Intro

Kompilieren

Der C-Compiler wird auf der Kommandozeile mit `cc` aufgerufen. Er nimmt u. A. diese Optionen:

`-o` *output*	Der Compiler wir angewiesen das Resultat (kompiliertes Programm) als „output“ zu speichern.
`-c`	Es wird keine ausführbare Datei erstellt, sondern lediglich eine übersetzte `.o`-Datei.
`-l`*foo*	Es wird die Bibliothek „libfoo“ dazugelinkt werden, diese muss in einem Suchpfad für Bibliotheken vorhanden sein
`-L`*/path/to/libs*	Der Pfad „/path/to/libs“ wird als Suchpfad für Bibliotheken hinzugefügt
`-I`*/path/to/incs*	Analog wird dieser Pfad für Header-Dateien hinzugefügt

Zusätzlich werden die zu kompilierenden Objekte als Operand dazu übergeben, bspw.:

```
1 $ cc -o programm source1.c source2.c ...
```

Während `cc` der standardmäßig als der auf dem System verwendete C-Compiler eingestellt ist, gibt es auf vielen System mehrere Compiler. Auf den Poolrechnern sind das `gcc` und `clang`, wobei ersteter als `cc` eingesetzt wird.

Diese Compiler unterstützen viele spezifische Flags, nützlich sind vor allem die, die den Compiler anweisen, einen bestimmten C-Standard zu benutzen:

`-std=`*c11*	Benutzt bei der Kompilation den Standard C11, weitere sind C99, C89 und nicht standardkonforme GNU-CC-Varianten davon.
`-pedantic`	Lässt den Compiler „strikt“ nach Standard arbeiten und lässt keine nicht-standardkonformen Programme zu.

Außerdem kann man Diagnostics anschalten, also Warnungen bei Code der möglicherweise falsch ist:

`-Wall`	Schaltet viele sinnvolle Warnungen an
`-Wextra`	… und noch mehr
`-Weverything` (Clang)	Schaltet *alle* Warnungen an

C Hosted Environment

Programmeintrittspunkte

Es gibt verschiedene Möglichkeiten den Programmeintrittspunkt zu schreiben:

- `int main(void)`
- `int main(int argc, char *argv[])`

Erstere gibt an, dass das Programm alle vom Nutzer übergebenen Parameter ignoriert, da die Funktion `main()` keine Möglichkeit hat auf diese zuzugreifen. Möchte man dies jedoch tun, nutzt man die andere Variante, hierbei ist `argc` der Argument Counter, und gibt somit an wie viele Argumente der Nutzer übergeben hat. Der Argument Vector `argv` hält die eigentlichen Argumente als Zeichenketten vor, wobei `argv[0]` (das erste Element im „Array“) der Programmaufruf ist, also:

```
$ ./programm arg1 "arg2 in quotes"
argv[0]: "./programm"
argv[1]: "arg1"
argv[2]: "arg2 in quotes"
```

Die zweite Variante der `main()`-Funktion kann man auch anders schreiben:

- `int main(int argc, char **argv)`
- `int main(const int argc, const char *const argv[argc+1])`

C-Standardbibliothek: stdio.h

Der Standard-I/O–Header enthält diejenigen Funktionen der C-Standardbibliothek, die wichtig für einfache Ein- und Ausgabe (i. d. R. auf der Konsole) sind.

Die Funktion `puts()` Nimmt einen String und gibt ihn auf der Standardausgabe `stdout` aus, und beendet die Zeile mit einem `'\n'`.

Deklaration:

```
int puts(const char *s);
```

Die Funktion `printf()` Nimmt als erstes Argument einen Formatstring. Enthält dieser Platzhalter werden die dafür benötigten Werte in den weiteren Argumenten in Reihenfolge der Platzhalter übergeben. Hängt *kein* terminierendes `'\n'` an!

Deklaration:

```
int printf(const char *format, ...);
```

```
float pi, daumen;
pi = 3.141;
daumen = 13.374;
int antwort = pi*daumen;

printf("Die Antwort auf die Frage nach dem Leben, "
/* Stringkonstanten die aufeinanderfolgen werden einfach
   zusammengesetzt und zählen als eine lange Konstante          */
       "dem Universum und dem ganzen Rest ist: %d\n", antwort);
```

Um `float` auszugeben benutzt man den Platzhalter `%f`, für `char*` (also Strings) `%s`.

Kontrollstrukturen

Schleifen

`for`-Schleife Gut geeignet zum iterieren mit Zählvariablen. Dazu werden drei Ausdrücke benutzt, eine Initialisierung, eine Schleifenbedingung und ein Inkrement. Die Initialisierung wird zuerst ausgeführt, danach wird die Schleifenbedingung getestet. Ist sie wahr, wir der Schleifenkörper ausgeführt, danach der Inkrement ausgewertet und dann wieder die Bedingung ausprobiert – solange, bis die Bedingung falsch wird.

Syntax:

```
for ( Ausdruck 1 ; Ausdruck 2 ; Ausdruck 3) {
    /* Anweisungen */
}
```

`while`-Schleife Simpler als die `for`-Schleife, hat nur eine Schleifenbedingung. Muss initialisiert oder inkrementiert werden, muss der Programmer das selber machen.

Syntax:

```
while ( Ausdruck 1 ) {
    /* Anweisungen */
}
```

Beispielcode

```c
#include <stdio.h>

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

    return (0);
}
```

Codebeispiel 1: Programmargumente ausgeben

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    if (argc < 2) {
        /**
     * fprintf(stdout, ...) == printf(...)
     * stdout: Standardausgabe
     * stderr: Standardfehlerausgabe
```

```
11      */
12         fprintf(stderr, "Maximum nicht angegeben!\n");
13     /**
14      * Exit-Code != 0: Fehler wird an den Nutzer/ das
15      * Betriebssystem signalisiert
16      */
17     return (1);
18     }
19
20     /* max wird nicht mehr verändert, bleibt konstant */
21     const int max = atoi(argv[1]);
22     if (max == 0 || max < 0) {
23     fprintf(stderr, "Keine Zahl oder Zahl <= 0 übergeben!\n");
24     return (1);
25     }
26
27     for (int i = 1; i <= max; i++) {
28         for (int j = 1; j <= max; j++) {
29             printf("%d*%d = %d\n", i, j, i*j);
30         }
31         if (i < max) {
32              printf("\n");
33         }
34     }
35 }
```

Codebeispiel 2: Einmaleins mit anzugebenden Maximum