

Computergrundlagen Programmieren in C

Maria Fyta

Institut für Computerphysik
Universität Stuttgart

Wintersemester 2012/13

Die Compilersprache C

Geschichte

- ▶ 1969-1973: entwickelt in Bell Labs von D.M. Ritchie für Systemprogrammierung des Unix
- ▶ 1978: C-Buch ('K&R-C') von Kernighan and Ritchie
- ▶ 1989: Standard ANSI C89 = ISO C90
- ▶ 1999: Standard ISO C99
- ▶ 2011: Standard C11

Die Compilersprache C

Eigenschaften

- ▶ 'general-purpose'
- ▶ imperative und strukturierte Programmiersprache
- ▶ Die meisten physikalische Software sind in C, bzw. Fortran geschrieben
- ▶ C-Code oft deutlich schneller als z.B. Python-Code
- ▶ Besonders bei numerischen Problemen
- ▶ Intensiver Einsatz von Zeigern
- ▶ Kein Schutz gegen Speicherüberläufe

Die Compilersprache C

C-Compiler

- ▶ übersetzt C-Quellcode in Maschinencode
- ▶ GNU gcc, Intel icc, IBM XL C, Portland Group Compiler, ...
- ▶ Für praktisch alle Prozessoren gibt es C-Compiler
- ▶ Compiler optimieren den Maschinencode
- ▶ Compiler übersetzt nur Schleifen, Funktionsaufrufe usw.
- ▶ Bibliotheken für Ein-/Ausgabe, Speicherverwaltung usw.

Anwendungen

- ▶ System- und Anwendungsprogrammierung
- ▶ Grundlegende Programme aller Unix-Systeme
- ▶ Systemkernel vieler Betriebssysteme, größtenteils GNOME, KDE
- ▶ Zahlreiche 'höhere' Programmiersprachen (Perl, Java, C++, C#, Objective-C, usw.) orientieren sich an Syntax von C
- ▶ Python-Interpreter ist in C geschrieben

Hello, World!

```
#include <stdio.h>

int main()
{
    printf("Hallo Welt\n");
    return 0;
}
```

```
#include <stdio.h>
    int main(){printf(
"Hallo Welt\n");return 0;}
```

- ▶ C-Quellcode muss als Textdatei vorliegen (z.B. helloworld.c)
- ▶ **Formatierung**: viele Freiheiten bei der Formatierung des C-Quelltexts (aber falsche Einrückung erschwert Fehlersuche)

Compilieren

- ▶ Vor der Ausführung mit dem GNU-Compiler **compilieren**:
`gcc -Wall -O3 -std=c99 -o helloworld helloworld.c`
- ▶ Erzeugt ein *Binär*programm helloworld, d.h. das Programm ist betriebssystem- und architekturenspezifisch
- ▶ „-Wall -O3 -std=c99“ schaltet alle Warnungen und die Optimierung an, und wählt den ISO C99-Standard anstatt C90

Standard Bibliothek

- ▶ **#include** ist Präprozessor-Anweisung
- ▶ Bindet den Code einer Headerdatei ein
- ▶ Die Standard-Bibliothek ist kein Teil der Programmiersprache C selbst, aber eine Umgebung die Standard-C realisiert
- ▶ Stellt Funktionen, Typen und Makros der Bibliothek zur Verfügung
- ▶ Headerdateien beschreiben Funktionen anderer Module
z.B. `stdio.h` Ein-/Ausgabefunktionen wie `printf`
- ▶ `stdio.h` ist Systemheaderdatei, Bestandteil der C-Umgebung
- ▶ Systemheaderdateien liegen meist unter `/usr/include`

Funktionen

- ▶ Jedes C-Programm besteht aus Funktionen (z.B. `printf` und `main`)
- ▶ Die Funktionen müssen nicht unbedingt einen Rückgabewert geben
- ▶ Das Hauptprogramm (sowie auch die Funktionen) werden durch geschweifte Klammern in Blöcke eingeteilt.
- ▶ Innerhalb des Blocks stehen dann die Anweisungen des C-Programms
- ▶ Anweisungen werden in C durch ein Semikolon `;` abgeschlossen.

main

- ▶ `main`: Hauptroutine, hier startet das Programm
- ▶ Der Rückgabewert von `main` geht an die Shell

Hello, World!

```
#include <stdio.h>

int main()
{
    printf("Hello, World\n");
    return 0;
}
```

- ▶ `printf`: formatierte Textausgabe
- ▶ Rückgabewert vom **Datentyp** `int`, keine Parameter („()“)
- ▶ **return** beendet die Funktion `main`
- ▶ Argument von **return** ist der Rückgabewert der Funktion (hier 0)

Datentypen in C

▶ Grunddatentypen

char	8-Bit-Ganzzahl, für Zeichen	'1','a','A',...
int	32- oder 64-Bit-Ganzzahl	1234, -56789
float	32-Bit-Fließkommazahl	3.1415, -6.023e23
double	64-Bit-Fließkommazahl	-3.1415, +6.023e23

- ▶ **Arrays** (Felder): ganzzahlig indizierter Vektor
- ▶ **Pointer** (Zeiger): Verweise auf Speicherstellen
- ▶ **Structs und Unions**: zusammengesetzte Datentypen, Datenverbände
- ▶ **void** (nichts): Datentyp, der nichts speichert
 - ▶ Rückgabewert von Funktionen, die nichts zurückgeben
 - ▶ Zeiger auf Speicherstellen un spezifizierten Inhalts

Variablen

```
int a,b,c,summe;  
a=1; b=2; c=3;  
summe=a+b+c;  
printf("Die Summe ist %d\n",sum);  
return 0;
```

Hier: a, b, c, summe
sind Ganzzahlen.

- ▶ Alle Variablen in C müssen (nur einmal) vor Benutzung deklariert werden
- ▶ können bei der Deklaration mit Startwert **initialisiert** werden
- ▶ Die Variablenamen können 1-9 Zeichen haben (A-Z, a-z, 0-9, _)
- ▶ Es gibt **lokale** und **globale** Variablen
- ▶ Globale Variablen werden außerhalb von Funktionen deklariert und sind von allen Funktionen les- und schreibbar

Variablen

```
int global;
int main() {
    int i = 0, j, k;
    global = 2;
    int i; // Fehler! i doppelt deklariert
}
void funktion() {
    int i = 2; // Ok, da anderer Gueltigkeitsbereich
    i = global;
}
```

Hier:

- ▶ Ganzzahl `global` : **globale** Variable
- ▶ Ganzzahl `i` : **lokale** Variable
 - ▶ Gültigkeitsbereich ist der innerste offene Block
 - ▶ daher kann `i` in `main` und `funktion` deklariert werden

Typumwandlung

```
int nenner = 1;
int zaehler = 1000;
printf("Ganzzahl: %f\n",
      (float)(nenner/zaehler)); // → 0.000000
printf("Fließkomma: %f\n",
      ((float) nenner)/zaehler); //→ Fließkomma: 0.001000
```

- ▶ C wandelt Typen nach Möglichkeit automatisch um
- ▶ Explizite Umwandlung: geklammerter Typname vor Ausdruck
*Beispiel: (**int**) ((**float**) a) / b)*
- ▶ Umwandlung in **void** verwirft den Ausdruck

Ein-/Ausgabe Funktionen

<code>putchar(c)</code>	Ausgabe eines Zeichens auf der Standard-Ausgabe (Bildschirm)
<code>puts(str)</code>	Ausgabe einer Zeichenfolge
<hr/> <code>printf ("%d + %d ergibt %d.", x, y, ergebnis)</code> <hr/>	formatierte Ausgabe (von Variablen unterschiedlichen Typs)
<code>getchar()</code>	Einlesen eines Zeichens von der Standard-Eingabe (Tastatur)
<code>gets(str)</code>	Einlesen einer Zeichenfolge
<code>scanf ("%d %f %d", &i, &x, &j)</code>	Einlesen von Variablen verschiedenen Typs

Einfaches I/O (getchar, putchar)

```
char c;  
c=getchar();  
putchar(c);
```

- ▶ `getchar()`, `putchar()`: die einfachsten Funktionen in der Standard-Bibliothek die **einzelne** Zeichen lesen/schreiben können

Ausgabe – printf

```
int n=511;
printf("Welche ist die Oktalzahl fuer %d?",n);
printf("%s! %d dezimal ist %o oktal\n","Richtig", n,n);
```

- ▶ '%d' für eine Zahl im Dezimalsystem
- ▶ '%s' gibt einzelne Zeichen aus
- ▶ '%s' gibt eine ganze Zeichenkette (string) aus
- ▶ '%o' gibt eine Oktalzahl statt eine Dezimalzahl aus (ohne führende Null)
- ▶ '%xh' gibt eine Hexadezimalzahl statt eine Dezimalzahl aus
- ▶ '%n' Zeilenendezeichen (newline)
- ▶ '%h' horizontaler Tabulator (Tab)
- ▶ '%v' vertikaler Tabulator

Eingabe – scanf

```
int num1 = 0, num2 = 0;

printf ("Geben Sie die erste Zahl ein: ");
scanf ("%3d", &num1);

printf ("Geben Sie nun die zweite Zahl ein: ");
scanf ("%d", &num2);

printf ("\nDie eingegebenen Zahlen waren:\n");
printf ("num1: %d\nnum2: %d\n", num1, num2);

return 0;
```

2 Ganzzahlen werden eingelesen. Die erste muss max. 3 Stelle haben ('%3d').

längere Eingabe: Rest bleibt im Eingabepuffer und wird von **scanf** weitergelesen.

Bedingte Ausführung – if

if (Ausdruck) Anweisung

```
c=getchar();  
if ( ) c=='?')  
printf("warum hasst du ein Fragezeichen gegeben?\n");
```

Vergleiche und logische Verknüpfungen

- ▶ == gleich
- ▶ != : nicht gleich
- ▶ > (<) : größer (kleiner)
- ▶ >= (<=): größer(kleiner) oder gleich

```
if (a<b) {  
    t=a;  
    a=b;  
    b=t;  
}
```

- ▶ „&&“: logisches „und“
- ▶ „||“ : logisches „oder“
- ▶ „!“ : logisches „nicht“

else Anweisung

if (Ausdruck) Anweisung1 else Anweisung2

```
if(a<b)
    x=a;
else
    x=b;
```

oder

Ausdruck1 ? Ausdruck2 : Ausdruck3:

Ausdruck1 wird ausgewertet; when nicht Null ist der Wert gleich dem Ausdruck2, sonst ist der Wert gleich Ausdruck3. z.B. der Wert von

$a < b ? a : b;$

ist a when a kleiner als b und b sonst.

if, else

- ▶ Es gibt kein 'elseif', aber Blocks von **if**'s und **else**'s `if(...)`

```
{...}  
else if(...)  
  {...}  
else  
  {...}
```

```
if( c==' ' || c=='\t' || c=='\n') ..c leer, ein Tabulator oder  
ein Zeilenendezeichen ist.
```

```
int i=1, j=5, m;  
if (i>j)  
  m=i;  
else  
  m=j;
```

i,j vergleichen um den
Wert von m
auszuwerten

while – Schleife

while (Ausdruck) Anweisung

Die Anweisung wird ausgeführt, wenn der Kontrollausdruck einen Wert ungleich 0 ergeben hatte.

```
int i=0;
  while(i!= 10)
  {
    printf(" i is %d\n", i);
    i++;
  }
```

Inkrement von i, bis i nicht 10 wird

```
int c;
  c=getchar();
  while (c!=EOF)
  {
    putchar(c);
    c=getchar();
  }
```

Dateien kopieren

do while – Schleife

do Anweisung while (Ausdruck);

Die Ausführung der Anweisung wird solange wiederholt, bis der Kontrollausdruck den Wert 0 ergibt. (Im Unterschied zur while-Anweisung wertet die do-Anweisung den Kontrollausdruck erst nach der Ausführung einer zugehörigen Anweisung aus.)

```
int i=1;
do
{
    printf("%d\n", i++);
} while (i<=10);
```

Im Beispiel, werden die Zahlen von 1 bis 10 gegeben.

Arithmetik

Die übliche Rechenzeichen werden verwendet: '+', '-', '*', '/' und Modulo '%'.
x=a%b → a mod b: x ist der Rest der Division a geteilt durch b.

char Variablen können wie **int** Variablen behandelt werden.

z.B. **c=c+'A'-'a'**: wandelt einen kleingeschriebenen ascii Buchstaben der in c gespeichert ist in einen großgeschriebenen Buchstaben um (Voraussetzung: fester Abstand aller ascii Zeichen)

```
char c;  
while((c=getchar())!='\0')  
    if('A'<=c && c<='Z')  
        putchar(c+'a'-'A');  
    else  
        putchar(c);
```

wandelt Groß- in
Kleinschreibung um

Inkrement und Dekrement

Kurzschreibweisen zum Ändern von Variablen:

- ▶ `i += v`, `i -= v`; `i *= v`; `i /= v`
 - ▶ Addiert *sofort* `v` zu `i` (zieht `v` von `i` ab, usw.)
 - ▶ Wert im Ausdruck ist der *neue* Wert von `i`
-

```
int k, i = 0;  
k = (i += 5);  
printf("k=%d i=%d\n", k, i); → i=5 k=5
```

- ▶ `++i` und `--i` sind Kurzformen für `i+=1` und `i-=1`
 - ▶ `i++` und `i--`
 - ▶ Erhöhen / erniedrigen `i` um 1 *nach* der Auswertung des Ausdrucks
 - ▶ Wert im Ausdruck ist also der *alte* Wert von `i`
-

```
int k, i = 0;  
k = i++;  
printf("k=%d i=%d\n", k, i); → i=1 k=0
```

for – Schleife

for (Ausdruck-1_{opt}; Ausdruck-2_{opt}; Ausdruck-3_{opt}) Anweisung

for (Deklaration Ausdruck-2_{opt}; Ausdruck-3_{opt}) Anweisung

```
for (int i = 1; i < 100; ++i) {  
    printf("%d\n", i);  
}
```

for-Schleifen bestehen aus

- ▶ Initialisierung der Schleifenvariablen
 - ▶ Eine hier deklarierte Variable ist nur in der Schleife gültig
 - ▶ Hier kann eine beliebige Anweisung stehen (z.B. auch nur `i = 1`)
 - ▶ Dann muss die Schleifenvariable bereits deklariert sein
- ▶ Wiederholungsbedingung:
die Schleife wird abgebrochen, wenn die Bedingung unwahr ist
(hier, bis `i` bzw. `k = 100`)
- ▶ Erhöhen der Schleifenvariablen

for – Schleife

```
int i, j;
for (i = 1; i < 100; ++i) {
    if (i == 2) continue;
    printf("%d\n", i);
    if (i >= 80) break;
}
for (j = 1; j < 100; ++j) printf("%d\n", i);
```

- ▶ **break** verlässt die Schleife vorzeitig
- ▶ **continue** überspringt Rest der Schleife („forever“)
- ▶ Deklaration in der **for**-Anweisung erst seit C99 möglich
- ▶ Vorteil: Verhindert unbeabsichtigte Wiederverwendung von Schleifenvariablen

Bedingte Ausführung – switch

```
switch (Ausdruck)
{
case Wert1: Anweisung1;
            Anweisung2;
            ...
            break;
case Wert2: Anweisung1;
            Anweisung2;
            ...
            break;
            ...

    default : Anweisungen;
}
```

- ▶ Das Argument von **switch** (Wert) muss ganzzahlig sein
- ▶ Die Ausführung geht bei **case konst:** weiter, wenn wert=konst
- ▶ **default:** wird angesprungen, wenn kein Wert passt
- ▶ Der **switch**-Block wird ganz abgearbeitet
- ▶ Kann explizit durch **break** verlassen werden

switch – Beispiel

```
int eingabe;
printf ("Geben Sie einen Wert ein: ");
scanf ("%d",&eingabe);
switch (eingabe)
{
    case 1: printf ("Eine eins wurde eingegeben.");
            break;
    case 2: printf ("Die Eingabe war zwei.");
            break;
    case 3: printf ("Drei wurde eingegeben.");
            break;
    default: printf ("Es wurde etwas anderes angegeben.");
}
return 0;
```

-
- ▶ Mehrfache Unterscheidungen: Der Eingabenwert muss sich auf drei (hier:1, 2 oder 3) verschiedene Werte überprüft werden.
 - ▶ Mit **switch** lassen sich Verzweigungen bilden.