Grundlagen von C

Jonas Gresens

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

2014-04-24

Gliederung (Agenda)

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 ⇒ Portierbarkeit
    - „C kann ohne Bibliotheken fast gar nichts.“
  - direkte Speicherzugriffe ⇒ Systemprogrammierung
  - nicht typsicher
  - Modularisierung auf Dateiebene (Header-Dateien)
- Verwendung
  - Systemprogrammierung
  - Programmierung von Mikrocontrollern
  - Compiler, Interpreter anderer Sprachen (JVM)

Entwicklung in C

- IDEs: Visual Studio, Eclipse, NetBeans, Anjuta, Geany
- Texteditoren mit Syntax Highligthing: Sublime Text, Notepad++
- Compiler: gcc
  - `-c` Kompilieren der source-Dateien, kein Linken
  - `-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

```c
/*
 * C_Grundlagen_Hello_World!.c
 */

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

int main(int argc, char** argv)
{
    //Print Hello World
    printf("Hello World!");

    return EXIT_SUCCESS;
}
```

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 `identifier(args ...)`
- Semikolon (`;`) am Ende jedes Statements

Basics: Variablen

- Typen: z.B. `int`, `double`, `char`
  - kein `boolean` oder `String`
- Deklaration: Vergabe eines Namens und eines Typs
  - `extern` - reine Deklaration der Variablen
- 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`
- Escapen von bestimmten Zeichen notwendig:
  - `\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 (expr)
    statement;
else
    if (expr)
        statement;
    else
        statement;
```

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

- mehrere `statements` müssen in Blockklammern stehen

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: statement; break;
    case const2: statement; break;
    default: statement; break;
}
```

- folgt auf ein `case` kein `break`, so wird auch das statement des nächsten `case` ausgeführt:

```
case const1: case const2: statement; break;
```

for-Schleife

- bedingte wiederholte Ausführung, eingebaute Laufvariable
- Syntax:

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

- Beispiel:

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

Ausgabe:

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

while-Schleife

- bedingte wiederholte Ausführung
- Syntax:

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

- Beispiel:

```c
int i = 0;
while (i < 10) {
    printf("%d ", i);
    ++i;
}
```

Ausgabe:

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

do-while-Schleife

- bedingte wiederholte Ausführung, eine Ausführung garantiert
- Syntax:

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

- Beispiel:

```c
int i = 0;
do  {
    printf("%d ", i);
    ++i;
} while (i < 10);
```

Ausgabe:

```
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 == 1s0) break;
4     printf("%d ", i);
5     ++i;
6 }
```

Zusammenfassung

- trotz Alter viel genutzt

Literatur