

Einführung

Praktikum „C-Programmierung“



Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Eugen Betke, Nathanael Hübbe, Michael Kuhn, Jannek Squar

2019-10-21

Wissenschaftliches Rechnen
Fachbereich Informatik
Universität Hamburg

Organisatorisches

Einführung

Übersicht

Einrichtung

Beispiele

Zusammenfassung

Quellen

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

- Ein Übungsblatt pro Woche
 - Viel Programmieren und Rückmeldung
 - Bearbeitung in Gruppen zu je 3 Personen
 - Eine Aufgabe muss abgegeben werden und wird bewertet
- Mindestens einmal präsentieren!
 - Vorrechnende werden zufällig ausgewählt
 - Jedes Gruppenmitglied muss in der Lage sein zu präsentieren
 - Wer nicht präsentieren kann/will, bekommt Punktabzug

- 21.10. Einführung
- 28.10. Syntax und Kontrollstrukturen
- 04.11. Arrays und Datenstrukturen
- 11.11. Speicher (Stack und Heap)
- 18.11. Zeiger, Zeigerarithmetik
- 25.11. Debugging und Valgrind
- 02.12. Zeigerarithmetik, Funktionszeiger
- 09.12. undefiniertes Verhalten
- 16.12. Kompilieren, Linken, Präprozessor
- 06.01. Bibliotheken, Header, Modularität
- 13.01. Linux-Konzepte
- 20.01. Ein-/Ausgabe
- 27.01. Modernes C

- 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

Organisatorisches

Einführung

Übersicht

Einrichtung

Beispiele

Zusammenfassung

Quellen

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

- 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)

- 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)

- 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

- Installation unter CentOS, Fedora etc.

```
1 $ dnf install gcc
```

- Installation unter Debian, Ubuntu etc.

```
1 $ apt install gcc
```

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

- 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 }
```

```
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`)

```
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


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

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

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

```
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

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

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

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

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

```
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

```
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


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

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

```
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

```
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 }
```

```
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

`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, %lu)!\n", bar, strlen(bar), sizeof(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, %lu)!\n", bar, strlen(bar), sizeof(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, %lu)!\n", bar, strlen(bar), sizeof(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` gibt Anzahl der Zeichen zurück, `sizeof` die Größe der Datenstruktur

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

- 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

Organisatorisches

Einführung

Übersicht

Einrichtung

Beispiele

Zusammenfassung

Quellen

[1] Wikipedia. **C (Programmiersprache)**.

[https://de.wikipedia.org/wiki/C_\(Programmiersprache\)](https://de.wikipedia.org/wiki/C_(Programmiersprache)).