Grundlagen und Konzepte von C
Datenstrukturen

Ausarbeitung von Andreas Gadelmaier

Proseminar C Grundlagen und Konzepte
Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

1.Generelles:

Warum brauchen wir Datenstrukturen?
Datenstrukturen sind essenziell zum Programmieren, sie machen unseren Programmcode effizient.
In ihnen werden Daten abgespeichert. Elementare Datenstrukturen stellen somit die Grundlage für Algorithmen dar und sind unverzichtbar.

2.Struct:

Ein Struct ist eine Sammelstruktur, welche mehrere primitive, als auch komplexe Variablen zu einer Einheit zusammen fast. Variablen in dieser Struktur werden als Komponenten bezeichnet. Diese Komponenten können von unterschiedlichen Datentypen sein.

Eine Deklaration einer Solchen Struktur könnte wie folgt aussehen. Man beginnt mit dem Schlüsselwort „struct“ und weist diesem im Anschluss einen Namen zu, gefolgt von geschweiften Klammern. Der Inhalt der Klammer definiert hierbei den Inhalt unserer Struktur. Abgeschlossen wird das ganze wie üblich mit einem Semikolon.

```
struct BankKunde{
   char vorname[20];
   char nachname[20];
   int kontonummer;
   float kontostand;
}Kunde;
```

Um nun Variablen von diesem Typ zu erzeugen, gibt es zwei Möglichkeiten, erstere ist im Beispiel aufgeführt. Die andere Möglichkeit besteht darin, den Strukturtyp zunächst zu deklarieren, die Variablen aber erst nachher zu definieren. Definition mehrerer Variablen ist natürlich durch setzten von Kommata möglich.
Eine Zuweisungen kann entweder in geschweiften Klammern erfolgen oder Komponentenweise, hierbei muss der Bezeichner der Struktur durch einen Punkt getrennt mit angegeben werden.

```
struct BankKunde Kunde

Kunde.vorname = "Max";
Kunde.nachname= "Mustermann";
Kunde.kontonummer = 1234567;
Kunde.kontostand = 98765.32;
```

Die Größe einer solchen Struktur kann mit sizeof ermittelt werden, aber Achtung, die Größe der Struktur kann von der Größe der einzelnen aufaddierten Komponenten abweichen, eine Folge der Speicherbelegung für schnellen zugriff auf die einzelnen Variablen.

3.Union:

Ein Union ist einem Struct sehr ähnlich. Der Unterschied besteht im wesentlichen darin, dass Komponente im Union sich den selben Speicherbereich teilen. Daraus folgt das eine Variable vom Typ Union so viel Speicherbedarf hat wie die größte Komponente.

```
union Kundendaten {
int kundennummer;
short filiale;
}Daten;
```

Aus dem geteiltem Speicherbereich, resultiert die Gefahr von verfälschten Ergebnissen. Wenn die Zuweisung der Komponenten filiale stattfindet wird kundennummer überschrieben. Fehler die beim aufrufen der überschriebenen Komponente entstehen sind nur schwer zu finden, da sie nicht vom Compiler erkannt werden.

4.Array:

Ein Array ist wie ein Struct eine Sammelstruktur. Im wesentlichen besteht der Unterschied zum Struct in den Komponenten die gesammelt werden, denn im Array können nur Daten vom gleichen Typ gespeichert werden.
Die Deklaration eines Arrays sieht aus wie folgt. Als erstes legt man den Datentyp fest den man sammeln möchte. Nun gibt man dem Array einen Namen, gefolgt von eckigen Klammern. In den eckigen Klammern wird die Größe des Arrays festgelegt, dies kann auch dynamisch passieren.

```
float monatlicherKontostand[12]= {2350.00,
2167.20,
2800.70,
5260.90,
5720.50,
....};
```

Hierbei ist zu beachten das dem Array entweder in geschweiften Klammern oder indexbasierend Daten zugewiesen werden können. Um indexbasiert auf das Array zu zu greifen, nutzt man den Variablennamen des Arrays und in den eckigen Klammern schreibt man den Index, auf welchen man zugreifen möchte. Zu beachten ist dabei, dass hierbei mit Null gestartet wird. Hat das Array eine Größe von 12, so beschränkt sich der Zugriff auf 0-11 und Deckt somit alle Komponenten des Arrays ab.

```
monatlicherKontostand[0]= 2350.00;
monatlicherKontostand[1]= 2167.20;
monatlicherKontostand[2]= 2800.70;
monatlicherKontostand[3]= 5260.90;
monatlicherKontostand[4]= 5720.50;
```

Mehrdimensionale Arrays funktionieren im wesentlichen wie normale Arrays. Man erstellt ein solches Array, indem man statt einer Eckigen klammer mehrere Eckige klammern hintereinander schreibt, zu beachten ist dabei das eine Klammer geöffnet und wieder geschlossen wird, bevor eine neue geöffnet wird. Somit erhält man ein Array in einem Array. Dies bedeutet wenn man in die erste Klammer bei der Deklaration eine 12 angibt und in der zweiten eine zwei. So erhält man ein Array, welches 12 Arrays der Größe zwei enthält.

5.String:

Ein String ist eine Zeichenkette. Zeichenkette sind nichts anderes als aneinander gereihte Buchstaben, daraus folgt das ein String im wesentlichen nichts anderes ist als ein Array von Chars und genau so verhalten sie sich auch.
Wenn man ein Char Array erstellt, gibt es verschiedene Möglichkeiten ihnen einen Wert zu zu weisen. Man geht hierbei wie beim Array beschrieben vor, nur das der Datentyp Char sein muss.
Hat man sich nun für einen Namen und die Größe entschieden, kann man jeden Buchstaben einzeln zuweisen. Ist dies der fall, muss man die Eingabe mit terminierendem null Byte beenden, um das Ende zu signalisieren.

```
char name[20]='m','a','x','\0';
```

Die andere Möglichkeit besteht darin das ganze Wort in Anführungszeichen zu setzten.

```
char name[20]="max";
```

Da man die Größe eines Wortes relativ einfach abzählen kann, es bei einem Satz jedoch mühselig wird, erstellt man diese Arrays oft dynamisch.
Auch von Chars ist es möglich mehrdimensionale Arrays zu erstellen, welche so aussehen könnten:

```
char Kundenliste [3][20] = {
"Mueller"
,"Krause"
,"Schaefer"
};
```

6.Enum:

Ein Enum ist ein Aufzählungsdatentyp, welcher uns das Lesen von Quelltext erleichtert. So genannte Magic Numbers werden ersetzt. Dies sind Variablen, von welchen man nicht genau weiß wofür sie stehen, beziehungsweise es ist nicht sofort ersichtlich.
Im wesentlichen werden viele Konstanten Definiert. Welche sich dann ähnlich wie Integer verhalten.
Enums sind wie folgt aufgebaut. Man startet mit dem Schlüsselwort enum. Dahinter kommen geschweifte Klammer in welche man die Bezeichner einsetzt, gefolgt von dem Namen, welchem man dem Enum geben möchte . Beendet wird das ganze mit einem Semikolon hinter der schließenden Klammer.

Beispiel:

```
float druckeZinsen (int x){
if(x < 6) printf("Zinssatz␣bei␣1,5%");
else(x >= 6) printf("Zinssatz␣bei␣2,7%")
};
```

Aus dem Beispiel ist nicht sofort ersichtlich wovon der Zinssatz abhängig ist, man sieht nur das die Variable x Einfluss auf den Zinssatz hat, aber man kann nicht sehen wofür x steht.
Führen wir nun eine Enum ein und ersetzen x durch dieses Enum, wird das ganze schon recht übersichtlich.

```
typedef enum{
januar, februar, maerz,
april, mai, juni,
juli, august, september,
oktober, november, dezember
} monate;
float druckeZinsen (monat m){
if(m < 6) printf("Zinssatz␣bei␣1,5%");
else(m => 6) printf("Zinssatz␣bei␣2,7%")
};
```

Nun kann man eindeutig erkennen das der Zinssatz von der Jahreszeit abhängt.

Im Enum hat der Januar den Wert null und alle darauf folgenden Monate sind aufsteigend nummeriert, falls ihnen nicht explizit ein anderer Wert zugewiesen wird. Falls doch ein Wert zugewiesen wird, hat der folgende Bezeichner den gleichen Wert um eins erhöht.
Es spielt beim Einsetzen als Funktionsparameter keine Rolle, ob man nun Januar einsetzt oder ob man den Wert null nimmt.

7.Typedef:

Typedef ist eine Datenvereinbarung, in der man bestehenden Strukturen einen Namen gibt, somit besteht kein Unterschied zur Variablendeklaration. Bei dem Beispiel mit dem Enum ist ein solches Typedef verwendet worden. Sollte nun jemand monate benutzen wollen, muss nicht jedes mal deklariert werden das dieses ein Enum ist.
Ein anderes Beispiel könnte wie Folgt aussehen.

Ohne Typedef:

```
int main ( ){
struct BankKunde max;
return 0;
}
```

Mit Typdef:

```
typedef struct BankKunde{
   char vorname[20];
   char nachname[20];
   int kontonummer;
   float kontostand;
}Kunde;


int main ( ){
BankKunde max;
return 0;
}
```

Dies mag auf dem ersten Blick nach keiner großen Sache aussehen, wenn man aber mehr als nur eine Struktur hat, erleichtert es das Programmieren ungemein, da man nicht bei jeder Variablen explizit angeben muss zu was für einem Datentyp sie angehört.