und swap() so umschreiben, dass sie mit Zeigervariablen arbeitet, werden die Originalwerte getauscht.

Beispiel: Swap (funktioniert)

```
// call by reference
#inlcude <stdio.h>
void swap (int *, int *);

int main(){
        int a= 11;
        int b= 99;

        printf("\nVor dem Aufruf von swap(): a=%d und b=%d\n", a,b);
        swap (&a, &b); //Adressen der Variablen übergeben
        printf("\nNach dem Aufruf von swap(): a=%d und b=%d\n", a,b);

        return 0;
}

//Werte werden getauscht
void swap (int *x, int *y){
        int help; // Hilfsvariable
        help = *x;
        *x = *y;
        *y = help;
}
```

Die Parameter der Zeigervariablen werden zwar auch kopiert, aber mittels des Dereferenzierungsoperators (*) wird auf die Originale zugegriffen.

Arrays uns Zeiger:

Das erste Feld eines Arrays ist auch die Adresse des Arrays oder der Arrayname ist die Adresse des ersten Arrayelements

Zum Dereferenzieren bei Arrays wird entweder **[ ]** oder ***** verwendet!

Beispiel: Arrays

```
#include <stdio.h>

int main(){
 int element[8]= {1,2,4,8,16,32,64,128};
 int *ptr;

 ptr= &element[0}; //ptr zeigt auf das erstes Element: Index 0

 printf("Der Wert auf den *ptr zeigt ist %d\n", *ptr ); //Ausgabe des Wertes
 return 0;
}
```

Die verschiedenen Schreibweisen bei Arrays sind nachstehend noch einmal aufgelistet. (int *ptr ist deklariert)

1. Schreibweise:
   ptr = &element[0];
2. Schreibweise
   ptr = element:

Beides mal zeigt ptr auf das erste Elemet des Arrays!

Zeigerarithmetik

Zeiger beinhalten nichts anderes als Ganzzahlwerte (Speicheradressen). Mit diesen kann man rechnen oder sich in einem Array bewegen. Voraussetzung ist immer, dass man die Arraygrenzen kennt.

Beispiel: Array

```
#include <stdio.h>

int main(){
  int i;
  int element[8]= {1,2,4,8,16,32,64,128};
  int *ptr;

  ptr= &element[0]; // zeigt auf erstes Element: Feldindex 0

  printf("Adresse=%p :Wert=%d\n", ptr+1,*(ptr+1)); // Zeiger wird um eins erhöht
```

```
  printf("Adresse=%p :Wert=%d\n", &ptr[1],ptr[1]); //Zeiger zeigt auf Element 1

  printf("\nJetzt alle zusammen : \n");

  for(i=0; i<8; i++)
     printf("element[%d]=%d\n",i, *(ptr+i) );

  return 0;
}
```

Vorsicht!

*ptr +2 ist **nicht das gleiche** wie *(ptr + 2)

*ptr +2 //ptr wird dereferenziert und 2 addiert

*(ptr +2) //ptr wird um 2 Stellen weitergezählt und dann dereferenziert