

Einführung in C

Die Sprache

The C Programming Language:

- ▶ 1972 erschienen
- ▶ Prozedural-strukturierte Programmiersprache (imperativ!)
- ▶ Für Betriebssystementwicklung entworfen
- ▶ Nahezu auf jedem Gerät verfügbar

Hello World

```
1 EXTERN puts      ; #include <stdio.h>
2 GLOBAL main      ; Symbolexport
3
4 ; (globale) Variablen
5 SECTION .data
6 pstr: DB `Hello_World\0` ; Stringkonst.
7
8 ; Code
9 SECTION .text
10 main:                ; Programmeintritt
11     mov rdi, pstr     ; Funktionsaufruf
12     call puts
13
14     mov rax, 0        ; Exit-Code
15     ret
```

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     puts("Hello_World");
6     return (0);
7 }
```

Programmausführung

- ▶ In für den Computer verständliche Sprache übersetzen (kompilieren) mit

```
$ cc -o programmname programmname.c
```

- ▶ Ausführen mit: `$./programmname`

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     puts("Hello World");
6     return (0);
7 }
```

Was haben wir da geschrieben?

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     puts("Hello World");
6     return (0);
7 }
```

- ▶ Zeile 1: Möchten Funktionalität für Ein-/Ausgabe benutzen
- ▶ Zeile 3–7: Definition `main()`
 - ▶ Zeile 3: Signatur: Rückgabewert und Parameter
 - ▶ Zeile 4,7: Beginn und Ende des Funktionsblockes
 - ▶ Zeile 5: Funktionsaufruf, gibt Argument auf der Konsole aus
 - ▶ Zeile 6: Beenden Funktion, geben 0 zurück

Warum genau so?

Wir schreiben in C, in einer „Hosted“ Umgebung

- ▶ D. h. Wir haben die C-Standardbibliothek zur Verfügung
- ▶ Diese stellt uns `stdio.h` mit `puts()` zur Verfügung
- ▶ Außerdem schreibt dies uns vor, dass der *Programmeintrittspunkt* `main()` heißt und auch seine Signatur
- ▶ Um zu verstehen, warum das vorgeschrieben sein muss, müssen wir uns überlegen, wie unser Programm überhaupt gestartet wird

Was ist ein Programm?

- ▶ Ein Betriebssystem, Firmware eines Mikrokontrollers, ...
- ▶ Wir schreiben hier (im weiteren Sinne) *Anwendungsprogramme*
- ▶ D. h. es gibt einen Nutzer, der den Rechner bedient und euer Programm startet
- ▶ Dazu ist ein Betriebssystem notwendig, wir verwenden hier Linux.
- ▶ Unsere Programme haben (zum. vorerst) keine graphische Oberfläche, sondern werden im Terminal ausgeführt

Programmumgebung: Allgemein

- ▶ Ein Programm besteht grob aus einer Menge an Funktionen
- ▶ Die Funktion mit der die Programmausführung beginnt, ist der *Programmeintrittspunkt*
- ▶ Nur sie wird vom Betriebssystem aufgerufen, aber sie kann andere Funktionen innerhalb des Programmes ausführen
- ▶ ... oder vorgefertigte Funktionen, als Teil von „Bibliotheken“

Programmumgebung: C, Hosted

- ▶ Der Programmeintrittspunkt heißt `main()`
- ▶ Für die Signatur dieser Funktion gibt es mehrere Möglichkeiten:
 - ▶ `int main(void)`
 - ▶ `int main(int argc, char *argv[])`
 - ▶ `int main(int argc, char **argv)`
 - ▶ `int main(const int argc, const char *const argv[argc+1])`
- ▶ Was diese bedeuten gucken wir uns gleich genauer an
- ▶ Jede Bibliothek bietet besteht außerdem sog. „Header“, welche beschreiben, welche Funktionen sie bereitstellt

Eigene Programme schreiben

- ▶ Wir schreiben erstmal nur die Funktion `main()` und schreiben keine neuen anderen Funktionen
- ▶ Stattdessen beschäftigen wir uns mit einfachen Anweisungen und Variablen
- ▶ Um Variablen zu benutzen, muss man sie dem Compiler bekannt machen (*deklarieren*)
- ▶ Sowohl Anweisungen als auch Deklarationen werden mit Semikoli beendet.

```
1 float pi, daumen;           // (Fließ-)Kommazahlen
2 daumen = 13.372;
3 pi = 3.141;
4 int antwort = pi*daumen;    // ~42
```

Formatierte Ausgabe

- ▶ Um berechnete Ergebnisse auszugeben, müssen sie in eine menschenlesbare Form umgewandelt werden
- ▶ `printf()` aus `stdio.h` ist eine mächtige Funktion, die das kann
- ▶ Dazu wird als erstes Argument ein sog. Formatstring übergeben, dieser bestimmt *wie* ausgegeben wird
- ▶ Bspw.: `printf("i ist: %d\n", i);`
 - ▶ Das `"%d"` ist ein Platzhalter, der das nächste Argument als Dezimalzahl interpretiert ausgibt
 - ▶ Das `"\n"` ist eine spezielle Escapesequenz welche die Zeile beendet (newline)
 - ▶ Der Rest in dem String wird direkt so ausgegeben
 - ▶ Für $i = 42$ also: „i ist: 42“

Schleifen

- ▶ Manchmal möchte man wiederholt die gleichen Anweisungen ausführen
- ▶ Hierzu bietet C unterschiedliche Schleifen, heute werden wir kurz `for` vorstellen
- ▶ Vorher nutzen wir eine andere Signatur von `main()`:
`int main(int argc, char *argv[])`
- ▶ `argc` („Argument Count“) gibt an wie viele Argumente dem *Programm* auf der Konsole übergeben wurden
- ▶ `argv` („Argument Vector“) hält eben diese als Strings vor

Schleifen II

```
1 int main(int argc, char *argv[])
2 {
3     for (int i = 0; i < argc; i++) {
4         printf("Arg %d: %s\n", i, argv[i]);
5     }
6     return (0);
7 }
```

- ▶ Die `for`-Anweisung besteht aus 3 Ausdrücken:

```
for ( <Initialisierung> ; <Bedingung> ; <Inkrement> )
```

- ▶ In einem weiteren Block: Die zu wiederholende Anweisung

Ausführung

Ausführung auf der Konsole:

```
1 $ ./prog Das sind meine Argumente "in Anführungszeichen"  
2 Argument 0: ./prog  
3 Argument 1: Das  
4 Argument 2: sind  
5 Argument 3: meine  
6 Argument 4: Argumente  
7 Argument 5: in Anführungszeichen
```

If-Abfrage

- ▶ Nimmt nur einen Ausdruck und führt den Block aus, wenn es wahr ist, sonst nicht
- ▶ Falls darauf ein `else` folgt, wird anstelle dieser Block ausgeführt

```
1 int main(int argc, char *argv[])
2 {
3     if (argc < 2) {
4         printf("Ein Argument übergeben\n");
5     } else {
6         printf("Mehr als ein Argument übergeben\n");
7     }
8     /* alles folgende wird so oder so ausgeführt */
9     printf("Hallo Welt!\n");
10 }
```

Parsen von Zahlen: atoi()

Die übergebenen Argumente liegen in Form von Zeichenketten vor. Um eine Zeichenkette einer Dezimalzahl in einen Integer umzuwandeln gibt es die Funktion `atoi()`:

```
1 #include <stdlib.h> // int atoi(char *str);
2
3 int main(void)
4 {
5     int a = atoi("3"); // a = 3
6 }
```


Weiterführende Quellen I

- ▶ Diverse Autoren.
Wikibooks

https://en.wikibooks.org/wiki/C_Programming

- ▶ Jens Gustedt.
Modern C

<http://icube-icps.unistra.fr/index.php/File:ModernC.pdf>