2.4 Funktionen/Sichtbarkeit und Lebensdauer

2.4.1 Variablen aus `main` in anderen Funktionen benutzen?

Im Programm `src/linie2.c` soll in der `main`-Funktion bestimmt werden, wie lang die Linie in der Funktion `linie` sein soll. Leider lässt sich das folgende Programm nicht compilieren:

```
1  #include <stdio.h>
2  void briefkopf(void)
3  {printf("Herbert␣Mustermann\nMusterweg␣13\n12345␣Osterhausen\n\n");}
4  void anrede(void)
5  {printf("Sehr␣geehrte␣Dame,␣sehr␣geehrter␣Herr!\n\n");}
6  void gruesse(void)
7  {printf("Mfg,\nHerbert␣Mustermann\n\f");}
8  /*****************************************************/
9  void linie(void)
10 {
11     int i;
12     for(i=0; i<lmax; ++i)
13         printf("-");
14     printf("\n");
15 }
16 /*****************************************************/
17 int main(void)
18 {
19     unsigned int lmax;
20     printf("Wie␣lang␣soll␣die␣Linie␣sein:␣"); scanf("%u", &lmax);
21     linie();
22     briefkopf();
23     printf("An\nSuperTeX␣GmbH\nPostfach\n98765␣Superhausen\n");
24     anrede();
25     printf("Hiermit␣bestelle␣ich␣das␣Abonnement␣der␣Zeitschrift\n"
26            "␣'Loeten␣fuer␣Anfaenger'␣zum␣naechstmoeglichen␣Zeitpunkt\n"
27            "ab.\n");
28     gruesse();
29     linie();
30     return 0;
31 }
```

Es gibt eine Fehlermeldung:

Terminal

```
schueler@debian964:~$ gcc src/linie2.c
src/linie2.c: In function 'linie':
src/linie2.c:12: error: 'lmax' undeclared (first use in this function)
...
```

Offenbar kennt die Funktion `linie()` die Variable `lmax` nicht – und dass, obwohl sie vorher und nachher in `main()` durchaus bekannt ist! Grund dafür ist eine Eigenschaft von Variablen, die absichtlich zum Schutz vor versehentlicher Veränderung eingebaut wurde, die sogenannte *Sichtbarkeit*. Zusammen mit einer anderen Eigenschaft, der *Lebensdauer*, bildet sie ein Grundkonzept in allen modernen Programmiersprachen.

2.4.2 Sichtbarkeit

Die Sichtbarkeit einer Variablen bedeutet: Man kann auf die Variable unter ihrem Namen zugreifen. Man kann sie also beschreiben, lesen, ihre Adresse und ihre Größe ermitteln.

Wann ist eine Variable in C sichtbar? Dazu gibt es zwei wichtige Regeln:

S1 Eine Variable ist in dem Block sichtbar, in dem sie vereinbart wurde.

S2 Die Ausnahme lautet: Wenn in einem inneren Block eine Variable vereinbart wird, die den gleichen Namen hat wie eine Variable, die bereits in einem äußeren Block vereinbart wurde, dann *verdeckt* die innen vereinbarte Variable im inneren Block die außen vereinbarte Variable.

Sichtbarkeit ist eine Eigenschaft, die sich auf den Quelltext bezieht und daher gut in Quelltextzeilen dargestellt werden kann.

Im folgenden Beispiel (`src/block0.c`) sind im inneren Block (Zeilen 8 bis 11) alle drei Variablen `i`, `j` und `k` sichtbar. Im äußersten Block (Zeilen 4 bis 14), der durch die `main()`-Funktion gebildet wird, ist dagegen nur `i` sichtbar:

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int i=3;
6      {
7          int j=4;
8          {
9              int k=5;
10             printf("i=%i , j=%i , k=%i\n", i,j,k);
11         }
12     }
13     return 0;
14 }
```

Nennt man dagegen alle drei Variablen `i`, ergibt sich ein anderes Bild (`src/block1.c`). Nun ist im innersten Block nur noch das innerste `i` sichtbar. Die beiden anderen Variablen, die auch `i` heißen, werden im inneren Block einfach verdeckt:

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int i=3;
6      {
7          int i=4;
8          {
9              int i=5;
10             printf("i=%i\n", i);
11         }
12     }
13     return 0;
14 }
```

Es ist übrigens eine gute Angewohnheit, Variablen so lokal wie möglich zu vereinbaren (also z.B. nur im inneren Block), so dass ihre versehentliche Verwendung in anderen Zusammenhängen so unwahrscheinlich wie möglich wird. Das absichtliche Verdecken von Variablen (etwa, indem man alle Variablen nichtssagend `i` nennt) sollte man dagegen eher vermeiden, weil man sonst leicht Verwirrung stiften kann.

2.4.3 Lebensdauer

Die Lebensdauer einer Variablen bedeutet: Der Speicherplatz der Variablen ist reserviert, und damit ist ihr Inhalt während der Lebensdauer gültig.

Nach Ende der Lebensdauer kann der Speicherplatz sofort oder später an eine andere Variable abgegeben werden. Selbst wenn man dann auf diesen Speicherplatz zugreifen kann, kann der Inhalt schon längst geändert worden sein.

Für die Lebensdauer gibt es nur eine Regel:

L1 Die Lebensdauer einer Variablen ist gleich der Laufzeit des Blockes, in dem sie vereinbart wurde.

Während die Sichtbarkeit von Variablen eine Frage des *Ortes* war, geht es bei der Lebensdauer von Variablen um eine Frage der *Zeit*. Daher betrachtet man bei der Lebensdauer nicht allein den Quelltext, sondern die Speicheradressen (y-Achse) im Verlauf der Zeit (x-Achse).

Im folgenden Beispiel (`src/funktion1.c`) gibt es eine Variable `i`, deren Lebensdauer offenbar nur in der Funktion liegt:

```c
#include <stdio.h>

void funk(void)
{
    int i;
    printf("Funktion beginnt...\n");
    printf("Alter Wert von i=%i\n", i);
    printf("Eingabe neuer Wert von i: "); scanf("%i", &i);
    printf("Funktion endet.\n\n");
}

int main(void)
{
    funk();
    funk();
    funk();
    return 0;
}
```

In der Funktion `funk()` wird der Wert von `i` zuerst auf den Bildschirm ausgegeben, anschließend wird ein neuer Wert von der Tastatur eingegeben. Damit kann man herausfinden, ob der Wert von `i` zwischen zwei Läufen konstant bleibt:

Terminal

```
schueler@debian964:~$ funktion1
Funktion beginnt...
Alter Wert von i=-1079171368
Eingabe neuer Wert von i: 3
Funktion endet.

Funktion beginnt...
Alter Wert von i=3
Eingabe neuer Wert von i: 5
Funktion endet.

Funktion beginnt...
Alter Wert von i=5
Eingabe neuer Wert von i: 7
Funktion endet.
```

Hier braucht man sich scheinbar um Lebensdauern keine Sorgen zu machen. Aber das böse Erwachen kommt wie so oft schon mit der nächsten Änderung wie in Beispiel `src/funktion2.c`:

```c
#include <stdio.h>

void funk(void)
{
    int i;
    printf("Funktion beginnt...\n");
    printf("Alter Wert von i: %i\n", i);
    printf("Eingabe neuer Wert von i: "); scanf("%i", &i);
    printf("Funktion endet.\n\n");
}

void andere(void)
{ int a=99,b=100,c=101,d=102; }

int main(void)
{
    funk();
    andere();
    funk();
    andere();
    funk();
    return 0;
}
```

Terminal

```
schueler@debian964:~$ funktion2
Funktion beginnt...
Alter Wert von i: -1081152616
Eingabe neuer Wert von i: 3
Funktion endet.

Funktion beginnt...
Alter Wert von i: 100
Eingabe neuer Wert von i: 5
Funktion endet.

Funktion beginnt...
Alter Wert von i: 100
Eingabe neuer Wert von i: 7
Funktion endet.
```

Hier wird anscheinend der Wert von `i` doch verändert; erstaunlicherweise bekommt er den Inhalt von `a` aus der Funktion `andere`. Die Regel von der Lebensdauer stimmt also doch: Funktionsende = Blockende = Ende der Lebensdauer von `i`. Aber warum hat es vorher geklappt und jetzt nicht mehr? Und wie kommt der Inhalt von `a` nach `i`?

Dazu muss man wissen, dass in C alle „normalen“ Variablen auf dem *Stack* angelegt werden. Das ist ein Stapelspeicher, auf dem jede Funktion, wenn sie aufgerufen wird, ihre lokalen Variablen (dazu ihre Parameter, Rücksprungadresse und weiteren Kleinkram) ablegen darf (ebenso jeder Block). Beim Verlassen der Funktion wird dieser Bereich wieder freigegeben, und wenn die letzte Funktion (z.B. `main()`) beendet wird, ist der Stapel wieder leer.

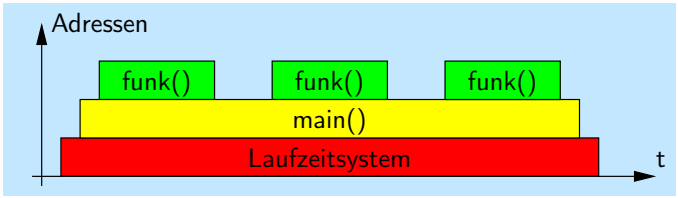

Abbildung 1: Stack bei `funktion1.c`

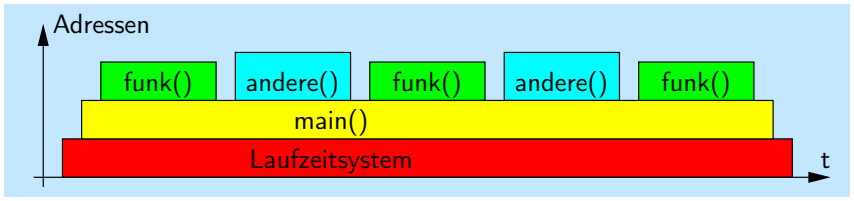

Abbildung 2: Stack bei `funktion2.c`

In Abbildung 1 sieht man diesen Ablauf für `src/funktion1.c`. Die Variable `i` lebt nur so lange wie der Stack der Funktion `funk()`. In Abbildung 2 ist das genauso. Nun aber wird zwischendurch der Speicherplatz, der in der Funktion `funk()` für die Variable `i` reserviert gewesen ist, in der Funktion `andere` für die Variable `a` verwendet. Man sieht also: Wenn man die Lebensdauer von Variablen nicht beachtet, kann das manchmal klappen, geht aber auf Dauer meistens schief.