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 auf der Konsole: `$ ./programmname`

```c
#include <stdio.h>

int main(void)
{
  puts("Hello World");
  return (0);
}
```

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 Microkontrollers, ...
- 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
- Wir nutzen diese 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

```c
int main(int argc, char *argv[])
{
    for (int i = 0; i < argc; i++) {
        printf("Arg %d: \"%s\"\n", i, argv[i]);
    }
    return (0);
}
```

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

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

- In einem weiteren Block: Die zu wiederholende Anweisung

Ausführung

Ausführung auf der Konsole:

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

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 < 3) {
4          printf("Ein Argument uebergeben\n");
5      } else {
6          printf("Mehr als ein Argument uebergeben\n");
7      }
8      /* alles folgende wird so oder so ausgefuehrt */
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()`:

```c
#include <stdlib.h> // int atoi(char *str);

int main(void)
{
  int a = atoi("3"); // a = 3
}
```

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
- C Standards Committee
  C11 Standard Darft
  http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf
- OpenGroup
  POSIX 2008 Standard
  http://pubs.opengroup.org/onlinepubs/9699919799/