C-Appendix

Benjamin Steffen, Nico Fröhlich

July 16, 2019

Contents

1 Die Programmiersprache C

C wurde 1972-1973 von Dennis Ritchie an den Bell Labs entwickelt. C ist eine systemnahe Sprache, da sie sehr nahe an der Hardware liegt. Sie gehört zu den am weitesten verbreiteten Sprachen und findet Anwendung in Betriebssystemen, Nutzeranwendungen, Integrierten Systemen uvm..

1.1 Datentypen

C kennt folgende primitive Datentypen:

Datentyp	**Speichergröße**	**Wertebereich**
bool	1 Byte	0 oder 1
char	1 Byte	-128 bis 127
short	2 Bytes	-32,768 bis 32,767
int	2 Bytes	-32,768 bis 32,767
long	4 Bytes	-2,147,483,648 bis 2,147,483,647
long long	8 Bytes	-9223372036854775808 bis 9223372036854775807
float	4 Bytes	1.2E-38 bis 3.4E+38 (6 Dezimalstellen)
double	8 Bytes	2.3E-308 bis 1.7E+308 (15 Dezimalstellen)

Bis auf float und double haben all diese Datentype auch eine *unsigned* Variante. Das bedeutet, dass dann keine negative Werte, aber doppelt so viele positive Werte möglich sind. Desweiteren gibt es noch den Datentyp *void*, welcher das Fehlen eines Datentyps darstellt. Außerdem gibt es keinen booleschen Datentyp, generell stellt 0 den Wert *false* dar, während alles andere *true* ist

1.2 Literals

Literal	**Nutzung**	**Beispiel**
0b	Binärschreibweise	0b0001 für 1
0x	Hexadezimalschreibweise	0x000F für 16

1.3 Pointer

Ein Merkmal und mächtiges Werkzeug von C sind *Pointer*. Pointer sind Variablen die keine alleinstehenden Werte beinhalten, sondern auf die Speicheradresse einer anderen Variable zeigen. Die Adresse einer variable kann mithilfe des *&-Operators ermittelt werden*. Der **-Operator* hat zwei Aufgaben. Zum einen zeigt er an, dass eine Variable ein Pointer ist, zum anderen nutzt man ihn um den Wert eines Pointers auszulesen (Dereferenzieren). Hat man also beispielsweise einen Integerwert in einem **int wert = 22** gespeichert, so kann man den Ort, an dem sich dieser Wert befindet in einem Pointer **int *ptr = &wert** festhalten. Hierbei muss beachtet werden, dass weiterhin der Datentyp angegeben wird. Grund dafür ist die unterschiedliche Größe der Datentypen. Intern wird zwischen ihnen nicht unterschieden, deshalb muss dem Compiler gesagt werden, wie viele Bytes gelesen oder geschrieben werden sollen.

1.4 Strings

In C werden Zeichenketten (Strings) wörtlich genommen dort gibt es nämlich keine String klasse wie in Java. Hier werden lediglich ein "array" an Zeichen verwendet bzw. ein **char***. Wobei die Zeichenkette bei dem ASCII code 0 bzw **"\0"** endet. Angegeben wird diese wie in Java im code mit "
Beispiel: **char* string = "HALLO";**.

1.5 Syntax

Im folgenden wird kurz aufgezeigt, wie das Deklarieren und Verwenden von Variablen und Kontrollstrukturen aussieht.

```
#include <stdio.h> //Einbinden eines Headers

int main(int argc, char** argv) { //Definition des Programmeinstiegpunktes
    int zahl = 12345;             //Deklaration verschiedenster Variablen
    float kommazahl = 123.45;
    char charakter = 'a';
    char *text = "Beispiel Text\0";  // \0 Markiert das Ende eines Characterstrings,
                                     // ist aber je nach implementation nicht immer nötig
    for(int i=0; i<10; i++) {
        printf("Hello World! %d", i); //10 malige ausgabe eines Textes,
                                      //mit integer Wert in einem Formatstring
    }

    while(zahl>0) //Blöcke {} sind nicht notwendig,
        zahl--;   //wenn unter dem Schleifenkopf nur eine Anweisung steht

    return 0; //Die Funktion ended mit "return 0;" um erfolgreiche Abarbeitung zu melden
}
```

1.6 Header und Source

Im Vergleich zu Java gibt es in C zwei "Phasen" beim Compilern. Den sog. Linker und den normalen Compiler. Dabei Compiliert der Compiler den Code während der Linker nur die Definitionen von Funktionen und Header Variablen hat. Der Linker verbindet die jeweiligen Dateien nach dem Compilieren vernünftig. Das wichtige dabei ist das man generell nur Deklarationen in die Header macht und diese dann auch "included". Siehe Folgendes Kapitel.

1.7 C8051F340.h

Siehe: https://github.com/MrTroble/ADC-C8051F340/blob/master/C8051F340.h

Der C8051F340 Header ist ein von Cygnal/SiLabs bereit gestellter Header der bereits alle ASM Register mit ihren jeweiligen Adressen enthält. Dabei koenne die jeweiligen Register wie normale variablen mit bit Operatoren angesteuert werden. Beispiel:

```
#include <C8051F340.h>

void main() {

    ADC0CN |= 0x80; // Enable ADC0 subsystem
    AMX0N = 0x1F; // Set negative mux to GND -> use singel end mode
    // use AMX0P default config -> P1.0
    while (1) { // Unlimited loop
        ADC0CN |= 0x10; // or 16 to trigger AD0BUSY
        while (!(ADC0CN & 0x20 /* or 32*/)) {} // wait for the AD0INT to become 1
        // go ahead after thread block
        P3 = ADC0L; // Put low bits in P3
        P0_0 = ADC0H & 1; // Put first bit of high bits into P0.0
        P0_1 = ADC0H & 2; // Put second bit of high bits into P0.1
    }

}
```

2 Advanced

2.1 SDCC - Small Device C Compiler

Siehe: http://sdcc.sourceforge.net/doc/sdccman.pdf