LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

MNM TEAM
MUNICH NETWORK MANAGEMENT TEAM

Einführung in die Programmierung mit C

Teil 1

Dieter Kranzlmüller
Nils gentschen Felde

Heute: Johannes Brechtmann

Literatur + Weitere Information

- C von A bis Z, Galileo Computing (Deutsch) und Linux-UNIX-Programmierung, Galileo Computing (Deutsch)
  - Verfügbar als openbook (openbook.galileocomputing.de/c_von_a_bis_z/)
    Verweise in den Folien auf das Buch gekennzeichnet mit:
    Kapitel xx
  - openbook.galileocomputing.de/linux_unix_programmierung/
    Verweise in den Folien auf das Buch gekennzeichnet mit:
    Kapitel xx
- The C Programming Language, Prentice Hall (Englisch)
- Online
  - C Reference Cards, Tutorials
  - http://en.wikibooks.org/wiki/C_Programming

Themenübersicht Block 1

- Hello World – Das erste Programm
- Vom Quellcode zum Programm
- Variablen & Datentypen
- Operatoren
- Kontrollstrukturen
- Funktionen
- Arrays
- Zeiger, Dynamischer Speicher
- Parameterübergabe, Kommandozeilenargumente
- Formatierte Ein-, Ausgabe, Steuerzeichen
- Datei Ein-, Ausgabe
- Makefiles

C Überblick

- C wurde von Dennis Ritchie[1] entwickelt und 1972 veröffentlicht
- Um eine Fragmentierung zu vermeiden und Portabilität zu gewährleisten wurde der ANSI C Standard ins Leben gerufen
  - C99 (2000)
  - C11 (veröffentlich 8. Dez. 2011)
- Für viele Plattformen verfügbar

[1] heise Artikel 8.10.2012
http://www.heise.de/newsticker/meldung/Hello-World-Zum-ersten-Todestag-von-Dennis-Ritchie-1724988.html

Vergleich zu anderen Sprachen

C vs. C++

- C++ begann als Weiterentwicklung von C
- C++ bietet objektorientierte Programmierung (Klassen)
- C Programme lassen sich in der Regel mit einem C++ Compiler übersetzen (nicht umgekehrt)

C(++) vs. Java

- C ist „systemnäher“
- Mehr Freiheiten, aber auch mehr Verantwortung beim Programmierer → mehr Risiken
- Programme werden direkt in Maschinencode übersetzt (keine VM, kein Bytecode)
- Programmierer muss sich um Speicherverwaltung kümmern → keine Garbage Collection

Erstes Programm: "Hello World" (hello.c)

```
#include <stdio.h>
int main() {
    printf("Hello World!\n");
    return 0;
}
```

Einbinden der Bibliothek „stdio“

Funktion main() mit Ganzzahl als Rückgabewert

„{“ und „}“ umgeben einen Anweisungsblock

Ausgabe

Rückgabe des Wertes 0

⇨ Kapitel 12

Hello World

- Erstes Programm: "Hello World" (hello.c)

```
#include <stdio.h>
int main() {
     printf("Hello World!\n");
    return 0;
}
```

- Übersetzung

  gcc hello.c -o helloworld

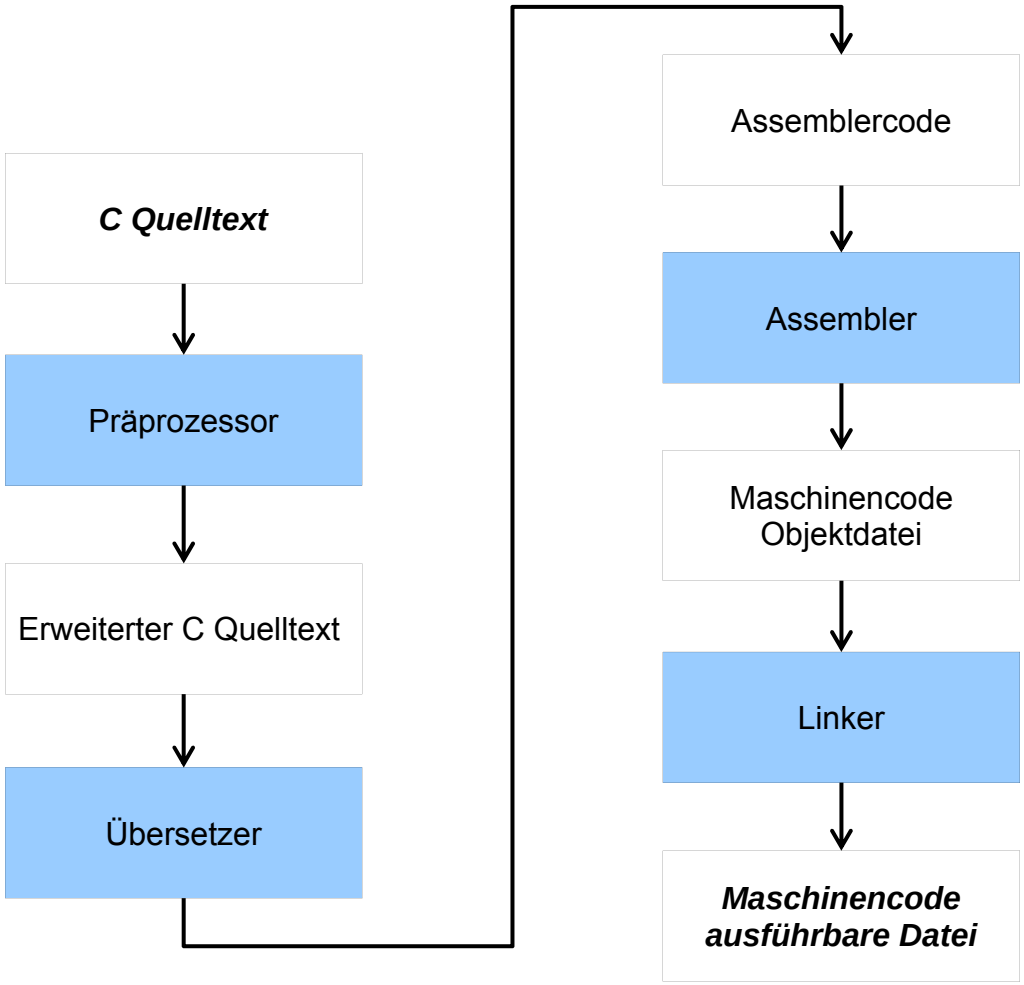
- Ausführung

  ./helloworld

- Ausgabe

  Hello World!

Vom Quelltext zum Programm



Dienstprogramm

Sprachebene

Vom Quelltext zum Programm
Dienstprogramme

- **Präprozessor:** Führt Änderungen am Quelltext durch, bevor er an den Übersetzter weitergeleitet wird.
  - Bedingtes Entfernen, Ersetzten und Einfügen von Quelltext
  - Präprozessor-Direktiven
    - #include
    - #define
    - #ifdef

⇨ Kapitel 10

- **Übersetzer:** Wandelt den in einer Ausgangssprache bereit gestellten Quelltext in eine Zielsprache um
- **Assembler:** Wandelt in Assemblersprache vorliegenden Code in Maschinensprache um
- **Linker:** Verbindet Module (Objektdateien) und Bibliotheken zu einem ausführbaren Programm

GNU Compiler Collection

- Unterstützt eine Vielzahl an Programmiersprachen und Prozessorarchitekturen
- Aufruf: `gcc`
  - Beinhaltet implizite Aufrufe von
    - Präprozessor
    - Übersetzer
    - Assembler
    - Linker
  - Typische Parameter
    - Ausgabe Datei spezifizieren `-o file`
    - Alle Warnungen aktivieren `-Wall`
    - Debugging Information inkludieren `-g`
    - C99 Standard `-std=c99`
  - Nicht alle Phasen müssen ausgeführt werden
    - Nur Objektdateien erzeugen `gcc -c`

Variablen

- Speicherbereich (im RAM) zum Ablegen eines Wertes
- Definition
  - `int a;`
- Initialisierung
  - `a=15;  // Dec`
  - `a=017; // Oct`
  - `a=0xf; // Hex`

⇨ Kapitel 5

auto	break	case	char
complex	const	continue	default
do	double	else	enum
extern	float	for	goto
if	imaginary	inline	int
long	register	restrict	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

Reservierte Schlüsselwörter

- Variablenname nicht aus reservierten Schlüsselwörtern
- Variablen enthalten, wenn sie nicht initialisiert werden, einen unbestimmten Wert (anders als Java!)

Datentypen (Auswahl)

Typ	Bytes	Wertebereich
`char`	1	-128 .. 127
`unsigned char`	1	0 .. 255
`int`	2	-32 768 .. 32 767
`short`	2	-32 768 .. 32 767
`long`	4	-2 147 483 648 .. 2 147 483 647
`unsigned int`	2	0 .. 65 535
`unsigned short`	2	0 .. 65 535
`unsigned long`	4	0 .. 4 294 967 295
`float`	4	1.17E-38 .. 3.4E38
`double`	8	2.2E-308 .. 1.8E308

Angaben für 32-bit (x86) Architektur

Variablen und Sichtbarkeit

Globale Variablen

- Sind für alle Funktionen sichtbar

Lokale Variablen

- Sind nur im jeweiligen Anweisungsblock und darunter sichtbar.
- Können in darunter liegenden Blöcken neu deklariert werden.

Kommentare

// Kommentar

- Nur über eine Zeile

/* Kommentar */

- Beginnt mit /*
  - Kommentar kann mehrere Zeilen lang sein
- Endet mit */

Operatoren

Arithmetisch		Logisch		Vergleichend	
+	Addition	&&	Logisches UND	<	Kleiner
-	Subtraktion oder Vorzeichen	||	Logisches ODER	>	Größer
*	Multiplikation	!	Logisches NICHT	==	Gleich
/	Division	&	Bitweiser UND Vergleich	<=	Kleiner gleich
%	Modulo	|	Bitweiser ODER Vergleich	>=	Größer gleich
++	Inkrementierung um 1	>>	Bitweiser Rechts-Shift	!=	Ungleich
--	Dekrementierung um 1	<<	Bitweiser Links-Shift		
		~	Bitweises NICHT		

`x=x+y;` entspricht `x+=y;`

⇝ Kapitel 6

Kontrollstrukturen

- Bedingungen (if)
- Fallunterscheidung (switch)
- Schleifen (for, while, do-while)

⇛ Kapitel 8

If / Else

Muster

```
if(Bedingung){
   /* Quelltext */
}else{
   /* Quelltext */
}
```

Beispiel

```
if ( x > y ){
   printf("x > y - %d\n",x);
}else{
   printf("y <= x - %d\n",y);
}
```

Verknüpfung von Bedingungen

- Und (Bedingung 1) && (Bedingung 2)
- Oder (Bedingung 1) || (Bedingung 2)

Fallunterscheidung

Muster

```
switch(Ausdruck){
   case Wert1 : /* Quelltext */
      break;
   case Wert2 : /* Quelltext */
      break;
   default:  /* Quelltext */
      break;
}
```

Beispiel

```
switch(x){
   case 1 : n=1;
      break;
   case 2 : n=2;
      break;
   default: n=0;
      break;
}
```

Schleifen

for

```
for(Init., Bed., Mod.){
   // Quelltext

}
```

```
for(int i=0;i<10;i++){
   printf(„%i“,i);

}
```

while

```
while(Bedingung){
   // Quelltext


}
```

```
while(i<10){
   printf(„%i“,i);
   i++;

}
```

do

```
do{
   // Quelltext


}while(Bedingung)
```

```
do{
   printf(„%i“,i);
   i++;

}while(i<10)
```

Funktionen

- Deklaration

```
Rückgabetyp Funktionsname(
    [Parametertyp Parametername], …);
```

- Definition

```
Rückgabetyp Funktionsname(
    [Parametertyp Parametername], …)
{
        /* Quelltext */
}
```

z.B.

```
int addiere(int x,int y){
   return x+y;
}

void ausgabe(int x){
   printf(„%i“,x);
}
```

- Aufruf

```
Rückgabewert = Funktionsname([Parameter], …);
```

- Soll eine Funktion keinen Rückgabewert besitzen

  → Rückgabetyp void

Arrays

- Arrays (Felder)
  - Statische, geordnete Folge von Werten eines Typs
  - Festgelegte Anzahl - Größe kann nicht nachträglich verändert werden
  - Numerierung beginnt mit 0

- Deklaration
  `Datentyp Name[gewünschte Anzahl der Elemente];`
  auch mehrdimensionale Arrays möglich z.B. `Datentyp Name[n][m];`
- Beispiel
  - `int i[5];` bietet Platz für 5 Integer Werte i[0], i[1], i[2], i[3], i[4]
    `i[0]=3;` erstem Element des Arrays wird 3 zugewiesen
    `i[4]=4;` fünftem und letztem Element des Arrays wird 4 zugewiesen

⇨ Kapitel 11

Zeiger und Adressen

- Der Inhalt einer Variable ist im Speicher abgelegt
- Dieser Ort hat eine Speicheradresse
- Diese Adresse kann wiederum in einer Variablen gespeichert werden – genannt Pointer (Zeiger)
- Ein Pointer zeigt somit auf eine andere Variable
- Der Wertebereich für einen Pointer hängt von der Architektur ab

Tutorial: boredzo.org/pointers/

⇨ Kapitel 12

Zeiger- und Adressoperatoren

- Deklaration
  - int *p;
- Addressoperator
  - &variable liefert einen Zeiger auf die Variable (Wert = deren Speicheradresse)
- Dereferenzierungsoperator
  - *pointer liefert den Inhalt einer Variable auf die der Pointer zeigt (Wert = Inhalt)
- Zuweisungen
  - pointer=&variable weist die Speicheradresse einer Variable zu
  - *pointer=32 weist den Wert der Variable zu, auf die der Pointer zeigt
  - *pointer=variable übernimmt den Wert der Variable für die Variable, auf die der Pointer zeigt
- Pointer können
  - auch auf Felder und Funktionen zeigen
  - inkrementiert und dekrementiert werden (pointer++, pointer--)

Dynamischer Speicher

- Anfordern
  - `void *malloc(size_t size);`
    - Gibt NULL zurück falls Speicher nicht verfügbar
  - Beispiel: (Speicherplatz für 2 int Werte reservieren)

```
int *p;
p=malloc(2*sizeof(int));
```

- Freigeben
  - `void free(void *pointer);`
  - Beispiel: (Speicherplatz freigeben)

```
free(p);
```

⇝ Kapitel 12

Parameterübergabe - Kommandozeilenargumente


```
int main(int argc, char *argv[]) {
   /* Quelltext */
}
```

- argc
  - Die Anzahl der Parameter
  - Mindestens 1 Parameter – der erste Parameter ist der Programmname
- argv
  - Ein Array von Zeichenketten, Zugriff auf Elemente mit [..]

⇛ Kapitel 13

Parameterübergabe - Beispiel


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

int main(int argc, char *argv[]) {

  printf("Es wurden %i Parameter übergeben.\n", argc-1);

  printf("Dieses Programm heißt: %s\n", argv[0]);

  if(argc >= 2) printf("Der erste Parameter ist: %s\n", argv[1]);

  if(argc >= 3) printf("Der zweite Parameter ist: %s\n", argv[2]);

  if(argc >= 4) printf("Der dritte Parameter ist: %s\n", argv[3]);

  return 0;
}
```

Formatierte Ein-/Ausgabe

- Eingabe
  - `scanf("%i", &variable);`
- Ausgabe
  - `printf("Hello World %i", variable);`
- Formatanweisungen
  - %s Zeichenkette (String
  - %c Einzelnes Zeichen (Character)
  - %i Ganze Zahl (Integer)
  - %f Gleitkommazahl (Float)
  - %o Oktalwert
  - %x Hexadezimalwert
  - %p Pointer (Zeiger/Adresse)

⇨ Kapitel 4

Formatierte Ein-/Ausgabe


```
#include <stdio.h>
int main() {
    char z = 'K';
    char s[] = "Beliebiger Text\n";

    printf(s);                          Beliebiger Text
    printf("%c\n", z);                  K
    printf("%i\n", z);                  75
    printf("%f\n", z);                  0.000000
    printf("%x\n", z);                  4b
    printf("%o\n", z);                  113


    scanf("%c", &z);                    X
    printf("%c\n", z);                  X

    scanf("%s", &s);                    abcdef
    printf("%s\n", s);                  abcdef

    scanf("%x", &z);                    a
    printf("%i\n", z);                  10
    return 0;
}
```

Steuerzeichen

- Repräsentieren nicht druckbare bzw. belegte Zeichen
- Beginnen mit Backslash

Beispiele für wichtige Steuerzeichen

- `\n` *Neue Zeile*
- `\r` *Zeilenanfang*
- `\t` *Tabulator*
- `\“` "
- `\‘` ‘
- `\?` ?
- `\\` \
- `\0` *Ende einer Zeichenkette*

Datei Ein- und Ausgabe

- OS verwaltet pro Prozess Tabelle für dessen geöffnete Dateien
  - I-Node-Nummer (identifiziert den physischen Speicherort aller Dateimeta- und -adressinformationen)
  - Position innerhalb der Datei
  - Modus (read, write, attach)

- Filedeskriptor
  - Index (integer) des jeweiligen Tabelleneintrags
  - Für die Ein-/Ausgabe am Terminal gibt es drei Standard Filedeskriptoren (Konstanten in **unistd.h**)
    - Eingabe stdin (STDIN_FILENO)
    - Ausgabe stdout (STDOUT_FILENO)
    - Fehlerausgabe stderr (STDERR_FILENO)

⇨ Kapitel 16

⇨ Kapitel 16.26

Ein- und Ausgabe Funktionen

- Datei öffnen
  - open(...) oder fopen(...)
- Filedeskriptor (und damit Datei) schließen
  - close(...)
- In eine geöffnete Datei schreiben
  - write(...)
- Aus einer geöffneten Datei lesen
  - read(...)

Kapitel 16.26

Kapitel 16.5

- Konstante OPEN_MAX in limits.h:
  - Maximale Anzahl geöffneter Filedeskriptoren für einen Prozess
  - Entspricht der minimalen Anzahl an Filedeskriptoren, die das Betriebssystem immer zur Verfügung stellen kann

Ein- und Ausgabe Funktionen

- Optionen der Funktion open()
  - Bearbeitungsflags beim Öffnen einer Datei (schließen sich gegenseitig aus!):
    - `O_RDONLY`: nur Lesen
    - `O_WRONLY`: nur Schreiben
    - `O_RDWR`: Lesen und Schreiben
  - Zusatzflags (können mit Bearbeitungsflags kombiniert werden):
    - `O_CREAT`: Datei erstellen, falls nicht vorhanden (Zugriffsrechte als dritter Parameter)
    - `O_APPEND`: Aktuelle Position auf Dateiende setzen (Anhängen)
    - `O_EXCL` (in Kombination mit O_CREAT): Datei wird nur geöffnet, falls sie zuvor noch nicht existiert hat
    - Weitere: siehe man 2 open
- Beispiel:

```
open(“test.txt“, O_WRONLY | O_EXCL | O_CREAT, 0644);
```

Beispiel: Datei Ein- und Ausgabe


```
#include <sys/types.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
  const char *new_file;
  int fd; // Filedeskriptor

  if(argc != 2) {
      fprintf(stderr, "Usage: file to open\n");
      return EXIT_FAILURE;
  }

  new_file = argv[1];
  fd = open(new_file, O_WRONLY | O_EXCL | O_CREAT, 0644);
  printf("fd value is: %i\n", fd);
  if(fd == -1) {
      perror("Error opening file");
      return EXIT_FAILURE;
  }
  return EXIT_SUCCESS;
}
```

Makefiles

- Automatisierung des Übersetzens und Linkens eines Programms
  - Erlaubt im Prinzip das Ausführen beliebiger Aktionen
  - Dabei werden nur die tatsächlich erforderlichen Übersetzungen (entsprechend der definierten Abhängigkeiten) vorgenommen.

- Befehl
  - make Ziel

- Konfiguration via Makefile
  - Ziele
  - Abhängikeiten
  - Regeln

Makefile

File Edit View Search Tools Documents Help

Open Save Undo

Makefile

```
CFLAGS = -g -Wall

CC = gcc

all: myprog1 myprog2

myprog1: p1_modul1.c p1_modul2.c
    $(CC) $(CFLAGS) -o myprog1 p1_modul1.c p1_modul2.c
myprog2: p2_main.c
    $(CC) $(CFLAGS) -o myprog2 p2_main.c
clean:
    rm –f *.o myprog1 myprog2
```

Makefile Tab Width: 8 Ln 1, Col 1 INS

Übungen: Blatt 1

- Aufgabe 1.1: temperaturUmrechner.c
- Temperaturangaben zwischen verschiedenen Einheiten konvertieren
- Quell- und Zieleinheit:
  - Grad Celsius
  - Grad Delisle
  - Grad Fahrenheit
  - Kelvin
  - Grad Rankine
- Quell-, Zieleinheit und der zu konvertierende Temperaturwert als Kommandozeilenparameter

Übungen: Blatt 1

- Aufgabe 1.2: matrixMult.c
- Gegeben: zwei n × n-Matrizen einer beliebigen, aber festen (apriori bekannten) Größe gefüllt mit ganzen Zahlen zwischen 0 und 9
- Gesucht: Produkt der beiden Matrizen
- Ausgabe des Ergebnisses auf der Standardausgabe

Übungen: Blatt 1

- Aufgabe 1.3: findMaxOfFloats.c
- Gegeben: eine Datei, mit beliebig viele Zahlen des Typs float (je Zeile genau eine Zahl)
- Gesucht: Maximum dieser Zahlen
- Einlesen der Zahlen in ein Array
- Übergabe des Dateinamens entweder als Kommandozeilenparameter oder Abfrage im Programm (beide Optionen)
- Ausgabe des Maximums auf der Standardausgabe

Übungen: Blatt 1

- Aufgabe 1.4: gameOfLife.c
- Regeln des game of life
  - Eine tote Zelle mit genau drei lebenden Nachbarn wird in der Folgegeneration neu geboren.
  - Lebende Zellen mit weniger als zwei lebenden Nachbarn sterben in der Folgegeneration an Einsamkeit.
  - Eine lebende Zelle mit zwei oder drei lebenden Nachbarn bleibt in der Folgegeneration lebend.
  - Lebende Zellen mit mehr als drei lebenden Nachbarn sterben in der Folgegeneration an Uberbevölkerung.
  - Hinweis: alle anderen Zellen leben nicht
- Größe des Spielfeldes: 16 × 16 Felder, optional per Kommandozeilenparameter alternative Größe