Programmieren I

Programmieren in C

WS 2024

1 Einführung

C ist eine imperative Programmiersprache, die bereits seit 1972 existiert und somit eine der ältesten Programmiersprachen ist. Aufgrund der Eigenschaft, dass C eine Low-Level Sprache ist, wird sie oft für Betriebssysteme, Treiber oder embedded Systeme wie Arduinos, RasPis oder Autos verwendet.

Anders als bei vielen moderneren Programmiersprachen, gibt es in C relativ wenige Beschränkungen. Prinzipiell macht C genau das, was man ihm sagt. Das bedeutet aber auch, dass man sich selbst um viele Dinge kümmern muss und es in vielen Fällen zu ungewollten Nebeneffekten kommen kann, wenn man nicht aufpasst.

Beispiel 1.1. Es existiert ein Array (der Datentyp ist dabei egal) mit 10 Elementen. Um auf ein Element dieses Arrays zuzugreifen, wird die Syntax `array[i]` verwendet, wobei `i` der Index (also die Stelle) des Elements ist. Das heißt, der Aufruf `array[0]` gibt das erste Element des Arrays zurück, `array[1]` das zweite, usw.

Ruft man allerdings das Element `array[10]` auf, würde man einen Fehler erwarten, da das Array nur 10 Elemente beinhaltet. In vielen Programmiersprachen wäre das auch tatsächlich der Fall; Java würde eine `ArrayIndexOutOfBoundsException` werfen, Python einen `IndexError`, usw. C hingegen verfolgt das Mindset, dass der Entwickler schon weiß, was er tut und gibt somit einfach den Wert zurück, der sich an der 11. Stelle des Arrays befinden würde. Das klingt vielleicht etwas verwirrend, da es gar keinen 11. Wert gibt, aber ein Array ist im Endeffekt nichts anderes als eine durchgängige Reihe an Speicherzellen im Arbeitsspeicher. Über den Index berechnet man die Speicheradresse der jeweiligen Zelle und wenn der Index größer als die Länge des Arrays ist, wird auf eine Speicherzelle zugegriffen, die nicht zum Array gehört. Daher kann man also auch nicht vorhersehen, welcher Wert in dieser Zelle steht. Das ist zumden natürlich auch ein Sicherheitsrisiko, da unbefugt auf fremden Speicher zugegriffen wird und dieser im Wort-Case sensible Daten wie Passwörter beinhalten könnte.

2 Hello, World!

Oft ist das erste Programm, das man in einer (neuen) Programmiersprache schreibt, ein "Hello, World"-Programm. Das ist ein simples Programm, das den Text "Hello, World!" auf der Konsole ausgibt. Der Sinn dahinter ist, zu überprüfen, ob bei der Installation der Entwicklungsumgebung, Runtime, usw. alles korrekt verlief oder ob es zu Fehlern kam.

Möchte man in C sieht ein solches Programm schreiben, könnte das wie folgt aussehen:

```c
#include <stdio.h>

int main() {
  printf("Hello, World!\n");
  return 0;
}
```

Fig. 1: Hello, World! in C

Anhand dieses Beispiels lassen sich schon einige Aspekte der generellen C-Syntax erkennen. Zum Beispiel, dass (vereinfacht ausgedrückt) fast jede Zeile mit einem Semikolon (`;`) oder einer geschweiften Klammer ({ }) endet.

3 Datentypen, Variablen und Operatoren

Der Code *Fig. 1* zeigt bereits die grundlegende Struktur eines C-Programms. Der Startpunkt ist die `main`-Funktion, welche somit beim Start des Programms aufgerufen wird. Der genaue Aufbau einer Funktion wird in Kapitel 5 genauer erläutert.

3.1 Datentypen

Ein grundlegender Bestandteil von Programmiersprachen sind Datentypen. Mithilfe von Datentypen unterscheidet man, welchen Wert eine Variable speichern kann. Eine Art von Datentypen sind die **primitiven Datentypen**. Diese bilden die Basis für alle anderen Datentypen. Folgende primitive Datentypen gibt es in C:

Datentyp	Keyword	Speichergröße	Wert
Integer	int	16 bit / 32 bit	Ganze Zahlen
Short	short	16 bit	Ganze Zahlen
Long	long	32 bit	Ganze Zahlen
Long Long	long long	64 bit	Ganze Zahlen
Float	float	32 bit	Gleitkommazahlen
Double	double	64 bit	Gleitkommazahlen
Character	char	8 bit	Ganze Zahlen

Table 1: Primitive Datentypen in C

Für alle ganzzahligen Datentypen kann außerdem ein `unsigned` vorangestellt werden, um nur positive Werte zuzulassen und somit den Wertebereich "nach oben" zu verdoppeln.

Die einzelnen Speichergrößen sind zudem abhängig von der verbauten CPU sowie dem verwendeten Compiler. So können insbesondere die Speichergrößen von Integern und Longs variieren.

3.2 Variablen

Wenn man in C von einer Variable spricht, meint man damit eigentlich nur einen fest reservierten Platz im Arbeitsspeicher. In C haben Variablen außerdem einen festen Datentypen, den man bestimmt, wenn man die Variable **deklariert** (also dem Compiler mitteilt, dass diese Variable existiert). Um eine Variable zu deklarieren, schreibt man in C:

```
<Datentyp> <Variablenname>;
```

Beispiel 3.1. Um eine Variable `a` vom Typ `int` zu deklarieren, verwendet man in C den Aufruf:

```
int a;
```

Durch eine reine Deklaration wird der Variable allerdings noch kein Wert zugewiesen. Würde man die Variable nun so aufrufen, würde sie den Wert zurückgeben, der noch im Speicher an der Stelle vorhanden ist. Daher sollte eine Variable immer bei der Deklaration einen Wert zugewiesen bekommen:

```
int a = 0;
```

Diese Zuweisung nennt man **Initialisierung**.

3.3 Operatoren

Operatoren bieten unter anderem die Möglichkeit, Variablen miteinander zu verknüpfen. Es gibt dabei verschiedene Arten von Operatoren. Einige, die man man bereits aus der Mathematik kennt, sind die **arithmetischen Operatoren**:

Operator	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo (Rest einer Division)

Table 2: Arithmetische Operatoren in C

Beispiel 3.2. Gegeben sind zwei Variablen `a` und `b` und man möchte den Rest der ganzzahligen Division von `a` durch `b` berechnen. Der Code dazu könnte wie folgt aussehen:

```c
int a = 10;
int b = 3;
int rest = a % b; // = 1, da 10/3 = 3 Rest 1
```

Des Weiteren existieren auch noch **logische Operatoren**, welche dazu dienen, einen Wahrheitswert aus zwei oder mehreren Variablen zu schließen. In C gibt es dafür folgende **logische Operatoren**:

Operator	Bedeutung
&&	logisches Und
||	logisches Oder
!	logisches Nicht

Table 3: Logische Operatoren in C

Logische Operatoren werden häufig genutzt, um Bedingungen zu überprüfen. Dazu mehr in Kapitel 4.

Die dritte Art von Operatoren sind **Vergleichsoperatoren**. Diese verwendet man, um zwei Werte direkt miteinander zu vergleichen und geben daher ebenfalls einen Wahrheitswert zurück. In C gibt es folgende Vergleichsoperatoren:

Operator	Bedeutung
==	Gleich
! =	Ungleich
>	Größer
<	Kleiner
>=	Größer oder gleich
<=	Kleiner oder gleich

Table 4: Vergleichsoperatoren in C

Wichtig: `=` und `==` sind zwei verschiedene Operatoren! Das einfache Gleichheitszeichen wird verwendet, um einer Variable einen Wert zuzuweisen. Möchte man den Wert einer Variable mit einem anderen Wert auf Gleichheit prüfen, verwendet man dazu immer ein doppeltes Gleichheitszeichen.

Ein entsprechendes Codebeispiel zur Anwendung von Vergleichsoperatoren folgt auch hier in Kapitel 4.

4 Kontrollstrukturen

Ein Programm, das lediglich aus Variablen und Operatoren besteht, wäre relativ sinnfrei, da es keine Möglichkeit gäbe, den Verlauf des Programms auf die Eingaben des Benutzers oder auf

andere Gegebenheiten anzupassen. Das könnte zum Beispiel die Berechtigung eines Benutzers sein; ein Programm soll vielleicht andere Funktionen ausführen, wenn es sich bei dem Benutzer um einen Administrator handelt.

Um solche Bedingungen in den Verlauf des Programms einzubeziehen, gibt es sogenannte **Kontrollstrukturen**. Grundsätzlich lassen sich diese in zwei Kategorien unterteilen: **Bedingte Anweisungen** und **Schleifen**.

4.1 Bedingte Anweisungen

Eine bedingte Anweisung ist ein Codeblock, der nur dann ausgeführt wird, wenn eine bestimmte Bedingung erfüllt wird. Das Keyword in C dafür lautet `if` und ist wie folgt definiert:

```
if (<Bedingung>) {
  // Code, der ausgeführt wird, wenn die Bedingung erfüllt ist
}
```

Fig. 2: if-Statement in C

Beispiel 4.1. Gegeben ist eine Variable `alter`, die das Alter des Benutzers speichert. Das Programm soll nun einen Text ausgeben, wenn der Benutzer volljährig ist. Der Code dazu könnte wie folgt aussehen:

```
int main() {
  int alter = 18;
  if (alter >= 18) {
    printf("Du bist volljährig!\n");
  }

  return 0;
}
```

Führt man dieses Programm nun aus, erscheint der Text `Du bist volljährig!` in der Konsole. Ändert man den Wert der Variable `alter` aber beispielsweise auf `17`, passiert gar nichts. Das liegt daran, dass der Code innerhalb des `if`-Statements wirklich nur dann ausgeführt wird, wenn die Bedingung in den runden Klammern erfüllt ist.

Um das Programm dahingehend zu erweitern, dass auch ein Text erscheint, wenn der Benutzer nicht volljährig ist, gibt es in C das Keyword `else`. Der Code innerhalb eines else-Blocks wird immer dann ausgeführt, wenn die Bedingung im vorangestelltn `if`-Statement nicht erfüllt ist. Für das obige Beispiel könnte ein passender `else`-Block wie folgt aussehen:

```
int main() {
  int alter = 17;
```

```
  if (alter >= 18) {
    printf("Du bist volljährig!\n");
  } else {
    printf("Du bist minderjährig!\n");
  }

  return 0;
}
```

5 Funktionen