C für Unverdrossene

Michael Dienert

8. Dezember 2016

Vorbemerkung

Natürlich könnte ich als erstes C-Programm “Hello World” nehmen. Um mit dem Kompiler zu spielen, kann man aber auch gleich einen sinnvolleren Quelltext übersetzen lassen. Das im folgenden beschriebene Programm wandelt alle Kommas einer ASCII-Datei in Punkte um.

1 Der Quelltext

Quelltexte lassen sich in ein LaTeXDokument am einfachsten mit der *verbatim*-Umgebung einbinden. Bild 1 zeigt das Ergebnis. Das C-Programm steht am Ende des Texts.

2 So wird compiliert

Compiliert wird das Programm mit folgender Kommandozeile:

```
gcc col2dot.c -0 col2dot
```

gcc ist der Compiler: GNU C Compiler [1]. Natürlich muss man dem Compiler noch sagen, was er kompilieren muss, man übergibt also noch den Namen der Quelldatei, *col2dot.c*.

Optional kann man GCC noch sagen, wie die ausführbare Datei heissen soll. Erreicht wird dies mit der Option -o *name*.

Das sieht zunächst sehr einfach aus, aber das Handbuch des GCC hat ca. 470 Seiten, wenn man es mit TeX übersetzt und ausdruckt. Es gibt beliebig viele, beliebig komplizierte Optionen. Ein paar davon werden wir später verwenden.

Warum nimmt der Dienert jetzt so einen umständlich zu bedienenen Compiler, und nicht so ein schönes, buntes Sinnlos-Teil?

1. GCC ist gratis, die Handbücher werden mitgeliefert und lassen sich mit TeXausdrucken.
2. GCC compiliert für nahezu jeden Prozessor und unter jedem OS.
3. Die Beispiele des Buchs *C++, Eine Einführung* von *Ulrich Breymann* (Informatik-Prof, Uni Bremen) sind alle mit GCC entwickelt.

[1] GNU steht für GNU is Not Unix. Eine typische LINUX-Distribution besteht aus 3% LINUX (der Kernel) und ca. 30-40% GNU-Software, ohne die ein LINUX-System nicht funktionieren würde.

4. Der Dienert schreibt seine Texte schliesslich auch nicht mit WORD.

3 Kommentare

Das Programm beginnt mit drei Kommentarzeilen:

```
/* Kommentarblock, auch über
mehrere Zeilen */
```

Eine weitere Möglichkeit Kommentare im Programmtext unterzubringen sieht so aus:

```
// leitet Kommentar bis Zeilenende ein
```

4 Anweisungen an den Preprozessor

Anweisungen an den sog. Preprozessor beginnen mit einem #. Sie werden als *Kontrollzeilen* bezeichnet.

Der Preprozessor ist ein Programm, das *vor* dem eigentlichen Kompiler über den Quelltext läuft und nur die Anweisungen der Kontrollzeilen ausführt. Zum Beispiel die Kontrollzeile:

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

Diese Kontrollanweisung bewirkt, dass der Inhalt der Datei *stdio.h* an dieser Stelle in den Quelltext eingefügt wird. Zumindest kann man sich das so vorstellen.

stdio.h ist hierbei die Abkürzung für *Standard Input Output Headerfile*. Headerfile ist eine Definitionsdatei die per Konvention (ANSI-C) auf ".h" enden muss. Beim geplanten C++-Standard, wird das ".h" weggelassen.

stdio.h enthält nun alle Deklarationen von Standard Ein-Ausgabefunktionen. Die vom Programm benötigten Funktionen werden während des Kompilierens aus einer sog. *Bibliothek*, engl. *Library*, zum Programm hinzugefügt.

Durch das Einbinden der Header-Datei sind die Bibliotheksfunktionen dem Compiler von Anfang an bekannt. Wäre das nicht der Fall, würde er beim ersten Aufruf einer dieser Funktionen mit einer Fehlermeldung abbrechen.

Der Linker

Der Teil des Compilers, der die Bibliotheken hinzufügt, ist der sog. *Linker*. Von GNU *ld* genannt. Der Linker kann auch separat gestartet werden. Seine Bedienung und seine Fehlermeldungen sind ein Kapitel für sich. Ld ist auf jeden Fall geeignet, einen unbedarften LINUX-Anwender wie mich zur Verzweiflung zu bringen. Grund sind die Bibliotheken, die ein Programm benötigt. Ld muss die richtigen Pfadangaben kennen und die muss man erstmal selbst herausfinden. Vor allem, weil es unüberschaubar viele sog. lib-Verzeichnisse gibt. Zumindest auf meiner Installation.

5 Die main-Funktion

Mit dem Begriff *Funktion* wird in C immer ein Unterprogramm bezeichnet. Auch *main* ist nicht etwa *das* Hauptprogramm, sondern ebenfalls ein Unterprogramm. Und zwar ein Unterprogramm, das vom Betriebssystem aus aufgerufen wird.

Und wie jedem Unterprogramm kann man auch *main* Parameter, man sagt auch *Argumente*, übergeben. Ebenso gibt *main* Parameter ans Betriebssystem zurück.

Vom OS erhält *main* genau zwei Argumente:

argc und *argv*

argc ist dabei als Integer-Variable deklariert und enthält die Anzahl der "Wörter" (alles was durch Leerzeichen getrennt ist) einschliesslich des Programmnamens, wenn man das Programm mit einer Komandozeile startet. Beispiel: starten wir unser Programm so:

```
col2dot altedatei neuedatei
```

hat *argc* den Wert *3*.

argv ist ein Feld von Zeigern. Das mit den Zeigern ist etwas kompliziert und kommt erst viel weiter hinten, deshalb reicht an dieser Stelle folgendes:

- `argv[0]` liefert den Programmnamen, im Beispiel also *col2dot*.
- `argv[1]` liefert das nächste Wort: *altedatei*.
- Und so weiter. `argv[2]` liefert also: *neuedatei*.

5.1 Kurzer Pointerausflug

Für diejenigen, die es jetzt schon genauwer wissen wollen, muss ich nun doch etwas weiter ausholen:

Ein *Pointer*, ein Zeiger ist nichts anderes als eine Adresse im Hauptspeicher. *argv* enthält also nicht direkt die jeweiligen Wörter sondern einen Zeiger auf die Stelle im Hauptspeicher, an der die Wörter stehen.

Genauer gesagt ist *argv* ein *Character-Pointer*. Character-Pointer werden so deklariert:

```
char *pmessage;
```

Bei der Deklaration teilt man dem Compiler durch das Sternchen (*) mit, dass *pmessage* die Adresse einer char-Variablen, genauer gesagt, die Anfangsadresse einer Zeichenkette, ist.

Hat man im folgenden Programm z.B. diese Zuweisung:

```
pmessage = ``hello world''
```

dann legt der Compiler an einer freien Stelle im Speicher die Zeichenkette (String) "hello world" ab und trägt die *Adresse des ersten Zeichens* in *pmessage* ein.

Und woher weiss man anschliessend, wo der String zu Ende ist?

Der Compiler schreibt direkt hinter die Zeichenkette eine *Null* in den Speicher!

6 Standard Ein-Ausgabe Funktionen in C

C stellt eine Reihe von recht einfach anwendbaren Funktionen zur Verfügung, um lesend oder schreibend auf Dateien zugreifen zu können. Diese Funktionen hat man aber nur zur Verfügung, wenn man über $< stdio.h >$ die entsprechende Bibliothek eingebunden hat.

Hier eine kurze Beschreibung der wichtigsten Dateifunktionen:

- *FILE *fopen(char *fname, char *modus)*

  Öffnet die Datei mit dem Namen *fname*. *modus* ist z.B. "w", wenn zum Schreiben (write) geöffnet werden soll und "r" (read) steht entsprechend für Lesebetrieb.

  Der Rückgabewert dieser Funktion ist ein Zeiger, der auf eine Struktur vom Typ FILE zeigt.

  Im Fehlerfall liefert die Funktion jedoch einfach eine Null zurück.

- *FILE *fclose(FILE *fzeiger)*

  Schliesst die Datei auf die mit dem Zeiger *fzeiger* zugegriffen wurde.

- *int fgetc(FILE *fzeiger)*

  Diese Funktion liest das nächste Zeichen aus einer Datei. Achtung: ihr Rückgabewert ist ein Integer! D.h. die Funkion liefert eigentlich nicht das nächste Zeichen, sondern den *ASCII-Code* des nächsten Zeichens!

  Im Fehlerfall, oder wenn das Dateiende erreicht ist, liefert die Funktion -1 zurück.

- *int fputc(char c, FILE *fzeiger)*

- *fputc()* schreibt das Zeichen c in die durch *fzeiger* angegebene Datei.

  Rückgabewert: -1 bei einem Schreibfehler.

- *fprintf(FILE *fzeiger, char *ausgabestrg,...)* Diese Funktion verhält sich wie die *printf*-Funktion. Nur wird nicht auf den Bildschirm geschrieben, sondern eben in die Datei auf die *fzeiger* zeigt.

  Rückgabewert: -1 im Fehlerfall, sonst liefert *fprintf* die Anzahl der geschriebenen Zeichen.

```
/*   kleines testprogramm fuer datei-io        */
/*   kompilieren mit: gcc col2dot.c -0 col2dot   */
/*   aufruf mit: col2dot lesefile schreibefile   */
#include <stdio.h>
#include <stdlib.h>
main( int argc, char *argv[] )
{
  FILE *ifile, *ofile;
  int zeichen, zaehler = 0 ;

  if ( argc<3 )
    {
      fprintf( stderr,"\n Aufruf: Kopiere ifile ofile !!\n\n " );
      return (-1);
    }

  ifile = fopen(argv[1], "r");
  if ( ifile == NULL )
    {
      fprintf( stderr,"\n ERROR: Fehler ifile !!\n\n " );
      return (-1);
    }

  ofile = fopen(argv[2], "w");
  if ( ofile == NULL )
    {
      fprintf( stderr,"\n ERROR: Fehler ofile !! " );
      return (-1);
    }
  printf("\n\n mit michas e r s t e m makefile compiliert!!");
  printf("\n\n jetzt steige ich in die neue leseroutine ein\n\n");

  while ( zeichen != EOF )
    {

      zeichen = fgetc( ifile );
      if ( zeichen == 0x2C ){
        zeichen = 0x2E;
        zaehler++;
      }

      if ( fputc(zeichen, ofile ) == -1 ){
      fprintf( stderr,"\n ERROR: Schreibfehler !! " );
      }



    }

  fclose( ifile );
  fclose( ofile );

  printf("\n\n%i kommas wurden ersetzt.\n\n", zaehler);

}
```

Abbildung 1: Ein C-Testprogramm