Einführung in die Programmiersprache C

Software Entwicklung 1

Annette Bieniusa, Mathias Weber, Peter Zeller

Dieses Skript gibt nur einen kleinen Einblick in C. Für eine ausführlichere und vertiefte Darstellung empfehlen wir:

- Dausmann, Bröckl, Goll, Schoop: C als erste Programmiersprache. Vieweg+Teubner Verlag. (auch als E-Book in der UB verfügbar!)
- Kernighan, Ritchie: The C programming language. Prentice Hall. 2. Auflage (Ein Klassiker, allerdings weicht ANSI C an vielen Stellen von C99 ab)

1 Eine Einführung in C

Die Programmiersprache C ist eine prozedurale und imperative Programmiersprache, die vielseitig einsetzbar ist. Die Anweisungen und Ausdrücke in C lassen sich einfach und effizient in Maschinencode übersetzen. C wird daher häufig in der Systemprogrammierung verwendet (Erstellung von Betriebssystemen, eingebetteten Systeme); es findet aber auch Einsatz in der Anwendungsprogrammierung.

Historie Dennis Ritchie entwickelte C zwischen 1969 und 1973, um es bei einer Re-Implementierung des Unix-Betriebssystems zu verwenden. Seitdem entwickelte es sich zu einer der meist verwendeten Sprachen. Mittlerweile gibt es zahlreiche Varianten und Versionen von C. Wir orientieren uns für die Vorlesung an C99, einem ISO-Standard, der 1999 veröffentlicht wurde und von den meisten aktuell verfügbaren C-Compilern übersetzt und ausgeführt werden kann.

Abgrenzung zu anderen Sprachen Viele Programmiersprachen, die nach der sehr erfolgreichen Einführung von C entwickelt wurden, orientieren sich an der Syntax und Struktur von C. Auch Java ist eine sogenannte ***C-ähnliche*** Sprache, die viele

syntaktische Gemeinsamkeiten mit C aufweist (z.B. Kontrollstrukturen, Methodendeklarationen), sich semantisch aber stark unterscheidet. Die Sprache C++, zu Beginn der 80er Jahre von Bjarne Soustroup entwickelt, ist eine "Geschwistersprache" zu C: Die meisten C Konstrukte werden auch von C++ unterstützt, so dass man C++ Programme schreiben kann, die auch gültige C Programme sind. Allerdings gibt es einige Inkompatibilitäten, die dem (strikteren) Typsystem von C++ geschuldet sind. C++ bietet außerdem objekt-orientierte Sprachfeatures und umfangreiche Erweiterungen (z.B. Klassen, Templates, Exceptions, Overloading, automatische Speicherverwaltung, Referenzen).

2 Unser erstes C Programm

Betrachten wir zum Einstieg das klassische Hello-World Programm in C:

```
1 #include <stdio.h>
2
3 /* Hello-World in C */
4 int main(void) // Hauptprogramm
5 {
6     printf("Hello world!\n");
7     return 0;
8 }
```

- In Zeile 1 bindet die Präprozessordirektive `#include <stdio.h>` die Ein-/Ausgabe-Bibliothek `stdio` (**st**andar**d** **i**nput/**o**utput) ein. Diese include-Anweisung veranlasst den C-Präprozessor, vor der Übersetzung die Header-Datei `<stdio.h>` in den Quelltext zu kopieren, die unter anderem eine Deklaration der weiter unten verwendeten Ausgabefunktion `printf` enthält. Include-Anweisungen werden in der Regel an den Anfang eines Programmtextes gestellt.
- Zeile 3 enthält einen Kommentar.
- In Zeile 4 beginnt das eigentliche Programm mit der Definition der Funktion `main`. Sie ist die Einstiegsfunktion eines jeden C-Programms. Die hier gezeigte Variante von `main` erwartet keine Programmparameter (`void` anstelle von Parametern).
- In Zeile 6 wird die Funktion `printf` aufgerufen, die den Text „Hello World!" gefolgt von einem Zeilen-Umbruch (`\n`) ausgibt.
- Die Anweisung `return 0` beendet den Funktionsaufruf; der Rückgabewert 0 indiziert in Unix-Systemen eine erfolgreiche Ausführung. Im Fehlerfall wird typischerweise 1 zurückgegeben. Seit C99 kann die Anweisung `return 0` auch entfallen. [1]

[1] Alternativ können die Konstanten `EXIT_SUCCESS` bzw. `EXIT_FAILURE` aus der Bibliothek `stdlib.h` verwendet werden. In Java ist kann übrigens durch die Methode `System.exit(int status)` die Programmausführung beendet und ein Fehlercode an das ausführende System zurückgegeben werden.

Compilieren, Linken und Ausführen von C Programmen Ein C Programm kann aus mehreren Programmeinheiten bestehen. Für jede dieser Programmeinheiten existiert eine Quellcode-Datei (mit der Endung .c) und evtl. eine Header-Datei (mit der Endung .h). Die Quellcode-Datei enthält im Wesentlichen die Implementierung, die Header-Datei die Schnittstelle nach außen. Programmeinheiten, die Funktionen aus anderen Programmeinheiten benutzen wollen, fügen deren Header-Dateien ein und geben dem Compiler damit die notwendigen Informationen über die vorhandenen Funktionen, Typen und Konstanten.
Jede Quellcode-Datei kann dann für sich übersetzt werden und erzeugt eine Object-Datei (mit der Endung .o, in Windows .obj). Mehrere Object-Dateien können zu einer Bibliothek zusammengefasst oder einzeln verwendet werden.
Mehrere Object-Dateien sowie Bibliotheken (= Sammlung von Objekt-Dateien) können durch den Linker (dt. Binder) zu einem ausführbaren Programm gebunden werden.
Die Übersetzung von C Programmen besteht aus den folgenden Schritten:

0. Der ***Präprozessor*** wird als Vorverarbeitungsschritt automatisch vom Compiler aufgerufen. Er führt die Präprozessor-Anweisungen aus (markiert durch #). Dabei werden beispielsweise der Inhalt der mit `#include` eingebundenen Header-Dateien in die zu compilierende Datei kopiert. Funktionen können in C erst verwendet werden, wenn zuvor in der Quelldatei deklariert oder definiert wurden; die `#include`-Anweisung erlaubt es daher existierenden Code wiederzuverwenden.

1. Der ***Compiler*** ruft den Präprozessor auf und übersetzt den vom Präprozessor verarbeiteten C Quellcode der Datei `foo.c` in eine Object-Datei `foo.o`:

   ```
   > clang -c foo.c
   ```

2. Der ***Linker*** fügt mehrere Object-Dateien (`foo.o, bar.o, baz.o`) zu einem Programm `myprog` zusammen:

   ```
   > clang foo.o bar.o baz.o -o myprog
   ```

   Dieses kann dann direkt ausgeführt werden:

   ```
   > ./myprog
   ```

Falls das Programm nur aus einer Datei besteht, kann man es in einem Schritt compilieren und linken:

```
> clang foo.c -o foo
```

Der Name nach `-o` ist der Name für die ausführbare Datei; wird diese Option weggelassen (`clang foo.c`), wird (aus historischen Gründen) als ausführbares Programm die Datei `a.out` (bzw. `a.exe` unter Windows) erzeugt.

Hinweise zur Übersetzung unter Windows: Die hier gezeigten Befehle können genau so mit `gcc` statt `clang` als Compiler ausgeführt werden:

```
> gcc -c foo.c
> gcc foo.o bar.o baz.o -o myprog.exe
> myprog.exe
```

Frage 1: Wie kann man das Hello-World Programm in einem Schritt übersetzen und dann ausführen, wenn es in der Datei `hello.c` gespeichert wurde?

3 Basisdatentypen, Operatoren, Variablen und Kontrollstrukturen

In C gibt es nur wenige elementare (numerische) Datentypen:

`short`, `int`, `long`	ganzzahlige Wert
`float`, `double`	Gleitkommawerte
`char`	Zeichen

Die `char`-Zeichen (i.d.R. durch 8 Bit dargestellt $\Rightarrow$ 256 Zeichen) werden anhand der ASCII-Tabelle umgerechnet.

Anders als in Java ist der jeweilige Wertebereich *maschinenabhängig* (Ausnahme: `char`); `short` darf allerdings nicht länger als `int` sein usw. Die Standard-Definitionsdateien `limits.h` und `floats.h` enthalten die konkreten Größeninformationen. Man unterscheidet bei den ganzzahligen Werten auch zwischen `signed` und `unsigned`; `unsigned` sind immer positiv oder 0.
Wenn der Wertebereich maschinenunabhängig sein soll, kann die Header-Datei `stdint.h` verwendet werden. Diese enthält die Typen `int8_t`, `int16_t`, `int32_t` und `int64_t` sowie die entsprechenden `unsigned` Typen `uint8_t`, `uint16_t`, `uint32_t` und `uint64_t`. Die Zahl im Typ gibt dabei jeweils die Größe der Zahl in Bits an und bestimmt somit den Wertebereich.

Auf diesen Datentypen kann man mit den üblichen arithmetischen und Vergleichsoperatoren arbeiten:

```
+  -  *  /  %  >  >=  <  <=  ==  !=
```

Es gibt weiterhin die logischen Operatoren `!` `&&` und `||`, die - wie auch in Java - von links nach rechts ausgewertet werden, und zwar nur solange, bis das Resultat feststeht. Auswertungsreihenfolge bei arithmetischen Operatoren ist nicht spezifiziert, falls keine Klammerung verwendet wird.
String-Literale werden im Programm, wie gewohnt, durch Hochkomma dargestellt (z.B. "Hello World"). Strings sind in C als `char[]`-Array implementiert, wobei das String-Ende durch ein spezielles Zeichen markiert wird. Stringmanipulation behandeln wir daher im späteren Abschnitt zu Arrays.

Boolesche Werte Es gibt keine dedizierten Boolschen Werte in C[2] Vergleiche wie `i > j` und auch logische Verknüpfungen sind vom Typ `int` und ergeben sich zu 1, falls sie wahr sind, und 0, falls nicht. In Bedingungen (bei `if`, `while`, `for` usw.) bedeutet "wahr" einfach "von Null verschieden". Es ist also gleichbedeutend zu schreiben

```
if (Ausdruck) {...}
```

oder

```
if ((Ausdruck) != 0) {...}
```

Variablen Die Variablendeklaration hat die gleiche Syntax wie in Java:

```
int x;        /* Deklaration ohne Initialisierung */
int y = 3;    /* Deklaration mit Initialisierung  */
```

Variablennamen in C bestehen aus Buchstaben, Ziffer und Unterstrich; sie dürfen nicht mit einer Ziffer beginnen. Leerzeichen u.ä. sind in Variablennamen nicht erlaubt. Außerdem dürfen Schlüsselwörter (`if`, `while`, ...) nicht als Variablennamen verwendet werden. Diese Regeln gelten auch für andere Bezeichner, z.B. Funktionsnamen.
Lokale Variablen sollten vor ihrer ersten Verwendung initialisiert werden; dies wird typischerweise **nicht** vom C-Compiler überprüft[3]. Das Lesen von nicht-intialisierten Variablen ist undefiniert und kann zu schwierig nachvollziehbaren Fehlern führen.

Undefiniertes Verhalten in C Der C Sprachstandard lässt das Verhalten eines Programms in vielen Fällen offen und undefiniert. Dies ermöglicht Compilern viele Optimierungen bei Übersetzung anzuwenden und so effizienten Code zu generieren. Es liegt allerdings an Ihnen, Programme zu vermeiden, die Konstrukte verwenden, deren Verhalten undefiniert ist. Sobald ein solches Konstrukt verwendet wird, ist die Semantik des gesamten Programms undefiniert!
Die Ausführung dieser Programme können zu Fehlern führen, die lange Zeit unerkannt bleiben und erst im weiteren Programmverlauf zu Programmabstürzen oder Speicherkorruption führen. Sie sind auch vielfach die Ursache für Sicherheitsprobleme und oft Angriffspunkte für Computerviren.

" Permissible undefined behavior ranges from ignoring the situation completely with unpredictable results, to having demons fly out of your nose." Siehe `http://blog.llvm.org/2011/05/what-every-c-programmer-should-know.html` und `http://www.catb.org/jargon/html/N/nasal-demons.html`.

Frage 2: Was passiert bei der Ausführung des folgenden Programms?

[2] C99 führt einen Datentyp `_Bool` ein, der nur die Werte 0 und 1 enthält. Über `stdbool.h` kann man den Typnamen `bool` und zwei Konstanten `true`, `false` verwenden.
[3] Mit der Option `-Wall` kann eine Überprüfung von nicht-initialisierten Variablen aktiviert für Compiler wie `gcc` oder `clang` aktiviert werden.

```c
#include <stdio.h>

int main(void)
{
    int y = 0;
    int x = 3/y;

    printf("Hello world!\n");
    printf("%d/n",x);
    return 0;
}
```

Einfache Eingabe/Ausgabe Die Bibliothek `stdio` enthält Funktionen zum Einlesen und Ausgeben auf die Konsole, in Dateien, etc.
Die Funktion `printf` ermöglicht die Ausgabe mit Formatierung auf die Kommandozeile. Als erstes Argument von `printf` sind *nur Strings* (Formatstrings) erlaubt. Alle weiteren, jeweils mit Komma getrennten Parameter, ersetzen die im String vorhandenen Formatelemente (gekennzeichnet durch %). Bei Gleitkommazahlen `%f` werden standardmäßig 6 Nachkommazahlen ausgegeben; dies kann durch Zusätze im Formatstring angepasst werden (wie in der Java-Übungsaufgabe zur formatierten Ausgabe (Blatt 4, Aufgabe 3)).

```c
#include <stdio.h>

int main(void)
{
  printf("Integer: %d\n", 42);
  printf("Gleitkommazahlen: %.6f\n", 3.141);
  printf("Zeichen: %c\n", 'z');
  printf("String: %s\n", "abc");
  printf("Prozentzeichen: %%\n");
  return 0;
}
```

Die Funktion `scanf` ist die äquivalente Funktion zum Einlesen von Werten. Der Formatstring enthält dabei die Platzhalter für die einzulesenden Werte. Die `scanf`-Methode benötigt außerdem die Speicheradresse, an denen diese Werte abgespeichert werden sollen (→ Abschnitt Speicheradressen im nächsten Kapitel).

```c
#include <stdio.h>

int main(void)
{
   int a;

   printf("Geben Sie einen Integer ein: ");
   scanf("%d", &a);
   printf("Sie haben eingegeben: %d\n", a);
```

```
10
11      return 0;
12  }
```

Beispiel: Umwandlung von Euro in Dollar Das folgende Programm liefert eine Umrechnungstabelle von Euro in Dollar.

```
1   #include <stdio.h>
2
3   int main (void)
4   {
5     double euro, dollar;
6     int lower = 0;
7     int upper = 100;
8     int step  = 10;
9
10    euro = lower;            /* Initialisierung von euro */
11    while (euro <= upper)
12    {
13      dollar = 1.05 * euro; /* Initialisierung von dollar */
14      printf("%6.2f  %6.2f\n", euro, dollar);
15      euro += step;
16    }
17  }
```

Wie im Beispiel angedeutet, bietet C die bereits bekannten Kontrollstrukturen für Verzweigungen (`if` und `if-else`), `while`-, `do-while`, `for`-Schleifen und Auswahlanweisung (`switch`) zur Strukturierung des Kontrollflusses.

> **Frage 3:** Ändern Sie das Programm zur Umwandlung von Euro in Dollar ab, sodass es eine `for`-Schleife statt der `while`-Schleife verwendet.

Hier eine Variante, bei der ein vom Nutzer eingegebener Betrag umgewandelt wird:

```
1   #include <stdio.h>
2
3   int main (void)
4   {
5     double euro = 0;
6     double dollar = 0;
7     double wechselkurs = 1.05;
8
9     printf("Betrag (Euro): ");
10    scanf("%lf",&euro);     /* Einlesen des double-Werts */
11    dollar = 1.05 * euro;  /* Kurs vom 04.01.17 */
12    printf("Betrag (Dollar): %6.2f\n", dollar);
13    return 0;
14  }
```

Hinweis: In der gezeigten Implementierung wird die Nutzereingabe **nicht** überprüft. Es werden vom Eingabestrom der Kommandozeile nur solange Zeichen konsumiert, wie sie zur Darstellung eines Double-Werts gehören. Die Verwendung von `scanf` kann außerdem beim Einlesen von Strings sicherheitskritisch sein; wir zeigen im nächsten Kapitel eine sichere Alternative.

Hinweise zu den Fragen

Hinweise zu Frage 1: `> clang hello.c -o hello > hello`

Hinweise zu Frage 2: Die Semantik ist undefiniert - probieren Sie es doch einfach mal aus. Es wird schon nicht Ihre Festplatte formatieren...

Hinweise zu Frage 3:

```c
#include <stdio.h>

int main (void)
{
  double euro, dollar;
  int lower = 0;
  int upper = 100;
  int step  = 10;

  for(euro = lower; euro <= upper; euro += step)
  {
    dollar = 1.05 * euro; /* Kurs vom 04.01.17 */
    printf("%6.2f  %6.2f\n", euro, dollar);
  }
}
```