Einführung

Praktikum „C-Programmierung“

Eugen Betke, Nathanael Hübbe,
Michael Kuhn, Jakob Lüttgau, Jannek Squar

Wissenschaftliches Rechnen
Fachbereich Informatik
Universität Hamburg

2018-10-22

Übersicht

- Jeder Termin besteht aus zwei Teilen:
  - Präsentation von 30–45 Minuten Länge
    - Interaktiv gehalten, Zwischenfragen erwünscht
  - Besprechung der Übungsaufgaben
    - Keine Abgabe, Besprechung im Praktikum
- Präsentationen und Übungsblätter werden auf der Webseite veröffentlicht
- Maximal zweimal unentschuldigt fehlen

Übungen

- Ein Übungsblatt pro Woche
  - Viel Programmieren und Rückmeldung
  - Bearbeitung in Gruppen zu je 2–3 Personen
- Mindestens zweimal präsentieren!
  - Potentielle Vorrechner werden ggf. vorher per E-Mail informiert
  - Jedes Gruppenmitglied muss in der Lage sein zu präsentieren
  - Wer nicht präsentieren kann/will, bekommt Punktabzug

Themen

22.10. Einführung
29.10. Syntax und Kontrollstrukturen
05.11. Debugging
12.11. Arrays und Datenstrukturen
19.11. Zeiger, Zeigerarithmetik
26.11. Zeigerarithmetik, Funktionszeiger
03.12. Speicher
10.12. Valgrind
17.12. Undefiniertes Verhalten
07.01. Kompilieren, Linken, Präprozessor
14.01. Bibliotheken, Header, Modularität
21.01. Objektorientierung
28.01. Modernes C

Tutorials und Bücher

- Liste empfohlener Bücher und Tutorials zu C:
  `http://www.iso-9899.info/wiki/Books`
- The C Book:
  `http://publications.gbdirect.co.uk/c_book/`
  `http://publications.gbdirect.co.uk/c_book/thecbook.pdf`

Übersicht

- Ziele des ersten Termins:
  - Allgemeine Informationen zu C
    - Geschichte
    - Entwicklung
    - Einsatzgebiete
  - Kennenlernen der Grundlagen
    - Programmaufbau
    - Kompilieren
    - Makefiles

Geschichte [1]

- Wurde in den 1970ern von Dennis Ritchie entwickelt
  - Seitdem stetige Weiterentwicklung
- Funktionsumfang wird durch Standards festgelegt
  - Bis 1989: Buch “The C Programming Language” von Brian W. Kernighan und Dennis Ritchie als Quasi-Standard (K&R C)
  - Ab 1989: Standardisierung durch ANSI (ANSI C, C89)
  - Ab 1990: Internationale Norm durch ISO (C90, entspricht C89)
  - Ab 1995: Erste Erweiterung (C95)

Geschichte… [1]

- Ab 1999: Neuer Standard (C99)
  - Größtenteils mit C90 kompatibel
  - Neue Funktionen, teilweise von C++ übernommen
    - Echte Booleans, komplexe Zahlen etc.
- Ab 2011: Neuer Standard C11
  - Neue Funktionen und bessere Kompatibilität mit C++
    - Atomare Datentypen, Threads etc.
- Ab 2018: Neuer Standard C18 (entspricht C11 plus Fehlerkorrekturen)

Einsatzgebiete [1]

- C wird häufig in der Systemprogrammierung genutzt
  - Betriebssysteme und eingebettete Systeme
  - Sehr portabel und effizient
  - Hardware kann direkt angesprochen werden
  - Keine Laufzeitumgebung
  - Bekanntestes Beispiel ist vermutlich Linux
- Auch Anwendungssoftware in C
  - Insbesondere Programmierschnittstellen und andere APIs
  - Anbindung an viele andere Sprachen gegeben
- Häufig auch Compiler und Bibliotheken für andere Programmiersprachen in C
- Manchmal auch als Zwischensprache genutzt
  - Portabilität und Bequemlichkeit
  - Beispiel: Vala

Vorbereitung

- Installation unter CentOS, Fedora etc.

```
$ dnf install gcc
```

- Installation unter Debian, Ubuntu etc.

```
$ apt install gcc
```

- Installation unter Windows
  - Z. B. via Windows Subsystem for Linux
  - Alternativ virtuelle Maschine mit Linux

Vorbereitung…

- Kommandozeile
  - Emacs, nano, Vim
- Grafische Editoren
  - gedit
- Integrierte Entwicklungsumgebungen
  - Atom, Eclipse, GNOME Builder

Hello World v0

```
1 int main (void)
2 {
3     return 0;
4 }
```

Hello World v0

```
1 int main (void)
2 {
3     return 0;
4 }
```

- Die Funktion `main` bildet den Einstiegspunkt
  - Die Funktion gibt einen Integer (`int`) zurück
  - Die Funktion erwartet keine Argumente (`void`)

Hello World v0

```
1 int main (void)
2 {
3     return 0;
4 }
```

- Die Funktion `main` bildet den Einstiegspunkt
  - Die Funktion gibt einen Integer (`int`) zurück
  - Die Funktion erwartet keine Argumente (`void`)
- Die Funktion gibt den Wert `0` zurück
  - `0` steht für Erfolg, alle anderen Werte für einen Fehler

Kompilieren

```
1 $ gcc hello.c
2 $ ./a.out
```

Kompilieren

```
1 $ gcc hello.c
2 $ ./a.out
```

- GCC produziert standardmäßig ein Programm namens `a.out`
  - Kann mit `-o` geändert werden

Kompilieren

```
1 $ gcc hello.c
2 $ ./a.out
```

- GCC produziert standardmäßig ein Programm namens `a.out`
  - Kann mit `-o` geändert werden
- Ausführung erfordert Pfadangabe
  - `./` steht dabei für das aktuelle Verzeichnis

Kompilieren

```
1 $ gcc -o hello hello.c
2 $ ./hello
```

Kompilieren

```
1 $ gcc -o hello hello.c
2 $ ./hello
```

- Mit -o wird die Ausgabedatei festgelegt
  - Programm ist dann als `hello` aufrufbar

Kompilieren

```
1 $ gcc -c hello.c
2 $ gcc hello.o
3 $ ./a.out
```

Kompilieren

```
1 $ gcc -c hello.c
2 $ gcc hello.o
3 $ ./a.out
```

- Bisher direktes Kompilieren des Programms aus dem Quelltext
  - Jetzt wird zuerst eine Objektdatei `hello.o` erzeugt

Kompilieren

```
1 $ gcc -c hello.c
2 $ gcc hello.o
3 $ ./a.out
```

- Bisher direktes Kompilieren des Programms aus dem Quelltext
  - Jetzt wird zuerst eine Objektdatei `hello.o` erzeugt
- Eine Objektdatei enthält Informationen einer Übersetzungseinheit
  - Eine Übersetzungseinheit besteht üblicherweise aus einer Quelltext-Datei
  - Eine oder mehrere Objektdateien werden zu einem Programm gelinkt

Makefile

```
1 all: hello
2
3 clean:
4     rm -f hello
```

- Makefiles erleichtern das Kompilieren von Code
  - Makefiles bestehen aus Regeln und Anweisungen

Makefile

```
1 all: hello
2
3 clean:
4     rm -f hello
```

- Makefiles erleichtern das Kompilieren von Code
  - Makefiles bestehen aus Regeln und Anweisungen
- Regeln haben Abhängigkeiten
  - Die Regel `all` hängt von `hello` ab
  - `hello` wird durch eine interne Regel kompiliert

Makefile

```
1 all: hello
2
3 clean:
4     rm -f hello
```

- Makefiles erleichtern das Kompilieren von Code
  - Makefiles bestehen aus Regeln und Anweisungen
- Regeln haben Abhängigkeiten
  - Die Regel `all` hängt von `hello` ab
  - `hello` wird durch eine interne Regel kompiliert
- Regeln können Anweisungen enthalten
  - Die Regel `clean` enthält eine Anweisung mit `rm`

Hello World v1

```
1  #include <stdio.h>
2
3  int main (void)
4  {
5      printf("Hello world!\n");
6      return 0;
7  }
```

Hello World v1

```
1  #include <stdio.h>
2
3  int main (void)
4  {
5      printf("Hello world!\n");
6      return 0;
7  }
```

- Zusätzliche Funktionen werden über Header bekannt gemacht
  - Die Funktion `printf` wird in `stdio.h` deklariert
  - Die Definition befindet sich in der libc (üblicherweise glibc)

Hello World v2

```
1  #include <stdio.h>
2
3  int main (void)
4  {
5      int const foo = 42;
6      printf("Hello world %d!\n", foo);
7      return 0;
8  }
```

Hello World v2

```
1  #include <stdio.h>
2
3  int main (void)
4  {
5      int const foo = 42;
6      printf("Hello world %d!\n", foo);
7      return 0;
8  }
```

- Zusätzliche Variablen können überall deklariert werden
  - Vor C99 mussten Variablen am Anfang eines Blockes deklariert werden

Hello World v2

```
1  #include <stdio.h>
2
3  int main (void)
4  {
5      int const foo = 42;
6      printf("Hello world %d!\n", foo);
7      return 0;
8  }
```

- Zusätzliche Variablen können überall deklariert werden
  - Vor C99 mussten Variablen am Anfang eines Blockes deklariert werden
- Variablen können als konstant (`const`) markiert werden
  - Dann ist keine weitere Zuweisung nach der Definition möglich

Hello World v2

```
1  #include <stdio.h>
2
3  int main (void)
4  {
5      int const foo = 42;
6      printf("Hello world %d!\n", foo);
7      return 0;
8  }
```

- Zusätzliche Variablen können überall deklariert werden
  - Vor C99 mussten Variablen am Anfang eines Blockes deklariert werden
- Variablen können als konstant (`const`) markiert werden
  - Dann ist keine weitere Zuweisung nach der Definition möglich
- Die Funktion `printf` akzeptiert Umwandlungsanweisungen für Variablen
  - Die Anweisung %d steht für Integer in vorzeichenbehafteter Dezimalnotation

Datentypen

`char`	Einzelne Zeichen (1 Byte)
`int`	Integer (üblicherweise 4 Bytes)
`float`	Gleitkommazahl (üblicherweise 4 Bytes)
`double`	Gleitkommazahl (üblicherweise 8 Bytes)
`void`	Unvollständiger Datentyp
`enum`	Aufzählungen (intern Integer)
`struct`	Strukturen
`[]`	Arrays
`*`	Zeiger

Hello World v3

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main (void)
5 {
6     char bar[] = "world";
7     printf("Hello %s (%lu)!\n", bar, strlen(bar));
8     return 0;
9 }
```

Hello World v3

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main (void)
5  {
6      char bar[] = "world";
7      printf("Hello %s (%lu)!\n", bar, strlen(bar));
8      return 0;
9  }
```

- C kennt keinen nativen String-Datentyp
  - Strings sind `char`-Arrays
  - Einzelne Zeichen mit `'`, Strings mit `"`

Hello World v3

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main (void)
5  {
6      char bar[] = "world";
7      printf("Hello %s (%lu)!\n", bar, strlen(bar));
8      return 0;
9  }
```

- C kennt keinen nativen String-Datentyp
  - Strings sind `char`-Arrays
  - Einzelne Zeichen mit `'`, Strings mit `"`
- Strings werden mit einem Null-Byte terminiert (`\0`)
  - `strlen` zählt die Anzahl der Zeichen

Hello World v4

```
1  #include <stdio.h>
2
3  void hello (void)
4  {
5      printf("Hello world!\n");
6  }
7
8  int main (void)
9  {
10     hello();
11     return 0;
12 }
```

Hello World v4

```
1  #include <stdio.h>
2
3  void hello (void)
4  {
5      printf("Hello world!\n");
6  }
7
8  int main (void)
9  {
10     hello();
11     return 0;
12 }
```

- Es können beliebige Funktionen definiert werden
  - Die Funktion `hello` hat keinen Rückgabewert und keine Argumente

Hello World v5

```
1  #include <stdio.h>
2
3  #define HELLO_WORLD "Hello world!"
4  #define HELLO(x) "Hello " x "!"
5
6  int main (void)
7  {
8      printf(HELLO_WORLD "\n");
9      printf(HELLO("world") "\n");
10     return 0;
11 }
```

Hello World v5

```
1  #include <stdio.h>
2
3  #define HELLO_WORLD "Hello world!"
4  #define HELLO(x) "Hello " x "!"
5
6  int main (void)
7  {
8      printf(HELLO_WORLD "\n");
9      printf(HELLO("world") "\n");
10     return 0;
11 }
```

- Mit #define können Makros definiert werden
  - Sowohl simple Textersetzung als auch funktionsähnliche Makros

Zusammenfassung

- C ist eine wichtige Programmiersprache
  - Insbesondere verbreitet in der Systemprogrammierung
- Es können unterschiedliche Compiler und Editoren genutzt werden
- Makefiles vereinfachen das Kompilieren von Programmen

- Aufgaben für nächste Woche:
  1. Einrichten eines Compilers
  2. Einrichten der Entwicklungsumgebung
  3. Nachvollziehen der Hello-World-Beispiele

Quellen I

[1] Wikipedia. C (Programmiersprache).
https://de.wikipedia.org/wiki/C_(Programmiersprache).