Praktikum C-Programmierung 2026

Einführung

Jannek Squar

2026-04-08

Scientific Computing
Universität Hamburg

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Outline

Table of Contents

Historie C

- Erste Version in den 1970ern (Dennis Ritchie)
- Nachfolge von B
- Versionen (u.a.):
  - 1978: K&R C (Kernighan & Ritchie)
  - 1989: ANSI C (C89)
  - 1999: C99
  - 2011: C11
  - 2018: C17
  - 2024: C23
- `-std=x` Flag setzt Standard
- `gcc` 15.2.1 nutzt `-std=gnu23`
  - Älterer Compiler für Aufgaben ausreichend

Warum überhaupt C?

- Systemnahe Programmierung
  - Performance
  - Speichermanagement
  - Kernel vs. User Space
- Alternativen: C++, Rust

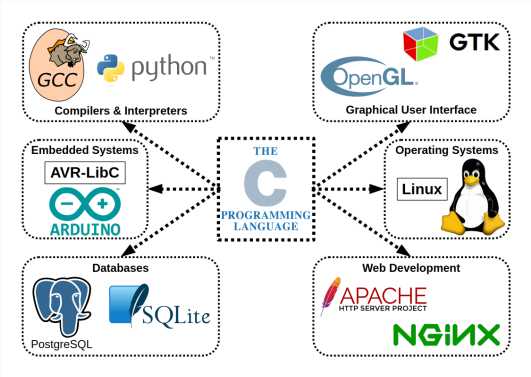

Abbildung 1: The C Programming Language[a]

[a] https://de.wikipedia.org/wiki/Datei:The_C_Programming_Language.png

Einrichtung Compiler

- Installation unter CentOS, Fedora etc. `$ dnf install gcc`
- Installation unter Debian, Ubuntu etc. `$ apt install gcc`
- Installation unter Windows[1]
  - Linux-Umgebung:
    - Windows Subsystem for Linux
    - Linux VM
  - native Compiler:
    - MinGW-w64 (GCC)
    - Microsoft Visual C (MSVC)[2]
    - Clang

[1] Kein Support für Windows von meiner Seite!
[2] Legt weniger Wert auf Standardkonformität

Tools

- Minimum: Texteditor + Compiler
  - vim, emacs, Notepad++, ...
- Advanced Editor:
  - VSCode
  - Sublime
  - (Atom)
  - ...
- IDE[3]:
  - Ecllipse
  - Visual Studio
  - Code::Blocks
  - ...

[3] Rate ich erstmal von ab

Style

- Keine Vorgabe, aber bleibt konsistent!
- Kommentare nutzen

```
1  // Einzeiliger Kommentar
2  /* Mehrzeiliger
3  Kommentar */
```

- Linter nutzen
  - Bsp: VSCode C/C++ Extension nutzt clang-format und `Visual Code` Style

```
1  UseTab: (VS Code current setting)
2  IndentWidth: (VS Code current setting)
3  BreakBeforeBraces: Allman
4  AllowShortIfStatementsOnASingleLine: false
5  IndentCaseLabels: false
6  ColumnLimit: 0
```

  - Style-Übersicht online

Style

- Style ist dem Compiler egal[4]
- Gleiches Programm:

```c
#include <stdio.h>
int main(void) { for(int i=0; i < 5; i++) printf("%d\n", i);return 0;}
```

```c
#include <stdio.h>

int main(void)
{
    for (int i = 0; i < 5; i++)
    {
        printf("%d\n", i);
    }
    return 0;
}
```

[4] Anders als z.B. bei Python

Table of Contents

Startup

- Aufruf der `main`-Funktion bei Programmstart
- Standard erlaubt zwei Varianten:

```c
int main(void) { /* ... */ }
```

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

Hello World v0

```c
int main (void)
{
  return 0;
}
```

- Einstiegspunkt Funktion `main`
  - Rückgabewert von Typ Integer (`int`)
  - Keine Übergabe von Argumenten (`void`)
- Funktion gibt den Wert `0` zurück
  - `0` steht für Erfolg, alle anderen Werte für einen Fehler
  - Verwechslungsgefahr mit `true` ($\neq 0$) bzw. `false` ($\equiv 0$)

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
  - Alternativ wird `$PATH` Umgebungsvariable durchsucht

Kompilieren

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

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

Kompilieren

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

- Expliziter Zwischenschritt: Erzeugung Objektdatei `hello.o`
  - Enthält Informationen einer Übersetzungseinheit
  - 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
```

- Buildsysteme wie Makefiles erleichtern Kompilieren
  - 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

```c
#include <stdio.h>

int main (void)
{
  printf("Hello world!\n");
  return 0;
}
```

- 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

```c
#include <stdio.h>

int main (void)
{
  int const foo = 42;
  printf("Hello world %d!\n", foo);
  return 0;
}
```

- Zusätzliche Variablen können überall deklariert werden
  - Vor C99 mussten Variablen am Anfang eines Blockes deklariert werden
- Variablen können als konstant (`const`) markiert werden
  - 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

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

int main (void)
{
  char bar[] = "world";
  printf("Hello %s (%zu, %zu)!\n", bar, strlen(bar), sizeof(bar));
  return 0;
}
```

- 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
  - Null-Byte benötigt ebenfalls Speicherplatz!

Hello World v4

```c
#include <stdio.h>

void hello (void)
{
  printf("Hello world!\n");
}

int main (void)
{
  hello();
  return 0;
}
```

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

Hello World v5

```c
#include <stdio.h>

#define HELLO_WORLD "Hello world!"
#define HELLO(x) "Hello " x "!"

int main (void)
{
  printf(HELLO_WORLD "\n");
  printf(HELLO("world") "\n");
  return 0;
}
```

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

Table of Contents

Zusammenfassung

- C ist eine wichtige Programmiersprache
  - Insbesondere verbreitet in der Systemprogrammierung
- Es können unterschiedliche Compiler und Editoren genutzt werden
- Buildsysteme vereinfachen das Kompilieren von Programmen
- Aufgaben für nächste Woche:
  1. Einrichten eines Compilers
  2. Einrichten der Entwicklungsumgebung
  3. Nachvollziehen der Hello-World-Beispiele