Literatur

- von Studenten empfohlen
  - Helmut Erlenkötter: C Programmieren von Anfang an
  - “ANSI C for Programmers on UNIX Systems”, http://www-h.eng.cam.ac.uk/help/tpl/languages/C/teaching_C/
- Standardwerk:
  - W. Kernighan, Dennis M. Ritchie: The C Programming Language
  - auf Deutsch: Programmieren in C
- C Crashkurs: optionaler Stoff: optional/05-1_C.pdf

Das Buch “Grundlagen der Informatik” hat auch einen C Abschnitt.

Handling von Variablen

- Prozess- / Speichermodell:
  - Globale Variablen
  - Heap
  - Stack
- Stackmodell
- Parameterübergabe
- Rückgabewert auf Stack
- Rücksprungadresse auch auf Stack!
- "Runtime" Parameterhandling von C Implementiert!

Einfache Deklarationen

- Einfache Typen:

```
int, char, long, ...
```

- Deklaration:

```
int zahl;
char buchstabe;
```

- Funktionsdefinition:

```
int f(int z) { return z + z; }
```

- Parameterübergabe *immer* “by value” auf dem Stack:

```
f(zahl);
```

siehe https://en.wikipedia.org/wiki/C_data_types

Pointer

- warum Pointer?
  - wir haben: Maschinensprache – wie implementieren wir ein Array? → Basis Pointer plus Index/Offset

-
```
int   zahl;
int*  zahl_p;              /* Pointer auf Integer*/
char* buchstaben_p;        /* oft ein Pointer auf
                              einen String! */
void f(int* z_p) { ... };/* Funktion */
```

Pointer

- Referenzieren

```
zahl_p = &zahl;
```

`&` -> "Adresse"
`zahl_p` enthält nun die Adresse der Variable `zahl` **zeigt** (Pointer = Zeiger) also auf diese Variable

- Dereferenzieren

```
zahl = *zahl_p;
```

dem Pointer `zahl_p` folgen und den Inhalt von dort wo er hinzeigt in die Variable `zahl` schreiben

- Übergeben wird *immer* Wert (d.h. hier wird **Adresse** als *Wert* übergeben):

```
f(zahl_p);
```

Strukturen

```
struct musik_note {
  char bezeichner;  /* A */
  int  frequenz;    /* 440 Hertz */
};
```

Evolution zu Objekten?

Werkzeuge: gcc

gcc
C Compiler
ruft automatisch Linker auf

```
$ ls
prog.c
$ gcc prog.c
$ ls
prog.c    a.out
$ ./a.out
[compiliertes prog.c wird ausgeführt]
```

mehr Info:

```
$ man gcc
$ info gcc
```

Werkzeuge: make

make

automatisches Erstellen von Programmen (und anderem)

verwendet die Datei '`Makefile`'

```
$ cat Makefile
prog: prog.c
   gcc prog.c -o prog
$ make
[ Compilation beginnt, "prog" wird erstellt ]
```

Werkzeuge: make

```
prog: prog.c
    gcc prog.c –o prog
```

- **`prog`** ist ein Ziel bzw. eine Datei
- die Erstellung von **`prog`** hängt vom Vorhandensein von **`prog.c`** ab
- das heisst auch, dass **`prog`** neu erstellt werden muss, sobald sich **`prog.c`** ändert
- das Rezept, um **`prog`** zu erstellen folgt in der nächsten Zeile und ist eine Shell Anweisung
- die Rezept Zeile ***muss*** zwingend mit einem Tabulator anfangen
- im obigen Rezept sieht man, dass der Compiler das Programm namens **`prog`** erstellt - somit ist das Ziel erreicht.