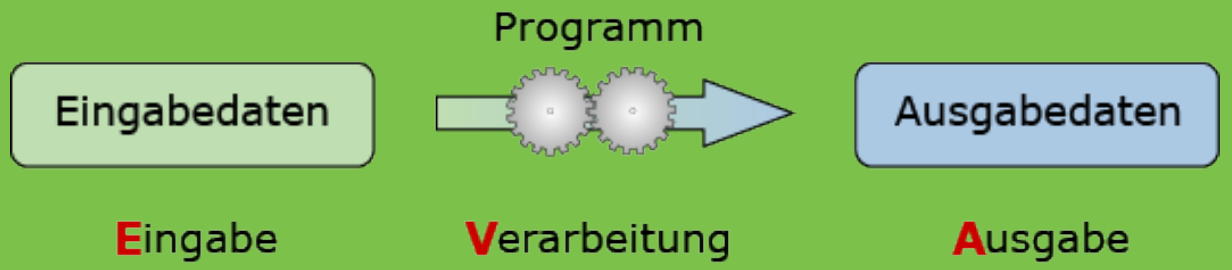
Hochschule Fulda
University of Applied Sciences

Programmierung 1 – Primitive Datentypen



Yvonne Jung

Primitive Datentypen in C

Hochschule Fulda
University of Applied Sciences

[*] gilt auch bei **unsigned**

Ganzzahlige Datentypen mit vorangestelltem `unsigned` sind vorzeichenlos

Größe in Byte bei…	OS 16 Bit	OS 32 Bit	OS 64 Bit
char[*]	1	1	1
short[*]	2	2	2
int[*]	2	4	4
long[*]	4	4	4 / 8
float	4	4	4
double	8	8	8
bool	≥ 1	≥ 1	≥ 1

Speicherbedarf einer Variablen/ eines Datentyps ermittelbar mit Operator *sizeof()*

#include <stdbool.h>

```
1 == sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)
```

Typumwandlungen

Hochschule Fulda
University of Applied Sciences

- Nötig, wenn zwei Operanden in Ausdruck verschiedene Typen haben
  - Typumwandlungen werden soweit nötig u. möglich implizit durchgeführt
    - Beispiel:
```
int a = 12, b = 18;
float f = b / a;
printf("%f\n", f);
```
    - Problem: **`f`** hat Wert 1.0, weil Typumwandlung von *int* auf *float* erst nach Division erfolgt
- Explizite Typumwandlung (Type Cast) mit Cast-Operator
  - Letzterer ist in Klammern gesetzte Typangabe vor Ausdruck
    - Beispiel:
```
int a = 12, b = 18;
float f = (float)b / a;
printf("%f\n", f);
```
    - Lösung: **`b`** wird explizit auf `float` gecastet und damit implizit auch **`a`**, also ist **`f`** nun 1.5

Konstanten

Hochschule Fulda
University of Applied Sciences

- In C ursprünglich mit Hilfe von Präprozessordirektiven
  - Letztere sind reine Ersetzung von Zeichenketten vor Kompilierung
    - Werden meist direkt nach Include-Direktiven angegeben
    - Präprozessor geht Code vor eigentlicher Übersetzung durch und reagiert auf ‚#‘
    - Bindet Include-Dateien in Code ein und fügt Definitionen durch Textersatz ein
  - Beispiel: `#define PI 3.14159265358979323846264338327950288`
    - Achtung: kein Semikolon am Ende, da bei Benutzung reiner Textersatz stattfindet!
    - Konstanten werden typischerweise in Großbuchstaben geschrieben
- Typsicher definierbar über Typ-Qualifizierer **const**
  - Durch zusätzliches Schlüsselwort `const` kann Variable nur gelesen werden
  - Beispiel: `const double Pi = 3.14159265358979323846264338327950288;`

Formatzeichen printf / scanf

Hochschule Fulda
University of Applied Sciences

```
%c    Einzelzeichen, Typ char
%s    Zeichenkette, d.h. char[]
%d    Ganzzahl als Dezimalzahl, Typ int
%x    Ganzzahl als vorzeichenlose Hexadezimalzahl
%p    Speicheradresse (Zeigerwert)
%hd   Ganzzahltyp short int
%ld   Ganzzahltyp long int
%lld  Ganzzahltyp long long
%u    vorzeichenlose Ganzzahl: unsigned int
%llu  vorzeichenlose Ganzzahl: unsigned long long
%f    Gleitkommazahl als float
%lf   Gleitkommazahl als double
%e    Gleitpunktzahl in Exponentendarstellung
```

Wichtige Steuerzeichen

Hochschule Fulda
University of Applied Sciences

```
\n    Newline (Zeilenvorschub)
\r    Carriage Return (Wagenrücklauf)
\t    Tabulator
\b    Backspace
\0    Endezeichen in String
\'    einfaches Anführungszeichen '
\"    doppeltes Anführungszeichen "
%%    Prozentzeichen %
\\    Escapezeichen \
```

- Hinweis: Textdateien unter Unix, Linux u. MacOS verwenden für Zeilenumbruch die Escape-Sequenz **`\n`**, während unter Windows **`\r\n`** dafür verwendet wird

Zusammengesetzte Operatoren

Hochschule Fulda
University of Applied Sciences

- C erlaubt abgekürzte Schreibweise für bestimmte Zuweisungen

+=	x += <Ausdruck>	←	x = x + <Ausdruck>
-=	x -= <Ausdruck>	←	x = x - <Ausdruck>
***=**	x *= <Ausdruck>	←	x = x * <Ausdruck>
/=	x /= <Ausdruck>	←	x = x / <Ausdruck>
%=	x %= <Ausdruck>	←	x = x % <Ausdruck>

  - Beispiel: `count += 2` ist gleichwertig mit `count = count + 2`
- Achtung: funktioniert in C nicht für boolesche Ausdrücke
  - Geht aber für korrespondierende Bitoperationen & bzw. | (→ später)

Inkrement und Dekrement

Hochschule Fulda
University of Applied Sciences

- C kennt Inkrement- und Dekrement-Operatoren
  - Beide Operatoren sind unär (haben also nur einen Operanden) und können sowohl als Präfix- als auch als Postfix-Operatoren verwendet werden
- i++: Der Wert der Variablen *i* wird inkrementiert (d.h. um 1 erhöht)
  - Der Wert des Ausdrucks ist allerdings der *vorherige* Wert
- ++i: Der Wert der Variablen *i* wird um 1 erhöht
  - Der Wert des Ausdrucks ist nun dieser *neue* Wert
- i--: Der Wert der Variablen *i* wird dekrementiert (d.h. um 1 erniedrigt)
  - Der Wert des Ausdrucks ist allerdings der *vorherige* Wert
- --i: Der Wert der Variablen *i* wird um 1 erniedrigt
  - Der Wert des Ausdrucks ist nun dieser *neue* Wert

Übung

Hochschule Fulda
University of Applied Sciences

- Was ist die Ausgabe des linken Programmstücks?

```
int j = 3;
j++;
printf("%d ", j);
++j;
printf("%d ", j);
printf("%d ", ++j);
printf("%d ", j++);
printf("%d ", j);
```

```
int i = 1, k = 0;
if (++i > 1)
    k = 5;
if (i++ > 2)
    k = 10;
printf("%d ", k);
```

- Was ist die Ausgabe des rechten Programmstücks?

Auswertungs-Reihenfolge

- Assoziativität
  - Besagt, in welcher Reihenfolge pro Zeile ausgewertet wird
    - Von links nach rechts oder von rechts nach links
- Präzedenz
  - Beschreibt, welcher Operator zuerst ausgewertet wird
    - Operatoren mit kleinster Präzedenz-Nummer (in Abbildung rechts) werden zuerst ausgewertet

https://en.cppreference.com/w/c/language/operator_precedence

Precedence	Operator	Description	Associativity

1	`++ --`
	`( )`
	`[ ]`
	`.`
	`->`
	`(`*type*`){`*list*`}`
		Suffix/postfix increment and decrement
		Function call
		Array subscripting
		Structure and union member access
		Structure and union member access through pointer
		Compound literal (C99)
			Left-to-right

2	`++ --`
	`+ -`
	`! ~`
	`(`*type*`)`
	`*`
	`&`
	`sizeof`
	`_Alignof`
		Prefix increment and decrement
		Unary plus and minus
		Logical NOT and bitwise NOT
		Type cast
		Indirection (dereference)
		Address-of
		Size-of
		Alignment requirement (C11)
			Right-to-left

3	`* / %`	Multiplication, division, and remainder	Left-to-right
4	`+ -`	Addition and subtraction	
5	`<< >>`	Bitwise left shift and right shift	
6	`< <=` `> >=`	For relational operators < and ≤ respectively For relational operators > and ≥ respectively	
7	`== !=`	For relational = and ≠ respectively	
8	`&`	Bitwise AND	
9	`^`	Bitwise XOR (exclusive or)	
10	`|`	Bitwise OR (inclusive or)	
11	`&&`	Logical AND	
12	`||`	Logical OR	
13	`?:`	Ternary conditional	Right-to-Left

14	`=`
	`+= -=`
	`*= /= %=`
	`<<= >>=`
	`&= ^= |=`
		Simple assignment
		Assignment by sum and difference
		Assignment by product, quotient, and remainder
		Assignment by bitwise left shift and right shift
		Assignment by bitwise AND, XOR, and OR	

15	`,`	Comma	Left-to-right

Vielen Dank!

Noch Fragen?