Kapitel 10

Strukturierte Datentypen

Über die normalen Datentypen hinaus gibt es in C weitere, komplexere Typen, die sich aus den einfacheren zusammensetzen. Außerdem besteht die Möglichkeit, eigene Synonyme für häufig verwendete Typen festzulegen.

`Feld (array)`	Zusammenfassung von Elementen gleichen Typs
`Struktur (struct)`	Zusammenfassung von Elementen verschiedenen Typs
`Union (union)`	Überlagerung mehrerer Komponenten verschiedenen Typs auf dem gleichen Speicherplatz
`Aufzählungstyp (enum)`	Grunddatentyp mit frei wählbarem Wertebereich

Der Typ Feld wurde bereits in Kapitel 6.3 vorgestellt.

10.1 Strukturen

Die Struktur definiert einen neuen Datentyp, welcher Komponenten unterschiedlichen Typs vereint. Die Länge einer solchen Struktur ist gleich der Gesamtlänge der einzelnen Bestandteile. Angewendet werden Strukturen häufig dann, wenn verschiedene Variablen logisch zusammengehören, wie zum Beispiel Name, Vorname, Straße etc. bei der Bearbeitung von Adressen. Die Verwendung von Strukturen ist nicht zwingend nötig, man kann sie auch durch die Benutzung von mehreren einzelnen Variablen ersetzen. Sie bieten bei der Programmierung jedoch einige Vorteile, da sie die Übersichtlichkeit erhöhen und die Bearbeitung vereinfachen.

10.1.1 Deklaration von Strukturen

Zur Deklaration einer Struktur benutzt man das Schlüsselwort `struct`. Ihm folgt der Name der Struktur. In geschweiften Klammern werden dann die einzelnen Variablen aufgeführt, aus denen die Struktur bestehen soll.
Achtung: Die Variablen innerhalb einer Stuktur können nicht initialisiert werden:

```
struct Strukturname
{
  Datendeklaration
};
```

Im folgenden Beispiel wird eine Stuktur mit Namen Student deklariert, die aus einer `int`–Variablen und einem String besteht.

Beispiel 10.1 Deklaration einer Struktur.

```
struct Student
       {
              int matrikel;
              char name[16];
       };
```

□

10.1.2 Definition von Strukturvariablen

Die Strukturschablone selbst hat noch keinen Speicherplatz belegt, sie hat lediglich ein Muster festgelegt, mit dem die eigentlichen Variablen definiert werden. Die Definition erfolgt genau wie bei den Standardvariablen (int a, double summe etc.), indem man den Variablentyp (struct Student) gefolgt von einem oder mehreren Variablennamen angibt, zum Beispiel:

```
struct Student peter, paul;
```

Hierdurch werden zwei Variablen peter, paul deklariert, die beide vom Typ struct Student sind. Neben dieser beschriebenen Methode gibt es noch eine weitere Form der Definition von Strukturvariablen:

Beispiel 10.2 Deklaration einer Struktur und Definition der Strukturvariablen.

```
struct Student
       {
              int matrikel;
              char name [16];
       } peter, paul;
```

In diesem Fall erfolgt die Definition der Variablen zusammen mit der Deklaration. Dazu müssen nur eine oder mehrere Variablen direkt hinter die Deklaration gesetzt werden. □

10.1.3 Felder von Strukturen

Eine Struktur wird häufig nicht nur zur Aufnahme eines einzelnen Datensatzes, sondern zur Speicherung vieler gleichartiger Sätze verwendet, beispielsweise mit

```
struct Student Studenten2006[1000];
```

Der Zugriff auf einzelne Feldelemente erfolgt wie gewohnt mit eckigen Klammern [].

10.1.4 Zugriff auf Strukturen

Um Strukturen sinnvoll nutzen zu können, muss man ihren Elementen Werte zuweisen und auf diese Werte auch wieder zugreifen können. Zu diesem Zweck kennt C den Strukturoperator '.', mit dem jeder Bestandteil einer Struktur direkt angesprochen werden kann.

Beispiel 10.3 Deklaration, Definition und Zuweisung.

```
#include <stdio.h>
#include <string.h> /* Fuer strcpy */

int main()
{
```

```
        /* Deklaration der Struktur Student */
        struct Student
        {
                int matrikel;
                char Vorname[16];
        };

        /* Definition eines Feldes der Struktur Student */
        struct Student Mathe[100];

        /* Zugriff auf die einzelnen Elemente */

        Mathe[0].matrikel=242834;
        strcpy(Mathe[0].Vorname,"Peter");

        Mathe[1].matrikel=343334;
        strcpy(Mathe[1].Vorname,"Paul");

        /* Der Zuweisungsoperator = ist auf Strukturen
           gleichen Typs anwendbar */

        Mathe[2]=Mathe[1];
        printf("Vorname von Student 2: %s\n",Mathe[2].Vorname);

        return 0;
}
```

□

10.1.5 Zugriff auf Strukturen mit Zeigern

Wurde eine Struktur deklariert, so kann man auch einen Zeiger auf diesen Typ wie gewohnt definieren, wie zum Beispiel

```
struct Student *p;
```

Der Zugriff auf den Inhalt der Adresse, auf die der Zeiger verweist, erfolgt wie üblich durch den Zugriffsoperator `*`. Wahlweise kann auch der Auswahloperator `->` benutzt werden.

Beispiel 10.4 Zugriff mit Zeigern.

```
#include <stdio.h>
#include <string.h> /* Fuer strcpy */

int main()
{
        /* Deklaration der Struktur Student */
        struct Student
        {
                int matrikel;
                char Strasse[16];
        };

        /* Definition Strukturvariablen peter, paul */
        struct Student peter, paul;
        struct Student *p;    /* Zeiger auf Struktur Student */

        p=&peter; /* p zeigt auf peter */
```

```
        /* Zugriff auf die einzelnen Elemente */

        (*p).matrikel=242834;
        strcpy((*p).Strasse,"Finkenweg 4");

        /* Alternativ */

        p=&paul;
        p->matrikel=423323;
        strcpy(p->Strasse,"Dorfgosse 2");

        printf("Paul, Matr.-Nr.: %i , Str.: %s\n "
        ,paul.matrikel,paul.Strasse);
        return 0;
}
```

□

10.1.6 Geschachtelte Strukturen

Bei der Deklaration von Strukturen kann innerhalb einer Struktur eine weitere Struktur eingebettet werden. Dabei kann es sich um eine bereits an einer anderen Stelle deklarierte Struktur handeln, oder man verwendet an der entsprechenden Stelle nochmals das Schlüsselwort struct und deklariert eine Struktur innerhalb der anderen.

Beispiel 10.5 Geschachtelte Strukturen.

```
#include <stdio.h>

int main()
{
        struct Punkt3D
        {
                float x;
                float y;
                float z;
        };

        struct Strecke3D
        {
                struct Punkt3D anfangspunkt;
                struct Punkt3D endpunkt;
        };

        struct Strecke3D s;

        /* Initialisierung einer Strecke */

        s.anfangspunkt.x=1.0;
        s.anfangspunkt.y=-1.0;
        s.anfangspunkt.z=2.0;
        s.endpunkt.x=1.1;
        s.endpunkt.y=1.2;
        s.endpunkt.z=1.4;
        printf("%f \n",s.endpunkt.z);
        return 0;
}
```

□

10.1.7 Listen

Wie in Abschnitt 10.1.6 bereits gezeigt wurde, können Strukturen auch andere (zuvor deklarierte) Strukturen als Komponenten enthalten. *Eine Struktur darf sich aber nicht selbst als Variable enthalten!*. Allerdings darf eine Struktur einen Zeiger auf sich selbst als Komponente beinhalten. Diese Datenstrukturen kommen zum Einsatz, wenn man nicht im voraus wissen kann, wieviel Speicher man für eine Liste von Datensätzen reservieren muss und daher die Verwendung von Feldern unzweckmäßig ist.

Beispiel 10.6 Liste.

```
#include <stdio.h>
#include <string.h>

int main()
{
        struct Student
        {
                char name[16];
                char familienstand[16];
                char geschlecht[16];
                struct Student *next;
        };

        struct Student Mathe[2];
        struct Student Bio[1];
        struct Student *startzeiger,*eintrag;

        /* Initialisierung der Mathe-Liste */
        strcpy(Mathe[0].name,"Kerstin");
        strcpy(Mathe[0].familienstand,"ledig\t");
        strcpy(Mathe[0].geschlecht,"weiblich");
        Mathe[0].next=&Mathe[1]; /* next zeigt auf den n\"achsten Eintag */

        strcpy(Mathe[1].name,"Claudia");
        strcpy(Mathe[1].familienstand,"verheiratet");
        strcpy(Mathe[1].geschlecht,"weiblich");
        Mathe[1].next=NULL; /* next zeigt auf NULL, d.h Listenende */

        /* Initialisierung der Bio-Liste */
        strcpy(Bio[0].name,"Peter");
        strcpy(Bio[0].familienstand,"geschieden");
        strcpy(Bio[0].geschlecht,"maennlich");
        Bio[0].next=NULL; /* next zeigt auf NULL, d.h Listenende */

        /* Ausgabe der Mathe-Liste */
        startzeiger=&Mathe[0];

        printf("Name\tFamilienstand\tGeschlecht\n\n");
        for (eintrag=startzeiger;eintrag!=NULL;eintrag=eintrag->next)
        {
                printf("%s\t%s\t%s\n"
                ,eintrag->name,eintrag->familienstand,eintrag->geschlecht);
        }
```

```
        /* Anh\"angen der Bio-Liste an die Mathe-Liste */
        Mathe[1].next=&Bio[0];

        /* Ausgabe der Mathe-Bio-Liste */
        printf("\n\nName\tFamilienstand\tGeschlecht\n\n");
        for (eintrag=startzeiger;eintrag!=NULL;eintrag=eintrag->next)
        {
                printf("%s\t%s\t%s\n"
                ,eintrag->name,eintrag->familienstand,eintrag->geschlecht);
        }
        return 0;
}
```

□

10.2 Unions

Während die Struktur sich dadurch auszeichnet, dass sie sich aus mehreren verschiedenen Datentypen zusammensetzt, ist das Charakteristische an Unions, dass sie zu verschiedenen Zeitpunkten jeweils einen bestimmten Datentyp aufnehmen können.

Beispiel 10.7 Union. Um die Klausurergebnisse von Studierenden zu speichern, müsste normalerweise zwischen Zahlen (Notenwerte im Falle des Bestehens) und Zeichenketten („nicht bestanden“) unterschieden werden. Verwendet man eine Unionvariable so kann man die jeweilige Situation flexibel handhaben

```
#include <stdio.h>
#include <string.h>

int main()
{
        union klausurresultat
                {
                        float note;
                        char nichtbestanden[16];
                };

        union klausurresultat ergebnis, *ergebnispointer;

        /* Zugriff mit pointer */
        ergebnispointer=&ergebnis;
        ergebnispointer->note=1.7;
        printf("Note %.1f :",ergebnispointer->note);
        strcpy(ergebnispointer->nichtbestanden,"bestanden");
        printf(" %s\n",ergebnispointer->nichtbestanden);

        /* Zugriff ohne pointer */
        ergebnis.note=5.0;
        printf("Note %.1f :",ergebnispointer->note);
        strcpy(ergebnis.nichtbestanden,"nicht bestanden");
        printf(" %s\n",ergebnispointer->nichtbestanden);

        return 0;
}
```

□

Der Speicherplatzbedarf einer Union richtet sich nach der größten Komponente (im Beispiel 10.7: 16 `sizeof(char)` = 16 Byte).

Gründe für das Benutzen von Unions fallen nicht so stark ins Auge wie bei Strukturen. Im wesentlichen sind es zwei Anwendungsfälle, in denen man sie einsetzt:

- Man möchte auf einen Speicherbereich auf unterschiedliche Weise zugreifen. Dies könnte bei der obigen Union doppelt der Fall sein. Hier kann man mit Hilfe der Komponente `nichtbestanden` auf die einzelnen Bytes der Komponente `note` zugreifen.
- Man benutzt in einer Struktur einen Bereich für verschiedene Aufgaben. Sollen beispielsweise in einer Struktur Mitarbeiterdaten gespeichert werde, so kann es sein, dass für den einen Angaben zum Stundenlohn in der Struktur vorhanden sein sollen, während der andere Speicherplatz für ein Monatsgehalt und der Dritte noch zusätzlich Angaben über Provisionen benötigt. Damit man nun nicht alle Varianten in die Struktur einbauen muss, wobei jeweils zwei unbenutzt blieben, definiert man einen Speicherbereich, in dem die jeweils benötigten Informationen abgelegt werden, als Union.

10.3 Aufzählungstyp

Der Aufzählungstyp ist ein Grundtyp mit frei wählbarem Wertebereich. Veranschaulicht wird dies am Beipiel der Wochtage.

Beispiel 10.8 Aufzählungstyp.

```
#include <stdio.h>
#include <string.h>

int main()
{
       enum tag
              {
                     montag, dienstag, mittwoch, donnerstag,
                     freitag, samstag, sonntag
              };

       enum tag wochentag;
       wochentag=montag;
       if (wochentag==montag) printf("Endlich wieder Vorlesungen!\n");
       return 0;
}
```

□

10.4 Allgemeine Typendefinition

Das Schlüsselwort `typedef` definiert neue Namen für bestehende Datentypen. Es erlaubt eine kürzere Schreibweise bei aufwendigen Deklarationen und kann Datentypen auf Wunsch aussagekräftigere Namen geben. Die Syntax für die Definition eines neuen Datentypnamens sieht wie folgt aus:

```
typedef typ Variablenname;
```

Beispiel 10.9 Typendefinition.

```
#include <stdio.h>
#include <string.h>
```

```
int main()
{
        struct vektor
                {
                        float x;
                        float y;
                        float z;
                };

        typedef char text[100];
        typedef struct vektor punkt;

        /* Deklaration der Variablen p und nachricht */
        punkt p;
        text nachricht;

        /* Initialisierung der Variablen p und nachricht */

        p.x=1.0;p.y=0.0;p.z=0.0;
        strcpy(nachricht,"nachricht ist eine Variable vom Typ text");
        printf("%s\n",nachricht);
        return 0;
}
```

Interessanterweise ist eine Variable vom Typ `text` nunmehr stets eine Zeichenkette der (max.) Länge 100. Weiterhin zeigt das Beispiel, dass auch strukturierten Datentypen eigene Typdefinitionen zugewiesen werden können. □