4.1 Datenstrukturen/Zeiger

4.1.1 Hintergrund

Wenn man mehrere Daten eines Programms von einer Funktion aus ändern will, braucht man dazu in C Zeiger- bzw. Adressvariablen. Eine solche Variable hat als Inhalt die Adresse einer anderen Variablen, ist also immer auf eine andere Variable bezogen.

4.1.2 Vereinbarung einer Adressvariablen

Eine Adressvariable, die die Adresse einer `int`-Variablen aufnehmen kann, erhält den Datentyp `int*`:

```
1 int x;    /* x kann eine ganze Zahl aufnehmen */
2 int* p;   /* p kann die Adresse einer int-Variablen aufnehmen */
```

Der Stern wird manchmal direkt hinter das `int` geschrieben (`int*␣p`), manchmal direkt vor den Variablennamen (`int␣*p`); das ergibt keinen Unterschied.

4.1.3 Zuweisung an eine Adressvariable

In eine solche Variable kann man Werte hineinschreiben, und man kann auch einen Wert aus ihr lesen:

- Meist schreibt man in die Adressvariable die Adresse einer anderen Variablen (dazu ist sie ja da):

```
1 int a;
2 int* p;
3 p = &a;
```

  Das Zeichen `&` ist der sogenannte Adressoperator[1]. Nun liegt in `p` die Adresse der Variablen `a`. Ab jetzt kann man über `p` auf die Variable `a` zugreifen.

- In einem Mikrocontroller-System kann es sinnvoll sein, in die Adressvariable eine feste Adresse zu schreiben:

```
1 int* p;
2 p = 64;
```

  Jetzt kann man über `p` auf einen Port oder auf ein Register zugreifen, das an der Adresse liegt.

- Es wurde festgelegt, dass an der Adresse 0 niemals eine Variable liegt.

```
1 int* p;
2 p = NULL; // bedeutet p=0;
```

  Deshalb zeigt der Wert `NULL` (eine symbolische Konstante mit dem Wert 0) dem System an, dass die Adressvariable einen ungültigen Inhalt hat.

[1] Der Programmierer braucht die Adresse von `a` nicht zu wissen, diese Arbeit nimmt ihm der Adressoperator ab.

4.1.4 Benutzung einer Adressvariablen mit dem Inhaltsoperator

Nur die Adresse einer Variablen zu haben, nutzt nicht viel. Stattdessen möchte man auf den *Inhalt* der Variablen zugreifen. Dazu dient ein spezieller Operator:

```
1  #include <stdio.h>
2  int main(void)
3  {
4      int x=3, y;
5      int* px;
6
7      px=&x;
8      y=*px;
9      printf("x=%i, y=%i\n", x, y);
10     return 0;
11 }
```

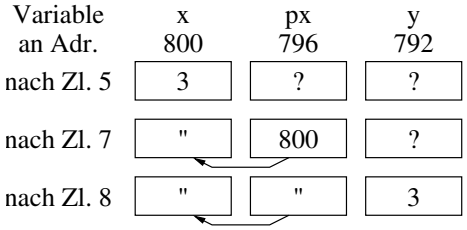

Abbildung 1: Benutzung einer Adressvariablen, Beispiel 1

- Zeile 7: Die Adresse von `x` wird nach `px` geschrieben (z. B. 800).
- Zeile 8: Mit dem Ausdruck `*px` bekommt man den Inhalt der Adresse 800, also den Inhalt von `x`. Die Zeile entspricht also der Anweisung `y=x;`.

  Daher wird hier der Wert 3 nach `y` geschrieben (Abbildung 1). Hier findet also über `*px` ein Lesezugriff auf die Variable `x` statt.

Merke: Mit dem Ausdruck `*px` wird auf den Inhalt der Variablen zugegriffen, deren Adresse in `px` steht.

Den Stern-Operator `*` nennt man darum *Inhalts-Operator*. Der Adress-Operator `&` und der Inhalts-Operator `*` heben einander auf: `*&x` ist gleich `&*x`, und das ist gleich `x`.

Der Stern als Inhalts-Operator ist **nicht** gleich dem Stern aus der *Vereinbarung* der Adressvariablen:

- Der Inhalts-Operator bewirkt einen Zugriff. Er ist Teil einer Anweisung und steht darum im Anweisungsteil der Funktion oder des Blocks.
- Der Stern in einer Variablen-Vereinbarung sagt aus, dass es sich um eine Adressvariable handelt. Er ist Teil einer Vereinbarung und steht deshalb im Vereinbarungsteil der Funktion oder des Blockes (oder in einer Parameterliste, siehe später).

Mit dem Ausdruck `*px` kann man die Variable, deren Adresse `px` enthält, nicht nur lesen, sondern auch schreiben:

```
1  #include <stdio.h>
2  int main(void)
3  {
4      int x=3, y=5;
```

```
5     int* px;
6
7     px=&x;
8     *px=y;
9     printf("x=%i, y=%i\n", x, y);
10    return 0;
11 }
```

- Zeile 7: Die Adresse von x wird nach px geschrieben (z. B. 800).
- Zeile 8: Mit dem Ausdruck *px greift man wieder auf den Inhalt der Adresse 800 zu, auf den Inhalt von x. Die Zeile entspricht damit der Anweisung x=y;. Daher wird der Wert 5 von y nach x geschrieben. Hier findet über *px ein Schreibzugriff auf die Variable x statt.

Ein Problem tritt im Zusammenhang mit dem Inhaltsoperator auf: Falls man vergessen hat, die Adresse von x nach px zuzuweisen, steht in px noch irgendeine unbekannte Adresse. Mit *px greift man dann auf diese zu, und das kann ungeahnte Folgen haben:

```
1  #include <stdio.h>
2  int main(void)
3  {
4     int x;
5     int* px;
6                 // vergessen zu schreiben: px=&x;
7     *px=1000; // schreibe die 1000 irgendwohin ...
8     return 0;
9  }
```

Bei einem so kleinen Programm wie diesem passiert höchstwahrscheinlich nur ein Speicherzugriffsfehler; bei größeren Programmen kann es durch einen solchen Fehler passieren, dass eine wichtige Variable (z. B. „aktuelle Flughöhe“) mit einem falschen Wert (z. B. „30 Meter“) überschrieben wird. Das ist wesentlich schlimmer, als wenn der Fehler sofort auffällt und das Programm sich beendet. Eine Lösung besteht darin, jede Adressvariable sofort mit einem geeigneten Wert zu initialisieren.

4.1.5 Weitere Beispiele für die Benutzung von Adressvariablen

Wenn px die Adresse von x enthält, dann kann man den Ausdruck *px überall dort verwenden, wo man auch x verwendet (Abbildung 2):

```
1  #include <stdio.h>
2  int main(void)
3  {
4     int x=3, y=4, z;
5     int* px;
6     int* py;
7
8     px = &x;
9     py = &y;
10    z = *px + *py; // z = x + y
11    *px = *py;     // x = y
12    *px = *px + 17;// x = x + 17
13    printf("x=%i, y=%i, z=%i\n", x, y, z);
14    return 0;
15 }
```

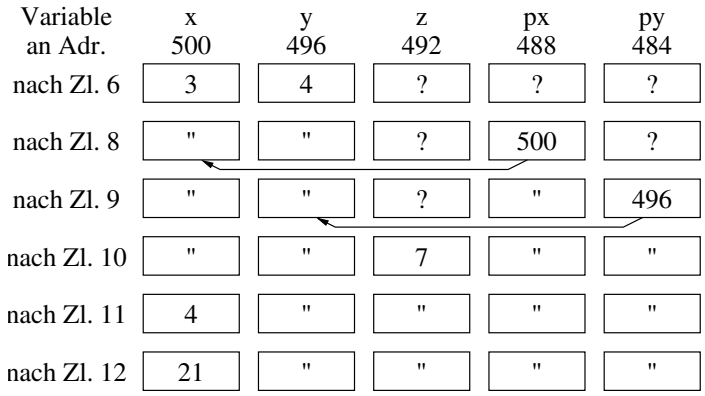

Abbildung 2: Benutzung einer Adressvariablen, Beispiel 2

Andererseits muss man die Anweisungen `*px=*py;` und `px=py;` unterscheiden. Im folgenden Beispiel werden x und y auf den Wert 4 gesetzt (Abbildung 3):

```
1  #include <stdio.h>
2  int main(void)
3  {
4      int x=3, y=4;
5      int* px;
6      int* py;
7
8      px = &x;
9      py = &y;
10     *px = *py; // x = y
11     printf("x=%i, y=%i\n", x, y);
12     return 0;
13 }
```

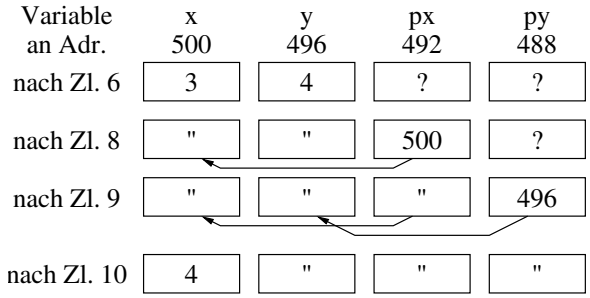

Abbildung 3: Benutzung einer Adressvariablen, Beispiel 3

Im nächsten Beispiel dagegen wird nur der Zeiger px umgesetzt, so dass beide Zeiger auf y zeigen Abbildung 4:

```
1  #include <stdio.h>
2  int main(void)
3  {
4      int x=3, y=4;
5      int* px;
6      int* py;
```

```
7
8       px = &x;
9       py = &y;
10      px = py; // Setze px=py, also auf Adresse von y
11      printf("x=%i, y=%i\n", x, y);
12      return 0;
13  }
```

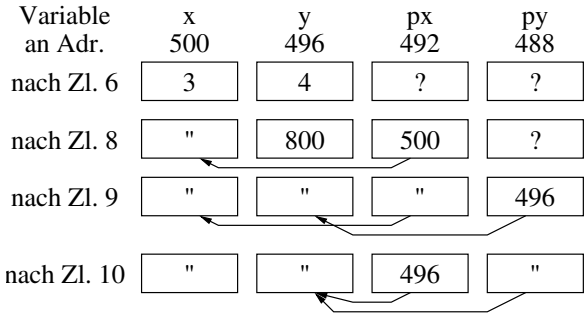

Abbildung 4: Benutzung einer Adressvariablen, Beispiel 4

4.1.6 Was man sonst noch über Adressvariablen wissen sollte

a) Die Adresse `NULL` (entspricht der Adresse 0) markiert eine besondere Adresse: Wenn man den Inhaltsoperator auf diese Adresse anwendet, erhält man einen Laufzeitfehler, das Programm bricht dann ab. Auf diese Art kann man mit dem Wert `0` die Zeiger versehen, die momentan keine gültige Adresse besitzen (also ungültig sind). Als Merksatz kann gelten: *Einem Nullzeiger darf man nicht folgen.*

Es bietet sich daher an, jede Adressvariable mit `NULL` zu initialisieren. Falls man vergisst, ihr eine Adresse zuzuweisen, bekommt man bei der ersten Benutzung des Ausdrucks `*px` einen Fehler – und kann ihn korrigieren. Das ist viel besser, als wenn `*px` auf *irgendeine* Adresse zugreift, deren Wert zufällig gerade in `px` steht.

b) Will man den Inhalt einer Adressvariable ausgeben, kann man das bei `printf()` mit der Formatanweisung `"%p"` tun. Die Ausgabe erfolgt dann meist so wie auf dem System für Adressen üblich (z.B. als Hexadezimalzahl).

c) Wenn man mehrere Adressvariablen in einer Zeile vereinbaren möchte, muss man für jede den Stern als Zeichen einer Adressvariablen angeben:

```
1   int* pa,  pb; // falsch: pa ist Adressvariable, pb ist int-Var.
2   int* pc,* pd; // richtig: pc und pd sind Adressvariablen
```