

# Grundlagen von C

Jonas Gresens

Proseminar C Grundlagen und Konzepte  
Arbeitsbereich Wissenschaftliches Rechnen  
Fachbereich Informatik  
Fakultät für Mathematik, Informatik und Naturwissenschaften  
Universität Hamburg

2014-04-24

# Gliederung (Agenda)

- 1 Einleitung
- 2 Entwicklung in C
- 3 Hello-World!
- 4 Konstrukte in C
- 5 Zusammenfassung
- 6 Literatur

# Einleitung

- Klassifikation
  - imperative Programmiersprache aus dem Jahr 1972
  - entwickelt von Dennis Ritchie an den Bell Laboratories
  - Beeinflusst von: BCPL (1966), ALGOL 68 (1968), B (1969)
  - Beeinflusste: C++ (1985), Java (1995), C# (2001), D (2007)
- Ziel
  - echte Sprachabstraktion zu Assembler  $\Rightarrow$  hardwarenah
  - geringe Abhängigkeit von einer Laufzeitumgebung

# Einleitung

- Eigenschaften
  - sehr kleiner Sprachkern  $\Rightarrow$  Portierbarkeit
    - „C kann ohne Bibliotheken fast gar nichts.“
  - direkte Speicherzugriffe  $\Rightarrow$  Systemprogrammierung
  - nicht typsicher
  - Modularisierung auf Dateiebene (Header-Dateien)
- Verwendung
  - Systemprogrammierung (Unix)
  - Programmierung von Mikrocontrollern
  - Compiler, Interpreter anderer Sprachen (JVM)

# Entwicklung in C

- IDEs: Visual Studio, Eclipse, NetBeans, Anjuta, Geany
- Texteditoren mit Syntax Highlighting: Sublime Text, Notepad++
- Compiler: gcc
  - `-o file` Schreibe Output in `file`, erzeugt ausführbare Datei
  - `-std=...` Festlegen des Sprachstandards  $\Rightarrow$  nutzbare Features
    - `c90` ISO C90
    - `c99` ISO C99
    - `c11` ISO C11  $\Leftarrow$  von uns verwendet

# Hello-World! - Code

```
1  /*
2   * C_Grundlagen_Hello_World!.c
3   */
4
5  #include <stdlib.h>
6  #include <stdio.h>
7
8  int main(int argc, char** argv)
9  {
10     //Print Hello World
11     printf("Hello World!");
12
13     return EXIT_SUCCESS;
14 }
```

# Hello-World! - Erklärung

- Einzeilige Kommentare: `//Kommentar`
- Mehrzeilige Kommentare: Zwischen `/*` und `*/`
- Einbinden von weiterem Code: `#include ...`
- `main`-Methode als Einstiegspunkt des Programms
  - Parameter zum Übergeben von Argumenten
  - Rückgabe an das Betriebssystem
- Blockklammern: `{ ... }`
- Methodenaufruf `identifizier(args ...)`
- Semikolon (`;`) am Ende jedes Statements
  - mehrere Statements durch Kommata (`,`) getrennt

# Basics: Variablen

- Typen: z.B. `int`, `double`, `char`
  - `bool` seit C99
  - `String` als `char[]`
- Deklaration: Vergabe eines Namens und eines Typs
  - `extern` - reine Deklaration der Variablen  $\Rightarrow$  Modularisierung
- Definition: Reservierung des Speicherplatzes
  - ist immer auch eine Deklaration
- Initialisierung: Zuweisung eines ersten Werts
- Beispiel:

```
1 int i;           //Deklaration und Definition
2 extern int j;   //nur Deklaration
3 i = 42;         //Initialisierung
4 char c = 'i';  //alles in einer Zeile
```



# Basics: Ausgabe

- benötigt `#include <stdio.h>`
- Konstante Ausgabe:
  - `printf("Hello World!");` ⇒ `Hello World!`
- Dynamische Ausgabe:
  - `printf("%d", i);` ⇒ `42`
- Kombinierte Ausgabe:
  - `printf("%c hat den Wert %d", c, i)` ⇒  
`i hat den Wert 42`
- conversion modifier:
  - z.B. `%d` oder `%i` für `int`, `%c` für `char`, `%f` für `double`
- Escapen von bestimmten Zeichen notwendig:
  - z.B. `\n`, `\t`, `\\`, `\"`, `\0`

## Fallunterscheidung: if-else (1)

- bedingte Ausführung (Syntax):

```
if (expr) statement;
```

- bedingte Ausführung mit Alternative (Syntax):

```
if (expr)
    statement;
else
    statement;
```

## Fallunterscheidung: if-else (2)

- geschachtelt (Syntax):

```
if (expr1)
    statement1;
else if (expr2)
    statement2;
else
    statement3;
```

```
if (expr1)
    if (expr2)
        statement1;
    else
        statement2;
```

- mehrere statements müssen in Blockklammern stehen
  - defensiv Klammern ist hilfreich

## Fallunterscheidung: ? (conditional)

- ternärer Operator (Kurzform von if-else)
- Syntax:  
bedingung ? wenn\_wahr : wenn\_falsch;
- kann einen Wert zurückliefern (funktionaler Ansatz):

```
1 int i = 5;  
2 printf("%d", (i >= 5) ? 200 : 300;
```

## Fallunterscheidung: switch-case

- Überprüfung mit konstanten Werten
- Syntax:

```
switch (var) {  
    case const1:  
        statement1;  
        break;  
    case const2:  
    case const3:  
        statement2;  
        break;  
    default:  
        statement3;  
        [break;]  
}
```

# for-Schleife

- bedingte wiederholte Ausführung, eingebaute Laufvariable

- Syntax:

```
for (vor_Beginn; Test_vor_Durchlauf; nach_Durchlauf) {  
    statement;  
}
```

- für alle Schleifen gilt: Blockklammern wie bei if

- Beispiel:

```
1 for (int i = 0; i < 10; ++i) {  
2     printf("%d ", i);  
3 }
```

Ausgabe:

```
1 0 1 2 3 4 5 6 7 8 9
```

# while-Schleife

- bedingte wiederholte Ausführung
- Syntax:

```
while (Test_vor_Durchlauf) {  
    statement;  
}
```

- Beispiel:

```
1 int i = 0;  
2 while (i < 10) {  
3     printf("%d ", i);  
4     ++i;  
5 }
```

Ausgabe:

```
1 0 1 2 3 4 5 6 7 8 9
```

# do-while-Schleife

- bedingte wiederholte Ausführung, eine Ausführung garantiert
- Syntax:

```
do {  
    statement;  
} while (Test_nach_Durchlauf);
```

- Beispiel:

```
1 int i = 0;  
2 do {  
3     printf("%d ", i);  
4     ++i;  
5 } while (i < 10);
```

Ausgabe:

```
1 0 1 2 3 4 5 6 7 8 9
```



# Schleifen: continue und break

## ■ continue: vorzeitiges Beenden eines Durchlaufs

### ■ Beispiel:

```
1 for (int i = -4; i <= 4; ++i) {  
2     if (i == 0) continue;  
3     printf("%d ", 100/i);  
4 }
```

## ■ break: vorzeitiges Verlassen der Schleife

### ■ Beispiel:

```
1 int i = 0;  
2 while (i >= 0) {  
3     if (i == 10) break;  
4     printf("%d ", i);  
5     ++i;  
6 }
```

# Zusammenfassung

- stark genutzt und verbreitet
- Prägung neuerer Sprachen
- Programmierung
  - Variablen
  - Ausgabe: `printf` und `%`
  - Fallunterscheidung: `if-else`, `?`, `switch-case`
  - Schleifen: `for`, `while`, `do-while`

# Literatur

„C-Programmierung“,

<http://de.wikibooks.org/wiki/C-Programmierung>, 23.04.2014