Zeiger

C-Kurs 2012, 2. Vorlesung

Tino Kutschbach
tino.kutschbach@campus.tu-berlin.de

`http://wiki.freitagsrunde.org`

10. September 2012

Inhaltsverzeichnis

Inhaltsverzeichnis

Speicherzellen, Adressen, Inhalte

- Der Arbeitsspeicher besteht aus vielen einzelnen Speicherzellen
- Jede Speicherzelle besitzt eine eindeutige Adresse und speichert einen Inhalt
- Bei Deklaration einer Variable wird eine Speicherzelle reserviert und mit einem Namen versehen

Direkter und indirekter Zugriff

- Direkter Zugriff per Name
  - erfolgt durch Verwendung des deklarierten Namens, wobei direkt auf den Inhalt der verknüpften Zelle zugegriffen wird
  - also direkter Bezug: Name liefert Inhalt
- Indirekter Zugriff per Adresse
  - ermöglicht es auch auf Speicherzellen zuzugreifen, welche keinen Namen haben, bzw. deren Name nicht sichtbar ist
  - erfolgt mit Hilfe von Zeigern.

Inhaltsverzeichnis

Der Adressoperator &

- Der Adressoperator **&** liefert die Adresse einer Variable
  - &x liefert die Adresse der Speicherzelle mit dem Namen x
- Eine Adresse ist eine Zahl und lässt sich wie jede andere Zahl behandeln, z.B in Variablen speichern
- Eine Variable, die eine Adresse enthält, nennt man **Zeiger**

Der Indirektionsoperator *

- Der Indirektionsoperator * nimmt eine Adresse und liefert den Inhalt der entsprechenden Speicherzelle
  - Ist y ein Zeiger auf x, also: Enthält y die Adresse von x (y=&x)
  - $*$y liefert den Wert der in der Speicherzelle mit der Adresse y gespeichert ist, also: $*$y==x
- * ist eine der wenigen Ausnahmen von Operatoren, die auch auf der linken Seite einer Zuweisung stehen können

Zeiger deklarieren und initialisieren

- Eine Zeigerdeklaration weist die folgende Form auf:
- typname ∗zeigername;
  - das bedeutet: der Inhalt der Speicherzelle mit Adresse *zeigername* ist vom Typ *typname*
  - oder anders formuliert: *zeigername* ist ein Zeiger auf den Typ *typname*
- Nach einer Deklaration muss ein Zeiger initialisiert werden, damit er auf eine gültige Speicherstelle zeigt:
- zeigername = &variable;
- **Verwende niemals uninitialisierte Zeiger!**

Einfaches Zeiger-Beispiel

Listing 1: src/zeiger.c

```c
#include <stdio.h>

int main(void) {
    int var = 1;
    int *zgr;    // Deklaration eines Zeigers auf int

    zgr = &var; // Initialisierung: zgr zeigt auf var

    /* Inhalt von var ausgeben */
    printf("Direkter Zugriff, var = %d \n", var);
    printf("Indirekter Zugriff, var = %d \n", *zgr);
    /* Adresse von var ausgeben */
    printf("Adresse von var = %p \n", &var);
    printf("Adresse von var = %p \n", zgr);

    return 0;
}
```

Inhaltsverzeichnis

Datenhaltung lokaler Variablen

- Funktionen hinterlegen ihre lokal deklarierten Variablen auf dem Stack in ihrem zugehörigen Stackframe
- Die Namen der lokalen Variablen einer Funktion sind nur in der Funktion selbst bekannt
  - Ein direkter Zugriff auf Variablen höherer Funktionen, Funktionen mit aufliegendem Stackframe, ist daher nicht möglich
  - Existiert jedoch ein Zeiger auf solche Variablen, so ist ein indirekter Zugriff möglich
- Ruft eine Funktion selbst eine weitere Funktion auf, so werden die Parameter auf dem Stack kopiert und so der Unterfunktion übergeben.

Call-by-value

Listing 2: src/cbv.c

```
1 int addiere(int p1, int p2) {
2   return p1+p2;
3 }
```

- Beim Aufruf der Funktion werden die Parameter in die Funktion hinein kopiert.
- Ein Zugriff auf die Originaldaten ist nicht möglich
- Ein Ergebnis kann nur über *return* zurückgegeben werden

Call-by-reference

Listing 3: src/cbr.c

```
1 void inkrementiere(int *p1, int *p2) {
2   (*p1)++;
3   (*p2)++;
4   return;
5 }
```

- Beim Aufruf der Funktion werden Parameter in die Funktion hinein kopiert, hier sind diese Parameter jedoch Zeiger.
- Ein Zugriff auf Originaldaten ist durch die Zeiger möglich. Es wird direkt mit auf den Originaldaten gearbeitet
- Zusätzlich könnte auch noch über *return* ein Wert zurückgegeben werden

Inhaltsverzeichnis

Array-Anordnung im Speicher

- Die Elemente eines Arrays werden hintereinander in sequentieller Reihenfolge im Speicher abgelegt
- Das erste Element steht an der niedrigsten Adresse. Die folgenden Elemente folgen bei höheren Adressen
- Die Anzahl von Speicherzellen, die ein Element belegt, hängt dabei vom Datentyp des Arrays ab
- Ein Array-Name ohne eckige Klammern ist ein Zeiger auf das erste Element
  - es gilt also: array == &array[0]
  - Bei einem Funktionsaufruf werden Arrays über ihren Namen, ohne Klammern, übergeben. Da der Name nur ein Zeiger ist, ist eine Array-Parameterübergabe immer Call-by-reference

Zeigerarithmetik

- Gibt es einen Zeiger auf das erste Array-Element, kann man durch inkrementieren des Zeigers auf nachfolgende Array-Elemente zugreifen
- Die Größe des Inkrements entspricht der Größe des Datentyps der im Array gespeicherten Elemente
- **Zeiger inkrementieren**:
  - Wird der Wert eines Zeigers um 1 erhöht, so sorgt die Zeigerarithmetik dafür, dass der Zeiger automatisch um die Größe des Datentyps, auf den der Zeiger zeigt, erhöht wird
  - Bei der inkrementierung eines Zeigers auf int um 1: zgr_auf_int ++ wird der Wert automatisch um die Größe des Typs int (4 Bytes) erhöht
  - Bei der inkrementierung eines Zeigers auf double um 10: zgr_auf_double += 10 wird der Wert automatisch um 80 (wenn double 8 Bytes groß) erhöht und rückt somit den Zeiger um 10 Elemente weiter
- **Zeiger dekrementieren** erfolgt analog, wobei hiermit der Zeiger verringert wird und somit auf vorherige Arrayelemente zugegriffen werden kann

Index-Zeigernotation bei Arrays

- Es besteht folgender Zusammenhang zwischen Index- und Zeigernotation bei Arrays
  - $*(\text{array}) == \text{array}[0]$
  - $*(\text{array} + 1) == \text{array}[1]$
  - $*(\text{array} + 2) == \text{array}[2]$
  - ...
  - $*(\text{array} + \text{i}) == \text{array}[\text{i}]$

Inhaltsverzeichnis

Zeiger auf Zeiger

- Zeiger sind selbst numerische Variablen und in eigenen Speicherzellen gespeichert
- Die Anwendung des Adressoperators & auf einen Zeiger ist also ebenfalls gültig
- Was man dann erhält ist ein Zeiger auf einen Zeiger

Listing 4: src/zgr_auf_zgr.c

```
1 int x = 12;                          // x ist eine int-Variable.
2 int *zgr = &x;                       // zgr ist ein Zeiger auf x.
3 int **zgr_auf_zgr = &zgr;            // zgr_auf_zgr ist ein Zeiger
4                                      // auf einen Zeiger auf int
```

- Um einen Zeiger auf einen Zeiger für den Zugriff auf eine Variable zu verwenden muss nun doppelt de-referenziert werden
- Der Indirektionsoperator * muss also doppelt verwendet werden
- Die folgende Anweisung gibt also den Wert von x auf dem Bildschirm aus
  - printf ("%d", **zgr_auf_zgr);

Mehrdimensionale Arrays

- Der Einsatz von Zeigern auf Zeiger wird vor allem bei der Verwendung mehrdimensionaler Arrays deutlich
- Ein Beispiel für ein zweidimensionales Array:
  - `int multi [2][4];`
  - Dieses Array kann man sich als Tabelle mit zwei Zeilen und vier Spalten vorstellen
  - C orientiert es jedoch auf eine andere Art und Weise:
    - *multi* ist ein Zeiger auf ein Array mit 2 Elementen
    - Jedes dieser Elemente (*multi[0]*, *multi[1]*) ist wiederum ein Zeiger auf ein Array mit 4 Elementen
    - Jedes dieser insgesamt acht Elemente ist vom Typ *int*
  - *multi* ist demnach ein Zeiger auf einen Zeiger
  - Es gilt daher die folgende Analogie:
    - $\text{multi}[i][j] == *(*(\text{multi}+i)+j)$

Inhaltsverzeichnis

Zusammenfassung Zeigeroperationen

Operation	**Operator**	**Beschreibung**
Zuweisung	=	Einen Zeiger kann ein Wert zugewiesen werden. Das sollte eine Adresse sein, die von einem Zeiger (Array-Name) kommt oder die durch den &-Operator ermittelt wurde
Indirektion	*	Der Indirektionsoperator * liefert den Wert an der Speicherstelle, auf die der Zeiger verweist
Adresse ermitteln	&	Wird der Adressoperator & auf einen Zeiger angewendet, so liefert dieser die Speicheradresse zurück, an welcher der Zeiger gespeichert ist. So können Zeiger auf Zeiger erzeugt werden
Inkrement	++, +=, +	Ein Zeiger kann um eine ganze Zahl erhöht werden, sodass er auf eine andere Speicherposition verweist. Die Größe des Datentyps wird dabei automatisch mit eingerechnet.
Dekrement	–, -=, -	Ein Zeiger kann um eine ganze Zahl verringert werden, sodass er auf eine andere Speicherposition verweist. Die Größe des Datentyps wird dabei automatisch mit eingerechnet.
Differenzbildung	-	Zwei Zeiger können voneinander subtrahiert werden. So kann festgestellt werden wie viele Elemente die Zeiger voneinander entfernt sind. Die Größe des Datentyps wird dabei automatisch mit eingerechnet.
Vergleich	==, >, !=, <=, >=, <	Zwei Zeiger können miteinander verglichen werden. Dies liefert allerdings nur gültige Ergebnisse, wenn beide Zeiger auf das selbe Array zeigen