HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN · FH
MÜNCHEN

Kapitel 8

Adressen und Zeiger

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 8 – Adressen und Zeiger

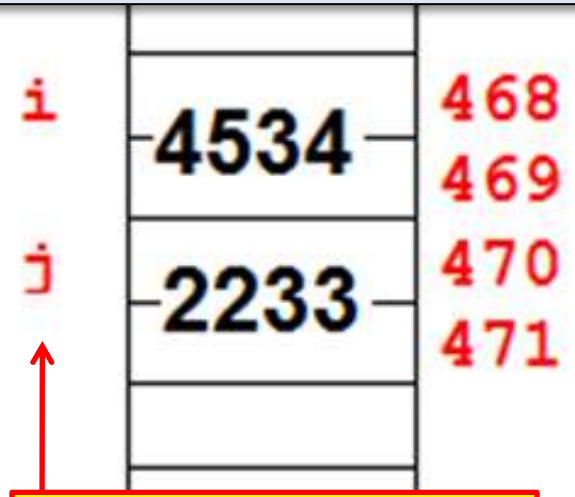
8.1. Definition

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Aufbau des Hauptspeichers:

- Der Hauptspeicher besteht aus einer langen Sequenz von Speicherzellen (jeweils ein Byte groß)
- Jede Speicherzelle besitzt eine eindeutige Nummer/Adresse
- Eine Variable in einem C-Programm belegt entsprechend ihres Typs eine feste Anzahl von Bytes

- *Die Adressen der Variablen sind nicht absolut festgelegt – sie ändern sich zum Beispiel beim Programmneustart*
- *Der Programmierer hat keinen Einfluss darauf, in welchen Speicherzellen die Variablen liegen*



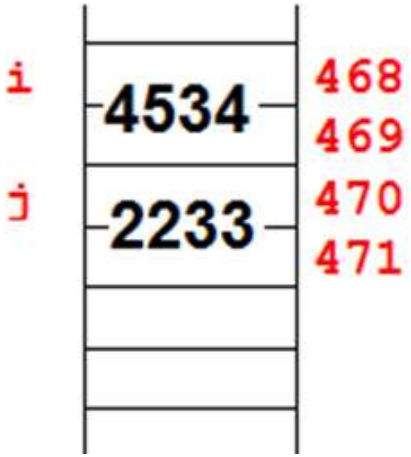
Beispiel: Variablen i, j vom Typ „short“

8.1. Definition

Bisher wurde auf den Hauptspeicher (indirekt) über Variablen zugegriffen. Nun folgt der (direkte) Zugriff auf bestimmte Adressen:

- „Schreib den Wert 123 in die int-Variable an Adresse 1000!“
  „Lies den Wert der short-Variablen an Adresse 1004!“
- Nur die Kombination von Adresse + Variablentyp identifiziert die Speicherzellen einer Variablen eindeutig
- Unter der „Adresse einer Variablen i“ verstehen wir die Adresse der ersten Speicherzelle, wo der Wert von i gespeichert ist

- *Um die Variable i auf 4534 zu setzen, haben wir bisher die Zuweisung „`i = 4534;`“ verwendet*
- *Nun werden wir direkt auf den Hauptspeicher zugreifen: „**Schreib den Wert 4534 in die short-Variable an Adresse 468!**“*



Fragen:

- *Wie erhält man die Adresse einer Variablen?*
- *Wie kann man eine Adresse zwischenspeichern?*
- *Wie liest/beschreibt man eine Variable, deren Adresse bekannt ist?*
- *Welche Vorteile bringt der direkte Zugriff auf den Hauptspeicher?*

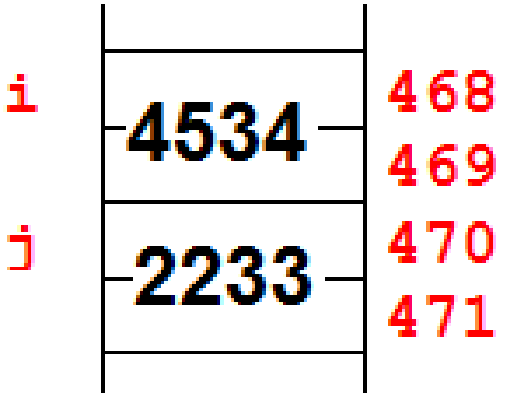
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Wie erhält man die Adresse einer Variablen?

- Ist i der Name einer Variablen, so ist &i die Adresse der Variablen
- Der Operator & ist der sog. „**Adressoperator**“
- Eine Adresse ist grundsätzlich eine ganze Zahl. Eine Adresse ist allerdings keine normale short- oder int-Zahl, man spricht vielmehr von sogenannten „**Zeigern**“

```
short i, j;
i = 4534;
j = 2233;

/*  i hat den Wert 4534      */
/*  &i hat den Wert 468      */
/*  Der Typ von i ist short  */
/*  Der Typ von &i ist "Zeiger auf short"  */
```



8.1. Definition

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Wie kann man eine Adresse zwischenspeichern?

- Zum Abspeichern der Adresse einer short-Variablen benötigt man eine Variable vom Typ „Zeiger auf short“, eine sog. **Zeigervariable**
- Definition einer Zeigervariablen (engl. „Pointer“):

```
short *z;
```

auf welchen Datentyp ***Name der Variablen***

z ist Zeigervariable

- z „zeigt auf“ eine Variable vom Typ short

- *Die Adresse der Variablen i wird in der Zeigervariablen z gespeichert:*

```
z = &i;   /* z erhält Adresse von i zugewiesen */
```

- *Direkt nach der Definition enthält z zunächst einen zufälligen Wert, nach der Zuweisung „zeigt z auf die Variable i“*
- *Zeigervariablen sind wie andere Variablen: Sie besitzen einen Namen und einen Wert und können durch Operatoren verändert werden*

Wie liest/beschreibt man eine Variable, deren Adresse bekannt ist?

- Um auf eine Variable zuzugreifen, deren Adresse bekannt ist, verwendet man den Operator * (sog. „**Inhaltsoperator**“):

```
*z = 17;
```

→ ***Die Zeigervariable z „zeigt auf“ eine bestimmte Variable – schreib in diese Variable den Wert 17!***

```
j = *z;
```

→ ***Die Zeigervariable z „zeigt auf“ eine bestimmte Variable – lies den Wert dieser Variablen und speichere ihn in der Variablen j!***

Beachte: *Der Operator * besitzt verschiedene Bedeutungen:*

- *Multiplikationsoperator* (`x1 = 3 * y1;`)
- *Definition einer Zeigervariablen* (`int *ptr;`)
- *Inhaltsoperator* (`y = *ptr;`)

→ ***Die konkrete Bedeutung ergibt sich jeweils aus dem Zusammenhang!***

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 8 – Adressen und Zeiger

Beispiel 1: Funktion „power“ zur Berechnung von Potenzen; Rückgabe des Ergebnisses mittels Zeiger (und nicht per „return“ oder über eine globale Variable)

```
#include <stdio.h>
void power(float basis, int exp, float *pErg);

int main(void)
{
    float x1, x2, y; int n;
    x1 = 3.0; n = 4;
    power(x1, n, &y); printf("%f\n", y);
    x2 = 3.75;
    power(x2, 5, &y); printf("%f\n", y);
    return 0;
}
```

Beispiel 2: Funktion „qsolve“ zum Lösen quadratischer Gleichungen, Rückgabe **beider** Nullstellen mittels Zeiger (statt über globale Variablen)

```
#include <stdio.h>
#include <math.h>
int qsolve(double p, double q, double *x1, double *x2);
int main(void)
{
    double p, q, null1, null2;
    printf("p eingeben: "); scanf("%lf", &p);
    printf("q eingeben: "); scanf("%lf", &q);
    if(qsolve(p, q, &null1, &null2) != 0)
        printf("Nullstellen: %f, %f\n", null1, null2);
    else
        printf("Keine Nullstellen!\n");
    return 0;
}
```

Beispiel 3:

Das folgende Programm kann ohne Fehler übersetzt werden.
Nach dem Programmstart passiert allerdings Folgendes:

- Manchmal stürzt das Programm ab
- Manchmal arbeitet es scheinbar korrekt
- Manchmal wird ein falscher Wert ausgegeben

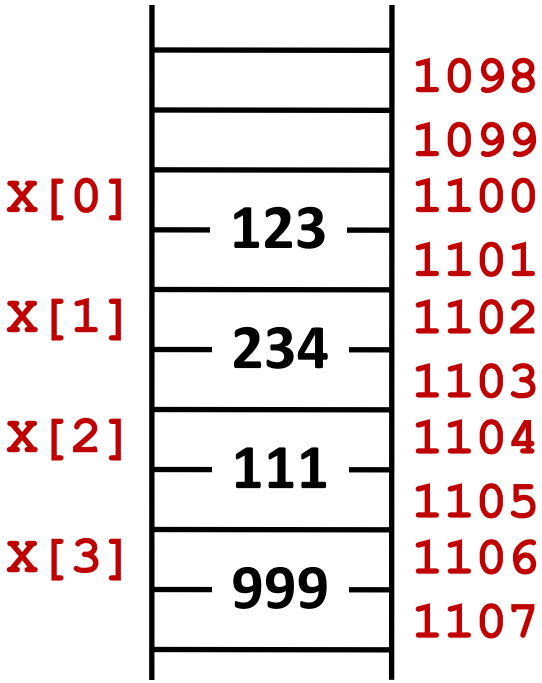
```
#include <stdio.h>
int main(void)
{
    int *z;
    *z = 5;
    printf("*z = %d", *z);
    return 0;
}
```

Wo liegt das Problem?

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 8 – Adressen und Zeiger

Zeigerarithmetik:
Für Zeiger gelten eigene Rechenregeln, die von den arithmetischen Rechenregeln für „normale“ ganze Zahlen abweichen.




```
short x[4];
short *z1, *z2, *z3, *z4;
int i = 3;

z1 = &x[0];  /* z1 zeigt auf x[0] */
z2 = z1 + 1; /* z2 zeigt auf x[1] */
z3 = z1 + i; /* z3 zeigt auf x[i] */

/* Nach folgender Anweisung zeigt */
/* z4 vor den Beginn des Vektors  */
/* x (Adresse 1098) - Vorsicht!!  */
z4 = z1 - 1;
```

8.3. Zeigerarithmetik und Vektoren

Die Zeigerarithmetik ist bei der Arbeit mit Vektoren besonders praktisch. So lassen sich sehr einfach Vektoren „durchlaufen“:

```
int i, x[100];
int *ptr;

for(i = 0; i < 100; ++i)
{
    /* Zeiger auf i-tes Vektorelement: */
    ptr = &x[0] + i;

    /* Ausgabe des i-ten Vektorelements: */
    printf("%d\n", *ptr);
}
```

Beispiel: Im Vektor x sollen die Quadratzahlen von 0^2 bis 99^2 gespeichert werden.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 8 – Adressen und Zeiger

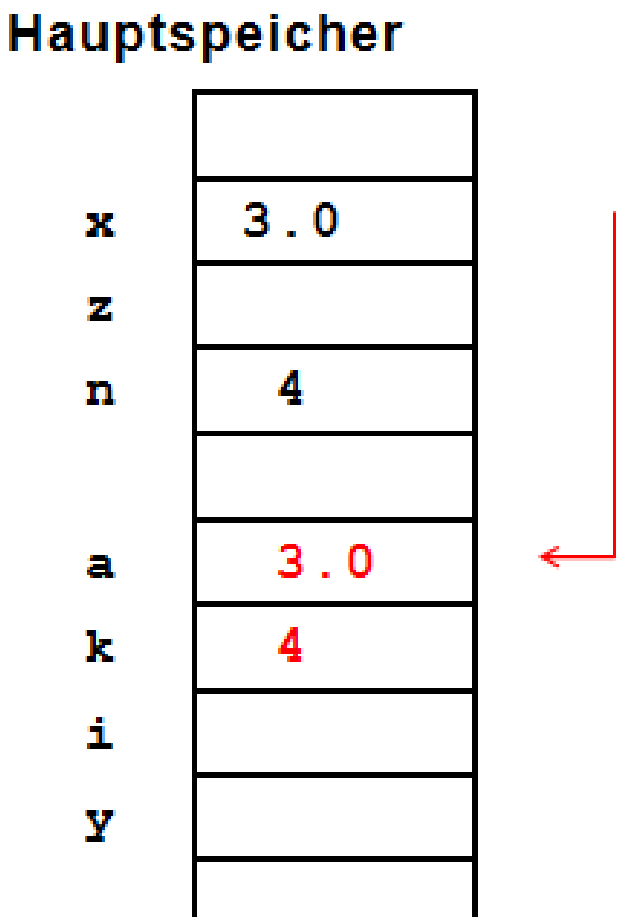
8.4. Vektoren als Funktionsparameter

Wird eine „normale“ Variable als Parameter an eine Funktion übergeben, dann wird stets der Wert der Variablen übergeben. Der Wert der Variablen wird beim Aufruf in einen lokalen Parameter der Funktion kopiert.

```
double power(double a, int k);

int main(void)
{
  double x, z; int n;
  x = 3.0; n = 4;
  z = power(x, n);
  :
}

double power(double a, int k)
{
  int i; double y = 1.0;
  for(i = 1; i <= k; i++)
    y = y * a;
  return y;
}
```



8.4. Vektoren als Funktionsparameter

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Problem:

Die Funktion „quadrat“ soll die ersten n Elemente eines Vektors, der als Parameter übergeben wird, mit Quadratzahlen belegen

```
void quadrat(int *ptr, int n);
int main(void)
{
    int x[100];
    quadrat(&x[0], 50);
    .
    .
    .
}
void quadrat(int *ptr, int n)
{
    int i;
    for(i = 0; i < n; i++)
        *(ptr + i) = i * i;
}
```

Bei der Übergabe eines Vektors an eine Funktion wäre es zu aufwendig, den gesamten Vektor mit all seinen Elementen zu kopieren.

Man übergibt stattdessen einen Zeiger auf das Anfangselement. Die aufgerufene Funktion greift dann mittels Zeigerarithmetik auf den Original-Vektor zu.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 8 – Adressen und Zeiger

Aufgabe:

Bislang verarbeitet das Messwertprogramm eine einzelne Messreihe mit max. 100 Messwerten. Die Messreihe wird in einem globalen Vektor `double x[100]` abgespeichert.

Das Programm soll so erweitert werden, dass eine zweite Messreihe mit ebenfalls max. 100 Messwerten verwaltet werden kann.

8.5. Messwertprogramm, zwei Messreihen

***Variante 1**: Zwei Vektoren x[100], y[100] und separate Funktionen für beide Vektoren. → Die Funktionen mittelwertx, mittelwerty unterscheiden sich nur in einem einzigen Buchstaben…!*

```
#define DIM 100
double x[DIM], y[DIM];
double mittelwertx(int n);
double mittelwerty(int n);
int main(void)
{
    double xmittel, ymittel;
    int anzahlx, anzahly;
    anzahlx = einlesenx();
    anzahly = einleseny();
    xmittel = mittelwertx(anzahlx);
    ymittel = mittelwerty(anzahly);
    return 0;
}
```

```
double mittelwertx(int n)
{
    int i; double z = 0.0;
    for(i = 0; i < n; i++)
        z = z + x[i];
    return z = z / (double)n;
}
double mittelwerty(int n)
{
    int i; double z = 0.0;
    for(i = 0; i < n; i++)
        z = z + y[i];
    return z / (double)n;
}
```

8.5. Messwertprogramm, zwei Messreihen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

***Variante 2**: Nur eine Funktion einlesen, mittelwert, sortiere usw. Der zu bearbeitende Vektor wird als Parameter an die Funktionen übergeben.*

```
int main(void)
{
  double x[SIZE], y[SIZE]; int anz1, anz2;
  printf("Messreihe 1 eingeben...\n"); anz1 = einlesen(&x[0]);
  printf("Messreihe 2 eingeben...\n"); anz2 = einlesen(&y[0]);
  sortiere(&x[0], anz1); sortiere(&y[0], anz2);
  printf("Messreihe 1...\n"); ausgeben(&x[0], anz1);
  printf("Messreihe 2...\n"); ausgeben(&y[0], anz2);
  printf("Mittelwert 1: %.2f\n", mittelwert(&x[0], anz1));
  printf("Mittelwert 2: %.2f\n", mittelwert(&y[0], anz2));
  printf("Maximum 1: %.2f\n", maximum(&x[0], anz1));
  printf("Maximum 2: %.2f\n", maximum(&y[0], anz2));
  return 0;
}
```

Durch die Übergabe als Parameter können die globalen Variablen double x[100] und y[100] vermieden werden!

8.5. Messwertprogramm, zwei Messreihen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

***Beispiel*: Funktionen „ausgeben“ und „maximum“ mit Vektor als Funktionsparameter**

```
/* Alle n Messwerte ausgeben */
void ausgeben(double *mw, int n)
{
    int i;
    for(i = 0; i < n; ++i)
        printf("\tMesswert %d: %.2f\n", i + 1, *(mw+i));
    printf("\n");
}

/* Maximalen Messwert ermitteln und zurueckgeben */
double maximum(double *mw, int n)
{
    double max; int i;
    max = *(mw+0);
    for(i = 1; i < n; i++)
        if(*(mw+i) > max) max = *(mw+i);
    return max;
}
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

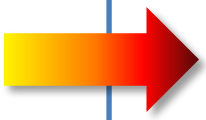
Kapitel 8 – Adressen und Zeiger

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Vektorschreibweise:

Bei der Arbeit mit Zeigervariablen, speziell bei Zeigerarithmetik im Zusammenhang mit Vektoren, können manche Operationen auch anders geschrieben werden (**ohne Änderung der Funktion!**):

```
void fill_n(int *x, int n);
int main(void)
{
    int x[100];
    fill_n(&x[0], 100);
    return 0;
}
void fill_n(int *x, int n)
{
    int i;
    for(i = 0; i < n; ++i)
        *(x+i) = i;
}
```



```
void fill_n(int x[], int n);
int main(void)
{
    int x[100];
    fill_n(x, 100);
    return 0;
}
void fill_n(int x[], int n)
{
    int i;
    for(i = 0; i < n; ++i)
        x[i] = i;
}
```

Zusammenfassung:

```
int x[100];
```

`&x[0]`	oder	`x`	Zeiger auf (bzw. Adresse von) x[0]
`&x[i]`	oder	`x+i`	Zeiger auf (bzw. Adresse von) x[i]
`x[0]`	oder	`*x`	Inhalt von x[0], Wert des Anfangselements
`x[i]`	oder	`*(x+i)`	Inhalt von x[i], Wert des i-ten Elements

```
void func(int *feld);
void func(int feld[]);
```

} Übergabe eines Zeigers an eine Funktion

Es handelt sich wirklich nur um verschiedene Schreibweisen mit derselben Bedeutung. Insbesondere bedeutet die Funktionsdeklaration in der letzten Zeile nicht, dass hier ein kompletter Vektor übergeben wird – auch in diesem Fall handelt es sich um die Übergabe eines Zeigers!