dbis
Institut für Informatik
Humboldt-Universität zu Berlin

Die Programmiersprache C

1. Überblick und Einführung

Anweisungen, Variablen, Funktionen, Operatoren

Vorlesung des Grundstudiums
Prof. Johann-Christoph Freytag, Ph.D.
Institut für Informatik, Humboldt-Universität zu Berlin
SoSe 2018

Ziel der Vorlesung

- Grundkenntnisse in C
- Unterschiede zu Java kennenlernen
  - Meine Java-Kenntnisse sind beschränkt
- Wichtig:
  - Dynamische Strukturen
  - „Zeigerkonzept“ (engl. pointer)
- Übungen notwendige
  - Praktische Kenntnisse unumgänglich

C versus Java

C

Java

C versus Java

- **C** ist ein offener Geländewagen. Kommt durch jeden Matsch und Schlamm, aber der Fahrer sieht hinterher auch dementsprechend aus.
- **Java** ist ein neues experimentelles Fahrzeug auf Luftkissenbasis. Es bewegt sich auf Straßen aller Art, ist allerdings noch schwer zu steuern. Es ist strengstens verboten Umbauten am Fahrzeug vorzunehmen.
- **Prolog** enthält statt eines Lenkrades eine Automatik, die alle Strassen so lange absucht, bis das gewünschte Ziel erreicht ist.

Eigenschaften von C

dbis
INSTITUT FÜR INFORMATIK
HUMBOLDT-UNIVERSITÄT ZU BERLIN

Einige Eigenschaften

- kompakte Sprache – Reduktion auf das “Wesentliche”
- extensive Nutzung von Prozeduraufrufen/Funktionen
- schwaches Typ-Konzept (im Gegensatz zu Java, PASCAL)
- aber: strukturierte Programmiersprache
- “*Low-Level*”: Bit-orientierte Programmierung ist möglich
- Zeiger (engl. *Pointer*) Implementation: extensive Nutzung von Zeigern für Speicher, Arrays, Strukturen und Funktionen (später)
- effizienter Code kann erzeugt werden
- Portabilität (durch Sprachstandard): es kann Code für verschiedene Rechner erzeugt werden (siehe Werkzeuge!)

Die Stärken von Java

- einfach
- objektorientiert
  Simula-67(1970), Smalltalk(1975), C++(1987), Eiffel(1988)
- architekturunabhängig, interpretativ
  virtuelle Maschine mit Bytecode als Interpreter: P-Code (Pascal, 1974), S-Code(Simula, 1978), M-Code(Modula, 1982)
- getrennte Übersetzung
  Modula-2 (1982)
  C, C++ nur textuelle Inklusion von Header-Files
- typsicher (zuweisungs- und ausdruckskompatibel)
  aber keine Pointer und Adressen (Maschinenorientierung)
  Indextest bei Feldern, "Garbage Collection" von Daten-Objekten

Schwächen von C (gegenüber Java)

- aus softwaretechnischer und sprachtheoretischer Sicht ein gewisser Rückschritt (1978)
- einige unsaubere (unregelmäßige) Sprachkonzepte
- unsichere Sprache (es wird weniger geprüft als in Java), nur unabhängige statt separate Compilation (damit fehleranfälliger)
- Beispiel: Initialisierung lokaler Variablen

Stärken von C (gegenüber Java)

- flexibel
- Präprozessor (Makros, Include-Files, bedingte Compilation)
- Maschinennähe ("Low-level"-Konzepte), damit hohe Laufzeiteffizienz erreichbar
- Betriebssysteme sind in C programmiert
- Objektorientierte Erweiterungen existieren (mit Beibehaltung der Stärkung und Verringerung der Schwächen von C)

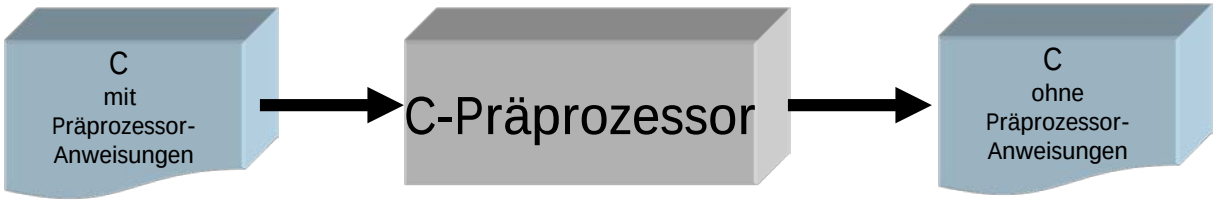
Geschichte der Sprache C

dbis
Institut für Informatik
Humboldt-Universität zu Berlin

Markante Ereignisse

- UNIX wurde etwa 1969 entwickelt (auf einer DEC PDP-7 in Assembler)
- BCPL – eine Sprache mit mächtigen Entwicklungswerkzeugen
  - Erfahrung: Assembler als Entwicklungssprache zu “langatmig” und fehleranfällig
- Sprache “B” als weiterer Anlauf um 1970
- dritter Anlauf: neue Sprache mit neuen Ansätzen (geprägt durch Pascal/Algol): “C” als Nachfolger von “B” (um 1971)
- bis 1973 wurde UNIX fast vollständig in “C” umgeschrieben
- Ur-C: Kernighan & Ritchie, heute aktuell ISO(ANSI)-C==C89, C99

Compilertechnologie für C



- Einfügen von Dateien

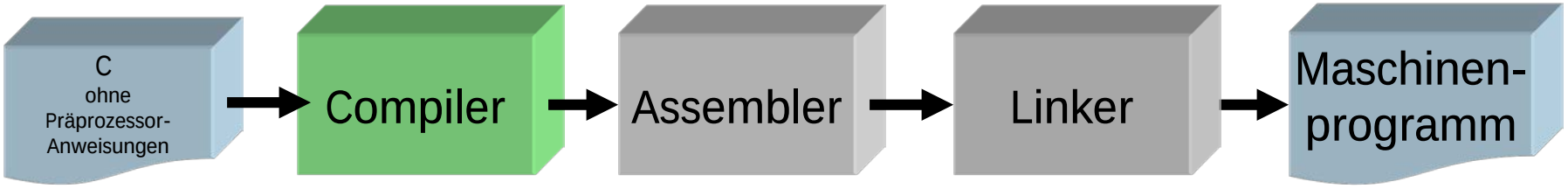
```
#include <stdio.h>      (Systemdatei)
#include "stack.h"      (nutzerdefiniert)
```

- Ersetzen von Text(Makros)

```
#define L 5                     (Konstantendefinition)
#define add100(x) ((x)+100)     (parametrisiert)
```

- bedingte Übersetzung

```
#ifdef TEST
        printf("Testversion") ...
#else
        printf("Produktionsversion") ...
#endif
```



Ausschnitt stdio.h

```
#define BUFSIZ      1024
#define NULL        0
#define FILE        struct _iobuf
#define EOF         (-1)
...
#define stdin       (&_iob[0])
#define stdout      (&_iob[1])
#define stderr      (&_iob[2])

#define getchar()   getc(stdin)
#define putchar(x)  putc((x), stdout)

extern int printf (const char *, ...);
extern int scanf (const char *, ...);
...
```

Sprachelemente in C

- ein C-Programm hat i.allg. folgenden Aufbau:
  - Präprozessor-Kommando(s)
  - Typdefinition(en)
  - Funktionsdeklaration(en):
    Deklaration von Funktionen mit Ein-und Ausgabeparameter(n) (ohne Definition)
  - Variablendefinition(en) und –deklaration(en)
  - Funktionsdefinition(en)

- ***jedes ausführbare Programm*** besitzt genau eine "main()"-Funktion

Anweisungen in C

Algorithmik analog zu Java in Syntax & Semantik

- Zuweisung (auch ++, --)
- if, switch
- while, do-while, for
- break, continue
- Funktionsaufruf (Java:Methodenaufruf)
- return

M: x= y;
...
goto M;

nicht vorhanden
throw, try, catch, synchronized

Zuweisungsoperator in C

- Zuweisung durch "="
- Zuweisung ist KEINE Anweisung, sondern ein Ausdruck!
- C erlaubt Mehrfachzuweisungen
- Beispiel:
  ```
  a=b=c=d=3;
  ```
- ...dies ist äquivalent zu (aber nicht notwendig effizienter als):
  ```
  d=3; c=3; b=3; a=3;
  ```

Typen in C

(1) Einfache Typen

- fehlt: byte
- vorhanden: bool, short, int, long, — unterschiedliche Länge
  float, double
  char — ASCII-Code (1 Byte)
  (void)

nicht in C89, in C99 via <stdbool.h>

(2) strukturierte Typen

- Felder (Arrays), Strukturen, Unions, Zeiger
- Spezielle Felder: Zeichenketten(Strings)

(3) Zeigerarithmetik

- Rechnen mit Adressen (flexibel, effizient, gefährlich)

Einfache Typen in C

C definiert die folgenden einfachen Typen: [mit typischen Implementationsgrößen]

C-Typ	Größe (Byte)	Min-Wert	Max-Wert
bool	1		
char	1	-	-
unsigned char	1	0	255
short int	2	-32768	+32767
unsigned short int	2	0	65535
(long) int	4	-2^{31}	$2^{31} - 1$
float	4	$-3.2 \times 10^{\pm 98}$	$+3.2 \times 10^{\pm 98}$
double	8	$-1.7 \times 10^{\pm 908}$	$+1.7 \times 10^{\pm 908}$
long long	8	-2^{63}	$2^{63} - 1$

auch signed,aber VORSICHT

Typdefinitionen

- nutzerdefinierte Typen werden mittels

  typedef Typkonstukt Typname;

  definiert

- Benutzung neuer Typen: wie vordefinierte C-Typen

Beispiel:

- /* Typdefinition */

  typedef float real;

  typedef char letter;

- /* Variablendefinition */

  real sum=0.0;

  letter nextletter;

Variablendefinition

In UNIX-Systemen ist zumeist int und long identisch

- es sei denn, int-Variablen sind explizit als short definiert

Bemerkung:

- es gibt keinen Boolean-Typ in (K&R- und ANSI-) C, wohl aber in C99
- stattdessen kodieren als char, int oder (besser) als unsigned char

“unsigned” kann mit allen char und int Typen genutzt werden,
ACHTUNG: Portabilitätsprobleme bei signed möglich !!!

Variablendeklaration in C:

- var_type list_variables;
- Beispiel

  int i,j,k; float x,y,z; char ch;

Konstanten

- ANSI-C erlaubt die Angabe von Konstanten

  Beispiel

  int const a = 1;

  const int a = 2;

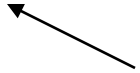
  Bemerkung

  - Konstantendefinition kann vor oder nach der Typdefinition erfolgen
  - alternativ (aber nicht besser):
    Definition von Konstanten durch den C-Präprozessor (mehr dazu später)

Funktionen in C

- eine Funktionsdefinition hat folgende Form:

  type function_name (parameters)
  { local variables C-Statements }

  < C99

- jede Funktion muss vor ihrem Aufruf per Prototypdeklariert werden !

  type function_name (parameters);

Funktionsbeispiel

- Beispielprogramm

```
/* sample program */
#include <stdio.h>
int main()
{   printf("I like C\n" ); return (0);   }
```

Bemerkung:

- C erfordert ein “;” am Ende eines Ausdrucks, um ihn zur Anweisung zu machen (Funktionsruf ist Ausdruck, NICHT Anweisung) !
- *printf()* ist eine Standard-C-Funktion
- “\n” erzeugt ein mit “new line” formatierte Ausgabe (später mehr)
- *return* ist ein Statement, das zum Beenden der Funktion führt (hier nicht unbedingt notwendig!)
  - kann gefolgt werden von einem Ausdruck (expression)

Definition globaler Variablen

- globale Variablen werden vor dem main()-Programm wie folgt definiert:

```
short number, sum;
int bignumber, bigsum;
char letter;
int main() { ...  }
```

- möglich: Initialisierung globaler Variablen mittels Zuweisungsoperator ("=")

- Beispiel:

```
float sum=0.0;
int bigsum=0;
char letter='A';
int main() {   }
```

Ausgabe von Variablen

- C erlaubt formatierte Ausgaben mittels printf()-Funktion
- printf() benutzt das spezielle Zeichen % zur Formatierung
  - %c : characters
  - %d : integers
  - %f : floats
  - %s : strings, á la "Hallo"
- Beispiel:
  - printf(" %c %d %f \n", ch, i, x);

- Bemerkung:
  - Formatanweisungen werden in Hochkommas eingeschlossen ("..."), danach folgen Variablen, die ausgegeben oder eingelesen werden sollen
  - Der Programmierer ist dafür verantwortlich, dass Formatangaben und Typen der Variablen übereinstimmen,
    sonst undefiniertes Verhalten (z. B. core dump) !!!

Eingabe von Variablen

- C erlaubt formatierte Eingaben mittels scanf()-Funktion von einfachen Werten und Datenstrukturen
- Formatierung ähnlich zu printf

  scanf("%c %d %f", &ch, &i, &x);
- Bemerkung:

  Das Zeichen “&” muss vor der Variable stehen (mehr dazu später)

Dateioperationen

- Bisher Operationen für Bildschirmein-/Ausgabe
- Notwendigkeit: Dateiein-/Ausgabe
  - fscanf und fprintf
    - Achtung: “f” deutet auf “File”-Operation hin
- Weiterhin: Datei öffnen und schliessen
  - fopen (name, modus) und fclose (fileptr)
  - modus kann sein: "r", (lesend), "w", (schreibend), "a", (anhängend)
- Beispiel:

```
FILE *fopen(), *fp;

fp = fopen (name, "r");
fscanf(fp, "%d",  &r );
printf ("%d",  r );
fclose ( fp )
...
```

Arithmetische Operationen

- arithmetische Standardoperatoren: + - * / %
- und es gibt noch mehr....
  - ++ und -- mit Variablen in Präfix- und Postfixmodus, also ++x, x++
  - Semantik: erhöhen/reduzieren um den Wert 1

Beispiel

```
int x,y,w;
int main()
{   x=((++y)-(w--)) % 100;   }
```

ist äquivalent zu

```
int x,y,w;
int main()
{   ++y; x=(y-w) % 100; w--;   }
```

Arithmetische Operationen

- Modulo-Operator “%” ist nur für int-Typ definiert
- Division “/” ist für *int* und *float* definiert
- Bemerkung:
  - Ergebnis von x = 3 / 2 ist 1, selbst wenn x als float definiert wurde!!
- Regel:
  - sind beide Argumente von "/" als integer definiert, wird die Operation als *integer* -Division durchgeführt
- korrekte Spezifikation:
  - x = 3.0 / 2 oder x= 3 / 2.0
  - oder (besser) x = 3.0 / 2.0

“Kurzform” von Operatoren

- C stellt “elegante” Abkürzungen für Operatoren zur Verfügung
  - Beispiel: i = i + 3 oder x = x*(y + 2)
- Umschreibung in C (generell) in “Kurzform”:

  expression_1 op = expression_2
- Dies ist äquivalent zu (und u. U. effizienter als):

  expression_1 = expression_1 op expression_2
- Beispiel umgeformt:
  - i = i + 3 als i += 3
  - x = x*(y + 2) als x *= y + 2.
- Bemerkung:
  - x *= y + 2 bedeutet x = x*(y + 2) und nicht x = x*y + 2.

Vergleichsoperatoren

- Test auf Gleichheit: "=="

  Achtung: Bitte "=" nicht mit "==" verwechseln !!!

- zulässig ist auch: if (i = j) ...
  - legales C-Statement (aus syntaktischer Sicht): Zuweisung des Wertes von "j" nach "i", gleichzeitig Wert des Ausdrucks, der als TRUE interpretiert wird, falls j ungleich 0 ist
  - manche Compiler (nicht alle) warnen

Vergleichsoperatoren (Forts.)

- ungleich ist: "!="
- andere Operatoren
  - < (kleiner als)
  - > (größer als)
  - <= (kleiner oder gleich),
  - >= (größer oder gleich)

Logische Operatoren

Die logischen Grundoperatoren sind:

- && für logisches AND
- || für logisches OR
- ! Für logisches NOT

Achtung: & und | existieren auch als zweistellige Operatoren, haben aber eine andere Semantik:

- Bit-orientiertes AND
- Bit-orientiertes OR (später)

Achtung: & ist auch ein einstelliger Operator (später)

- Verwendung in logischen Ausdrücken (als int bewertet)

Präzedenzen von Operatoren

- Bedeutung von a + b * c
  - Gemeint könnte sein
    - (a + b) * c
    - a + (b * c)
  - alle Operatoren besitzen einen “Präzedenzwert” (Priorität)
  - Operatoren mit hoher Priorität werden vor Operatoren mit geringerer Priorität evaluiert
  - Operatoren mit gleicher Priorität werden von links nach rechts evaluiert, wenn sie rechts-assoziativ sind:
    - a - b - c wird als (a - b) - c evaluiert
- im Zweifelsfalle besser ein Klammerpaar zu viel, als eines zu wenig

Präzedenzordnung

- Operatoren in C von hoher bis niedriger Priorität (Präzedenz):
  (sind noch nicht alle eingeführt):

  ( ) [ ] -> .
  ! - * & sizeof *cast* ++ -- (diese werden von rechts nach links ausgewertet)
  / %
  + -
  < <= >= > == !=
  &
  |
  &&
  ||
  ?: (rechts nach links)
  = += -= .... (rechts nach links)
  , (comma)

- **Beispiel**: "a < 10 && 2 * b < c" wird als
  (a < 10) && ((2 * b) < c) interpretiert

Fragen