Kurzeinführung in C99

Institut für Numerische Simulation
Rheinische Friedrich-Wilhelms-Universität Bonn

Oktober 2013

Überblick

1. Compiler und Editoren
   - Was wird benötigt um ein Programm zu erstellen
2. Variablen und Datentypen
   - Initialisierung und Deklaration
3. Datentypen
   - Integer, Double, Float, ...
4. Kontrollstrukturen
   - Verzweigungen und Schleifen
5. Schleifen
   - for ,while,...
6. Hilfsmittel
   - Debugger und Profiler

Compiler und Editoren

– Was wird benötigt um ein Programm zu erstellen

Compiler als Übersetzer

- Ein Programm muss dem Computer in Maschinensprache vorliegen, damit es ausgeführt werden kann.
- Der Compiler übersetzt das Programm von C99 in Maschinensprache.
- Es entsteht ein ausführbares Programm (Executable).
- Für die Programmiersprache C gibt sehr viele proprietäre und freie Compiler.

- In der Vorlesung wird die freie GNU Compiler Collection[a] GCC verwendet
- C-Programme werden mit dem *gcc* kompiliert.

[a] `http://gcc.gnu.org/`

Editoren

Editoren

- SciTE[a] (Windows)
- Kate, Gedit, ... (Linux)[b]

[a] `http://www.scintilla.org/SciTE.html`
[b] http://wiki.ubuntuusers.de/Editor

Entwicklungsumgebungen

- Eclipse CDT [a]

[a] `http://www.eclipse.org/cdt/`

Für einen besseren Lernerfolg empfehlen wir die Verwendung eines Editors.

Übersetzen eines C-Programms

Inhalt der Datei Main.c

```
#include <stdio.h>

int main(void)
{

    printf("Hello World\n");
    return 0;
}
```

Kompilieren und Ausführen

```
diehl@ossus ~ $ gcc -std=gnu99 -o run Main.c
diehl@ossus ~ $ ./run
Hello World
```

Variablen und Datentypen

– Initialisierung und Deklaration

Variablen - Behälter für Daten

- Variablen sind Behälter für Daten
- Diese Behälter sind unterschiedlich!
- Nicht jedes Datum passt in jeden Behälter

Variablen - Deklaration

Bevor man die Variable verwenden kann, muss man sie deklarieren, d.h. bekanntmachen, dass sie existiert. Die Variable bekommt dabei einen (sinnvollen!) Namen und einen Typ.

- **char** linebreak ;
- **int** alter ;
- **float** zinssatz ;
- **double** epsilon ;

Variablen - Zuweisungen

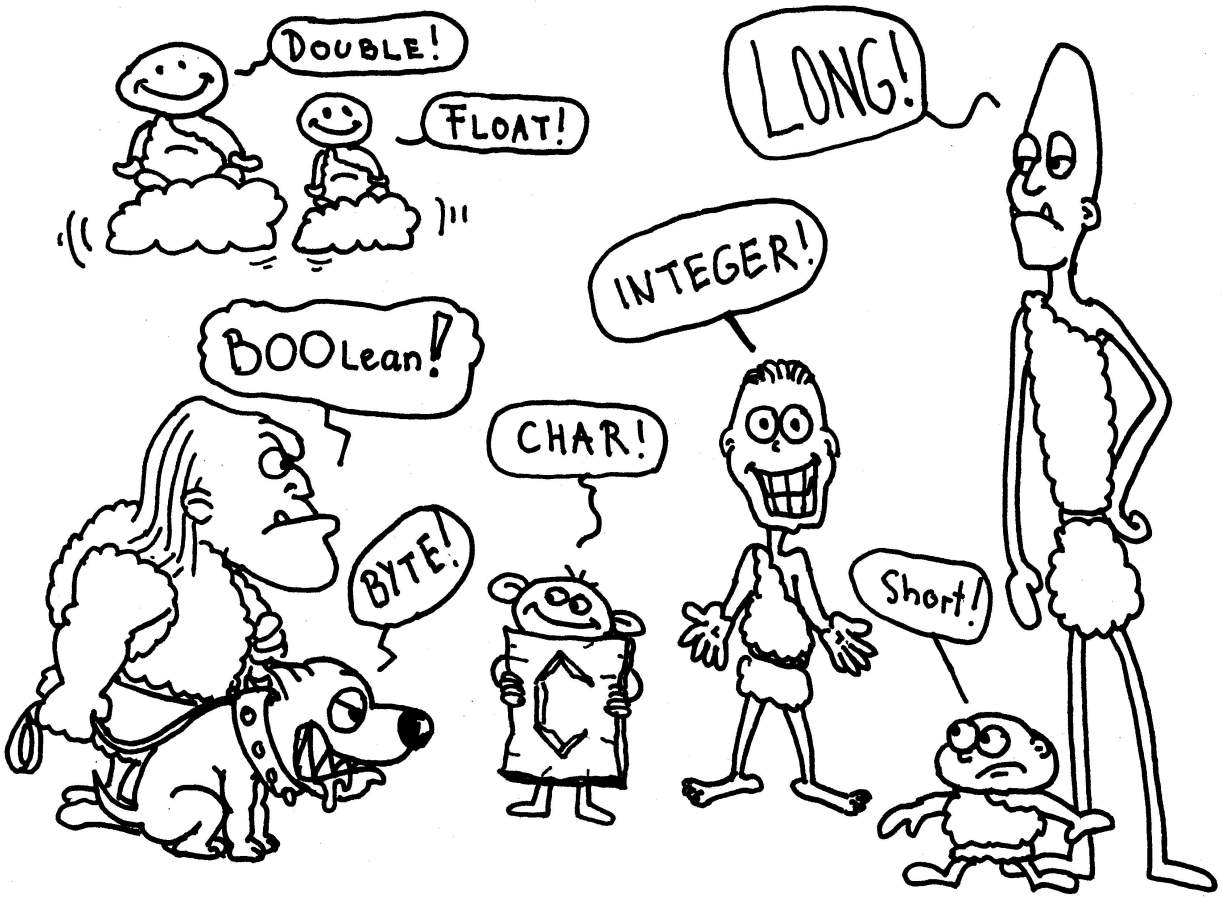
Bei der Deklaration kann man Variablen auch gleich Werte zuweisen:

- **char** linebreak = 27;
- **int** alter = 42;
- **float** zinssatz = 0.025;
- **double** epsilon= 1e6;

Datentypen

– Integer, Double, Float, ...

Datentypen



Zahlen in C

- **unsigned** integer ($\sim \mathbb{N}_0$):
  Ganzzahl zwischen 0 und 65 535
- **int** ($\sim \mathbb{Z}$):
  Ganzzahl zwischen -32 768 und 32 767
- **float** ($\sim \mathbb{R}$):
  Gleitkommazahl zwischen -1.17E-38 und 3.4E38
- **double** ($\sim \mathbb{R}$):
  Gleitkommazahl zwischen 2.2E-308 und 1.8E308

int, **float**, **double** - Operationen

Grundrechenarten:

- **int** zahl $= (2 + 3) * 5 - 9;$

Ganzzahldivision und Rest (%):

- **int** quotient $= 10 / 3;$ **int** rest $= 10 \% 3;$

Mathematische Funktionen und Konstanten

Potenzieren:

- **double** zahl $=$ pow(7.0,2.0);

Kreiszahl π:

- **double** pi $=$ M_PI

Hierfür muss *math.h* eingebunden werden: **#include** <math.h>

Tupel in C

Das Tupel $T = \{1, 2, 3, 4, 5, 6, 7, 8\}$ kann als:

Array

```
int tupel[ ] = { 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 };
```

oder

mittels Pointer

```
int * tupel = malloc(8 * sizeof(int));
tupel[0] = 1;
    ⋮
tupel[7] = 8;
```

dargestellt werden.

Auf das Konzept von Pointern wird später in der Vorlesung eingegangen.

Kontrollstrukturen

– Verzweigungen und Schleifen

Kommentare

- Die wichtigste „Anweisung“ überhaupt!
- So sehen Kommentare aus:

```
// Dies ist ein Kommentar

/*
* Dies ist ein Kommentar
* ueber zwei Zeilen.
*/
```

- Kommentare erhöhen das Verständnis.
- Für Teamarbeit sind Kommentare unabdingbar!

Was macht diese Funktion?

```
#include <math.h>
#include <stdio.h>

float my_func(float a, float b, float c)
{
  float s = b/a;
  float t = c/a;
  float h = 2.0*a;
  float i = 4.0*a*t;
  float j = -s/2.0;
  float x1 = j+sqrt(j*j-t);
  float x2 = j-sqrt(j*j-t);
  // ...
}
```

Mit Kommentaren

```
#include <math.h>
#include <stdio.h>

float solve_quadratic_eq(float a, float b, float c)
{
  // Diskriminante berechnen
  float diskriminante = pow(b,2.0) - 4.0 * a * c;
  float Ergebnis_1, Ergebnis_2;
  // wenn es reelle Loesungen gibt, diese berechnen
  if (diskriminante > 0.0) {
    Ergebnis_1 = (-b + sqrt(diskriminante)) / (2.0 * a);
    Ergebnis_2 = (-b - sqrt(diskriminante)) / (2.0 * a);
  }
  else // sonst im Reellen unloesbar
    printf ("Die Gleichung ist im Reellen unloesbar!");
  // ...
}
```

If-Then-Else

Manchmal muss man Entscheidungen treffen.

- Hat ein Student mehr als 10 Bier getrunken, sollte man eventuell Hilfe holen.
- Sonst, wenn er mehr als 5 Bier hatte, wäre es vielleicht besser, es dabei zu belassen.
- Und wenn er auch die 5 noch nicht erreicht hat, kann die Party ja ruhig weitergehen :)

If-Then-Else:

```
#include <stdio.h>

if (Anzahl_Bier > 10)
    printf ("Geht's dir gut? Sicher?");
else if (Anzahl_Bier > 5)
    printf ("Ich glaub, du hast jetzt erstmal genug.");
else
    printf ("Get the party started!!");
```

Operatoren

Logische Operatoren

x	y	!x	x && y	x || y
true	true	false	true	true
true	false	false	false	true
false	true	true	false	true
false	false	true	false	false

Vergleichsoperatoren

a	b	a == b	a != b	a < b	a > b	a <= b	b >= a
5	9	false	true	true	false	true	false

Schleifen

– **for**,**while**,. . .

Schleifen (**for**)

`for` Inkrementiert oder dekrementiert eine Variable. Bricht ab, wenn die Variable einen vorgegebenen Wert über- oder unterschreitet.

```
#include <stdio.h>

int a [ ] = { 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 };

for (int i = 0; i < 7 ; i++)
{
   printf("%d \n" , a[i]);
}
```

Anwendung: Wenn die Anzahl der Iterationen schon bekannt ist.

Schleifen (**while**)

`while` Prüft bei jedem Durchlauf eine boolsche Bedingung. Bricht ab, wenn diese nicht erfüllt ist.

```
while (error > eps)
{
  error = gaussSeidelIteration(...);
}
```

Verwendung: Wenn die Anzahl der Iterationen noch unbekannt ist, aber durch eine boolesche Bedingung geprüft werden kann.

Hilfsmittel

– Debugger und Profiler

Debugger

Eigenschaften

- Zeigt den Zustand des Programms zum Zeitpunkt des Absturzes an.
- Erlaubt den Zustand des Programms zu jedem Zeitpunkt zu analysieren.

Debugger für C

- GDB: The GNU Project Debugger[a]
- CGDB: The curses debugger[b]

---

[a] `http://www.sourceware.org/gdb/`
[b] `http://cgdb.github.io/`

Ohne Debugger

Beispiel

```
#include <stdio.h>

int main(void)
{
int sum = 42;
int n;
 printf ("Durchschnitt: %d n", sum / n );
return 0;
}
```

Ausführung

```
diehl@ossus ~ $ gcc -std=gnu99 -o prog Main.c
diehl@ossus ~ $ ./prog
Floating point exception
```

Mit Debugger

Verwendung von gdb

```
diehl@ossus ~ $ gcc -std=gnu99 -o prog Main.c -g
diehl@ossus ~ $ gdb prog
⋮
(gdb) run
⋮
Program received signal SIGFPE, Arithmetic exception.
0x000000000040050b in main () at Main.c:9
9  printf ("Average: %d n", sum / n );
(gdb) print n
$1 = 0
```

- Das Programm muss mit -g kompiliert werden.
- Mittels des Debuggers sehen wir leicht, dass die Variable *n* nicht initialisiert wurde.

Profiler

Eigenschaften

- Prüft, ob der verwendete Speicher auch initialisiert wurde.
- Prüft, ob außerhalb der Speichergrenzen geschrieben wird.

Beispiel

```
#include <stdio.h>

int main(void)
{
int i;
 printf ("%d\n", i );
return 0;
}
```

Valgrind[1]

Ausführung

```
diehl@ossus   $ ./run
0
```

Ausführung mit valgrind

```
diehl@ossus   $ valgrind ./run
==11267== Conditional jump or move depends on uninitialised
value(s)
==11267== at 0x4E7C4F1: vfprintf (vfprintf.c:1629)
==11267== by 0x4E858D8: printf (printf.c:35)
```

Der Profiler zeigt an, dass eine Variable nicht initialisiert wurde.

Sehr hilfreich um Fehler zu finden, die nicht offensichtlich im Algorithmus stecken.

[1] http://valgrind.org/