4.11.F Datenstrukturen/Rückgabe von Arrays durch Funktionen – Ergänzungen und Bilder

4.11.F.1 Rückgabe eines Nullzeiger statt der Adresse einer lokalen Variablen

Wenn ein Programm die Adresse einer (nicht mit static gekennzeichneten) lokalen Variablen zurückgibt, tritt einer der folgenden Fälle ein:

a) Der Rückgabewert zeigt nur noch auf Unsinn, weil die Lebensdauer der Variablen vorbei ist

b) Der Rückgabewert enthält die Adresse NULL, ein Zugriff über diese Adresse (z. B. mit `printf`) erzeugt einen Speicherzugriffsfehler

Bei neueren GNU-Compilern wird ein Code erzeugt, der den zweiten Fall bewirkt.

Warum gibt es überhaupt diese beiden verschiedenen Fälle? Der Sprachstandard erlaubt es, dass bei einem Zeiger auf eine Variable, deren Lebensdauer zu Ende ist, dieser Zeiger selbst (!) auf einen undefinierten Wert gesetzt wird. Das heißt, nicht nur das Zeiger-Ziel kann verändert werden (durch folgende Zugriffe), sondern der Zeiger selbst darf verändert werden.

Wozu dieser Eingriff? Man will verhindern, dass durch den Zeiger ein versehentlicher Zugriff auf nicht mehr gültigen Speicherplatz passieren kann. Stattdessen kann der Zeiger so umgebogen werden, dass er auf eine vergleichsweise harmlose Adresse zugreift. Am besten ist das, wenn es die Adresse `NULL` ist. Ein solcher Zeiger gilt als ungültig und fällt deshalb auf.

Neuere GCC-Versionen setzen deshalb einen solchen Zeiger auf `NULL`, wenn sie ihn erkennen. Aber wie macht man so etwas? Kein übliches C-Laufzeitsystem prüft zur Laufzeit des Programmes immer wieder alle Zeiger ab, ob sie auch auf gültige Variablen zeigen. Mit diesem Vorgehen würde das Programm viel zu sehr verlangsamt. Deshalb sieht der GCC schon beim Compilieren nach, ob der Rückgabewert einer Funktion zugleich die Adresse einer lokalen Variablen ist. Wenn ja, wird (wie bisher) beim Compilieren eine Warnmeldung ausgegeben; zusätzlich wird das erzeugte Programm `a.out` so gestaltet, dass statt der Adresse der lokalen Variablen der Wert `NULL` zurückgegeben wird:

```
1 char *funk(void)
2 {
3     char v[80]="Blabla";
4     return v; // wird zu: return NULL;
5 }
```

Aber auch der GCC kann nicht zaubern. Sobald man die Adresse an eine andere Variable weitergibt, wird der Fehler nicht mehr erkannt:

```
1 char *funk(void)
2 {
3     char v[80]="Blabla";
4     char* p = v;
5     return p; // wird _nicht_ zu return NULL ...
6 }
```

4.11.F.2 Rückgabe eines mehrdimensionalen Arrays (Beispiel)

Auch bei mehrdimensionalen Arrays ist die Rückgabe per Zeiger möglich. Natürlich muss man auch hier auf die Lebensdauer des Arrays achten. Hier ist ein Beispiel:

```
1  #include <stdio.h>
2
3  char (*funk(void))[5]
4  {
5      static char x[3][5]={"Audi", "Fiat", ""};
6      return x;
7  }
8  /*****************************************/
9  int main(void)
10 {
11     char (*y)[5];
12     int lauf;
13
14     y=funk();
15     for(lauf=0; y[lauf][0]!='\0'; ++lauf)
16     {
17         printf("%s\n", y[lauf]);
18     }
19     return 0;
20 }
```

Zeile 5 Hier ist die lokale Variable. `x` ist ein Array mit drei Elementen vom Typ Array mit fünf Elementen vom Typ `char`. `x` enthält also drei Strings mit je maximal vier Zeichen plus Terminator.

Die ersten beiden Elemente enthalten je ein Wort, das dritte Element dient als Terminator-Element: Hinter einem Leerstring folgt nichts mehr.

Die Variable ist mit `static` gekennzeichnet; daher ist ihre Lebensdauer gleich der Programmdauer.

Zeile 6 Die Startadresse der lokalen Variablen `x` wird an das Hauptprogramm zurückgegeben.

Zeile 3 Der Funktionskopf sagt: `funk` it eine Funktion (Parameterliste: leer) mit Rückgabetyp Zeiger auf ein Array mit fünf Elementen vom Typ `char`.

`funk` gibt also das zurück, was die Variable `y` in Zeile 11 erwartet.

Zeile 11 Die Variable `y` ist ein Zeiger auf ein Array mit fünf Elementen vom Typ `char`.

Genauso wie bei der Zuweisung eines mehrdimensionalen Feldes an einen Zeiger ist auch hier nur die äußerste Dimension zu einem Zeiger geworden; die innere Dimension bleibt erhalten, damit man mit `++y` zum nächsten Element kommt.

Zeile 12 Hier ist die Zuweisung.

Zeile 15 Hier wird das Array durchlaufen. Und zwar Zeile für Zeile, nicht Zeichen für Zeichen. Mit `++y` (bzw. `++lauf`) kommt man zur nächsten Zeile.

Zeile 17 `y[lauf]` ist zwar ein Element des äußeren Arrays `y`, aber gleichzeitig die Startadresse des inneren Arrays.