4.5 Datenstrukturen/Mehrdimensionale Arrays

4.5.1 Anwendungsfall: Array von Strings

4.5.1.1 Problem Manchmal ist es notwendig, ein Array von Strings (Zeichenketten) zur Verfügung zu haben, aus dem man einen String nach Belieben auswählen kann. So könnte manches Programmstück vereinfacht werden, zum Beispiel dieses (aus dem Programm `src/notenausgabe1.c`):

```
1  switch(note)
2  {
3  case 1: printf("sehr gut");     break;
4  case 2: printf("gut");          break;
5  case 3: printf("befriedigend");break;
6  case 4: printf("ausreichend"); break;
7  case 5: printf("mangelhaft");  break;
8  case 6: printf("ungenuegend"); break;
9  default: printf("unbekannt");
10 }
```

Mit einem Array von Strings (Variablenname: `strarray`) soll das gleiche Programmstück ganz einfach aussehen:

```
1  printf(strarray[note]);
```

4.5.1.2 Deklaration Wie sieht nun ein solches Array aus? Ein String ist ja bereits ein Array:

```
1  char ch='B';           /* enthaelt genau ein alphanum. Zeichen */
2  char str[81]="Hallo"; /* enthaelt bis zu 80 alphanum. Zeichen */
```

Was man hier also braucht, ist ein Array von Array von Zeichen (Abbildung 1). In C wird das wie folgt deklariert:

```
1  char strarray[7][13];
```

Das bedeutet: `strarray` ist ein Array (mit 7 Elementen) von Arrays (mit je 13 Elementen) von `char`. Man beginnt das Lesen dieser Deklaration mit dem ersten unbekannten Wort: Das ist der Variablenname `strarray`. Dann geht es weiter mit dem Konstrukt, das mit dem Variablennamen am nächsten verbunden ist: Das ist die erste eckige Klammer samt Inhalt `[7]`; es bedeutet, dass `strarray` ein Array mit sieben Elementen ist. Weiter geht es mit dem Konstrukt, das mit dem bisherigen Vereinbarten `strarray[7]` am nächsten verbunden ist: Das ist die zweite eckige Klammer mit Inhalt `[13]`; es bedeutet, dass die Elemente des ersten Elementes Arrays mit 13 Elementen sind. Weiter geht es mit dem Rest, dem Datentyp `char`: Er bedeutet, dass die Elemente des zweiten Arrays alphanumerische Zeichen sind. Auf diese Art kann man auch kompliziertere Deklarationen analysieren.

4.5.1.3 Initialisierung Ein solches Array von Strings soll oft mit den richtigen Inhalten initialisiert werden:

```
1  char strarray[7][13]={"unbekannt", "sehr gut", "gut",
2                        "befriedigend", "ausreichend",
3                        "mangelhaft", "ungenuegend"};
```

Die geschweiften Klammern sind dabei notwendig.

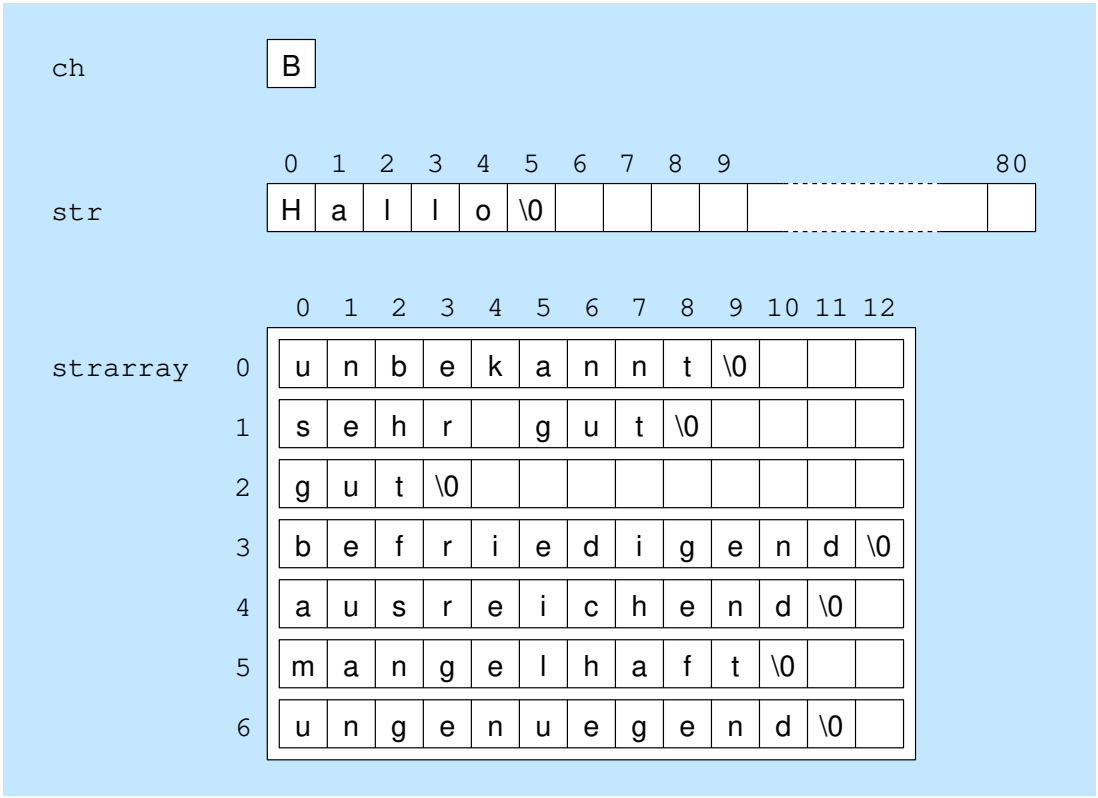


Abbildung 1: Von oben nach unten: Zeichen, String und Array von Strings

4.5.1.4 Zugriff Da nun ein Array von Arrays vorliegt, tauchen drei Ebenen auf:

a) `strarray` meint das ganze Array. Ausdrucken mit einem einzigen `printf()`-Befehl kann man es nicht.

b) `strarray[2]` meint die dritte Zeile, die zum Inhalt `"gut"` gehört. Man kann sie als ganzes ausdrucken und einlesen mit den Anweisungen:

```
1    scanf("%12[^\n]", strarray[2]); while(getchar()!='\n'){}
2    printf("%12s", strarray[2]);
```

Hier fällt bei der `scanf()`-Anweisung der Adressoperator `&` weg; das hat mit den besonderen Eigenschaften von Arrays in C zu tun.

c) `strarray[2][0]` meint das erste Zeichen in der dritten Zeile, es ist mit dem Buchstaben `'g'` initialisiert. Auch darauf kann man mit einem Befehl zugreifen:

```
1    scanf("%c", &strarray[2][0]); while(getchar()!='\n'){}
2    printf("%c", strarray[2][0]);
```

Oder einfacher mit `putchar()` und `getchar()`:

```
1    strarray[2][0] = getchar(); while(getchar()!='\n'){}
2    putchar(strarray[2][0]);
```

4.5.1.5 Ergebnis Das komplette Beispiel zeigt das Programm `src/notenausgabe2.c`:

```c
#include <stdio.h>
int main(void)
{
    unsigned int note;
    char strarray[7][13]={"unbekannt", "sehr gut", "gut",
                          "befriedigend", "ausreichend",
                          "mangelhaft", "ungenuegend"};
    printf("Wie lautet die Note als Zahl: ");
    scanf("%u", &note); while(getchar()!='\n'){}
    if(note > 6)
    {
        note = 0; /* bei Fehleingaben "unbekannt" ausgeben */
    }
    printf("Die Note als Wort lautet: %s\n", strarray[note]);
    return 0;
}
```

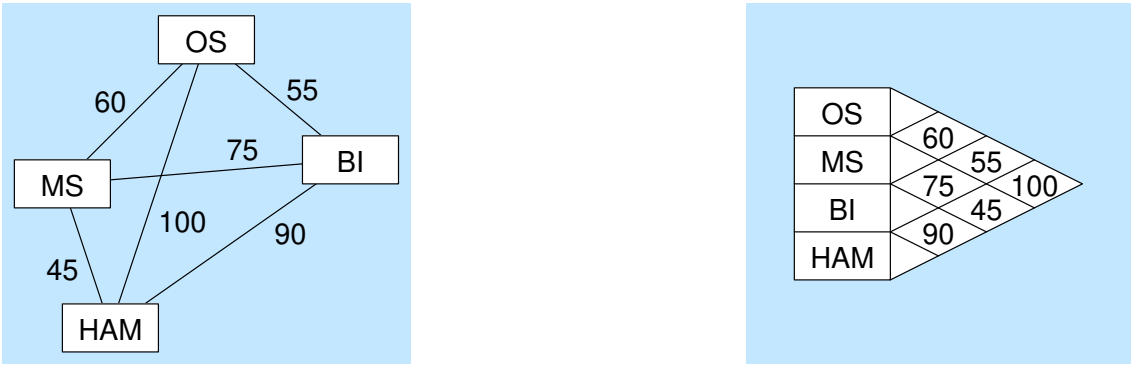
4.5.2 Anwendungsfall: Entfernungstabelle

4.5.2.1 Problem Die Entfernungen zwischen verschiedenen Städten sollen in einem Programm gespeichert und verarbeitet werden. Nach Eingabe eines Start- und eines Zielorts soll die Streckenlänge ermittelt und ausgegeben werden:

Terminal

```
schueler@debian964:~$ entfernung
Startort eingeben (0:OS, 1:MS, 2:BI, 3:HAM): 3
Zielort  eingeben (0:OS, 1:MS, 2:BI, 3:HAM): 1
Die Strecke hat eine Laenge von 45 km.
```

Bis jetzt liegen nur ein Graph und ein daraus entwickeltes Diagramm vor (Abbildung 2).



(a) Graph (b) Diagramm

Abbildung 2: Entfernungen zwischen Osnabrück, Münster, Bielefeld und Hamm

4.5.2.2 Analyse Das Diagramm in Abbildung 2 ist zwar dreieckförmig, weist aber – wenn man es mit etwas Phantasie schräg anschaut – Ähnlichkeit mit einer quadratischen Matrix auf: Jede Stadt hat gegenüber jeder anderen Stadt eine Entfernung, bei n Städten sind das n^2 Entfernungen. Allerdings ist die Entfernung von A nach B immer gleich der Entfernung von B nach A (wenn

keine Einbahnstraßen im Spiel sind). Damit ist diese Matrix *symmetrisch*. Tabelle 1 zeigt die Entfernungsmatrix, wie sie im Programm vorkommen soll.

	OS	MS	BI	HAM
OS	0	60	55	100
MS	60	0	75	45
BI	55	75	0	90
HAM	100	45	90	0

Tabelle 1: Entfernungen: Matrix

4.5.2.3 Deklaration Im Programm wird eigentlich ein zweidimensionales Array erwartet. Leider gibt es so etwas in C nicht. Aber es gibt — wie schon oben gesehen — die Möglichkeit, ein Array von Arrays von Zahlen zu deklarieren:

```
1   double entfernungen[4][4];
```

Die Variable `entfernungen` bezeichnet also ein Array (mit 4 Elementen) von Arrays (mit 4 Elementen) von Gleitkommazahlen des Typs `double`.

4.5.2.4 Initialisierung Auch dieses Array kann wieder initialisiert werden (die Kennzeichnung der `double`-Konstanten durch Dezimalpunkt ist hier der Übersichtlichkeit wegen vorsätzlich weggelassen worden):

```
1   double entfernungen[4][4]={{  0,60,55,100},
2                              { 60, 0,75, 45},
3                              { 55,75, 0, 90},
4                              {100,45,90,  0}};
```

Die inneren geschweiften Klammern können dabei weggelassen werden, was aber nicht empfohlen wird, da dann kleine Flüchtigkeitsfehler eventuell nicht mehr erkannt werden.

4.5.2.5 Zugriff Wieder tauchen drei Ebenen auf:

a) `entfernungen` meint das ganze Array.

b) `entfernungen[3]` meint die unterste Zeile, die zu allen Entfernungen von Hamm aus gehört.

c) `entfernungen[3][1]` meint die Entfernung von Hamm nach Münster und enthält den Wert `45.0`.

4.5.2.6 Lösung Die komplette Lösung zeigt das Programm `src/entfernung.c`:

```
1  #include <stdio.h>
2  #define AS          4 /* Anzahl der Staedte */
3  int main(void)
4  {
5     unsigned int so, zo;
6     double strecke;
7     double entfernungen[AS][AS]={{  0, 60, 55, 100},
8                                  { 60,  0, 75,  45},
9                                  { 55, 75,  0,  90},
10                                 {100, 45, 90,   0}};
11    printf("Startort␣eingeben␣(0:OS,␣1:MS,␣2:BI,␣3:HAM):␣");
```

```
12      scanf("%u", &so);
13      printf("Zielort  eingeben (0:OS, 1:MS, 2:BI, 3:HAM): ");
14      scanf("%u", &zo);
15      if(zo>=AS || so>=AS) return 1;
16      strecke=entfernungen[zo][so];
17      printf("Die Strecke hat eine Laenge von %.0f km.\n", strecke);
18      return 0;
19  }
```