

Einführung in die Programmiersprachen C und C++

Prof. Dr. Ulf Rehmann, Fakultät für Mathematik

Übungsblatt 2 (4 Seiten)

http://www.mathematik.uni-bielefeld.de/~rehmann/CC++2000/

Elementare Datentypen in C und C++

- char - 1 Byte, kann 'einen Charakter' (z. B. 'a', '3', '&', ...) enthalten.
- int - ganze Zahl, Größe ist implementationsabhängig.
- float - Gleitkommazahl einfacher Genauigkeit.
- double - Gleitkommazahl doppelter Genauigkeit.

int kann qualifiziert werden durch unsigned, short, long zu unsigned int, short int, long int, oder kurz zu unsigned, short, long.

char kann qualifiziert werden zu unsigned char

Bereichsbeispiele:

	PC	Viele Workstations
char	-128 -- +127 (1 Byte)	-128 -- +127 (1 Byte)
short	-32768 -- +32767 (2 Bytes)	-32768 -- +32767 (2 Bytes)
int	-32768 -- +32767 (2 Bytes)	(4 Bytes)
long	-2147483648 -- +2147483647 (4 Bytes)	(8 Bytes)

Die Fließkommatypen sind gleich für PC/Workstations (IEEE-Format) (siehe /usr/include/float.h):

float	+/- 1.7 E-38 -- 3.4 E+38 (4 Bytes)
double	+/- 2.2 E-308 -- 1.7 E+308 (8 Bytes)

Schlüsselwörter von C und C++ - und damit verboten als Namen - sind die folgenden (mit ++ sind Schlüsselwörter von C++ bezeichnet, die nicht auch Schlüsselwörter von C sind):

asm	continue	float	new++	signed	try++
auto	default	for	operator++	sizeof	typedef
break	delete++	friend++	private++	static	union
case	do	goto	protected++	struct	unsigned
catch++	double	if	public++	switch	virtual++
char	else	inline++	register	template++	void
class++	enum	int	return	this++	volatile
const	extern	long	short	throw++	while

C - und C++ - Operatoren:

1. Unäre Operatoren:

+, - (Vorzeichen), ! (Negation), ~ (Bit-Komplementierung), ++ (Inkrement), -- (Dekrement, beide als Prä- oder Postoperatoren), * (Umlenkung, Indirection), & (Adressoperator), sizeof (liefert Größe eines Datentyps in Byte).

Operatoren, angewandt auf Ausdrücke, ergeben wieder Ausdrücke, die Werte haben. Die In-/Dekrement-Operatoren unterscheiden sich in ihrer Prä- oder Postfix-Notation:

int a, b; a = 5; b = a++;

Wert von a wird nach b kopiert, danach(!) wird inkrementiert; liefert als Resultat: b hat den Wert 5, a hat den Wert 6.

Dagegen:

int a, b; a = 5; b = ++a;

Wert von a wird erst(!) inkrementiert, der neue Wert von a wird nach b kopiert. Resultat: b hat den Wert 6, a hat den Wert 6.

2. Binäre Operatoren:

2.1. Arithmetische Operatoren:

+, -, *, /, % (Rest bei ganzzahliger Division)

2.2. Relationale (logische) Operatoren:

< (kleiner), <= (kleiner-gleich), > (größer), >= (größer-gleich), == (gleich), != (ungleich), && (logisches 'UND'), || (logisches 'ODER'),

2.3. Bit-arithmetische Operatoren:

<<, >> (Bitshift-Operatoren).

Beispiele:

a << 3 verschiebt die Bitfolge von a um 3 Positionen nach links bei Einfügen von Nullen: a = 5 hat z. B. die Bitcodierung 101. Also: a << 3 liefert 101000 \cong 40 bei "Standard"-Bitcodierung.

a >> 2 verschiebt die Bitfolge von a um 2 Positionen nach rechts: a = 50 hat die Bitcodierung 110011010. a >> 2 ergibt 1100 \cong 12. (Siehe das Programm bitshift.c.)

& (bitweise 'UND'):

00101000 & 00110010 liefert 00100000,

| (bitweise 'ODER'):

00101000 | 00110010 liefert 00111010,

^ (bitweise ausschließliches 'ODER'):

00101000 ^ 00110010 liefert 00011010.

2.4. Cast-Operator: (Typ-Name) Ausdruck macht Ausdruck zu einem Objekt vom Typ Typ-Name.

Beispiel: int a; double e=2.718; a = (int)e; liefert den Wert a = 2.

2.5. Funktions- und Array-Operatoren:

() , [], Beispiele: int ggt(int a, int b); char s[20];

2.6. Komponenten-Operatoren für Strukturen: . und ->. In C++ gibt es darüber hinaus: .* und ->.*

3. Konditionaloperator (ternär):

Syntax: Ausdruck ? ausdruck2 : ausdruck3

Liefert als Wert ausdruck2, falls ausdruck1 ungleich 0, und ausdruck3 sonst (vergleiche if - else).

Beispiel: int a = 3, b = 5, c = (a > b) ? a : b; liefert für c den Wert 5.

4. Zuweisungsoperatoren:

=, Beispiel: a = 17;

Eine Zuweisung liefert einen Wert, nämlich den der zugewiesenen Größe, also sind Mehrfachzuweisungen w

a = b = u = 17; möglich.

+=, Beispiel: a += 2 ist logisch äquivalent mit a = a + 2, analog für jeden (bit-)arithmetischen binäre

Operator: a /= b ist äquivalent mit a = a/b usw.

5. Kommaoperator: , (gruppiert Ausdrücke als Trennungszeichen oder Separator).

Operatorenpräzedenz in C:

Operatoren	Assoziativität
() [] -> .	links nach rechts
! ~ ++ -- + - * & (type) sizeof	rechts nach links
* / %	links nach rechts
+	links nach rechts
< << > >>	links nach rechts
< <= > >=	links nach rechts
== !=	links nach rechts
&	links nach rechts
~	links nach rechts
	links nach rechts
&&	links nach rechts
	links nach rechts
?:	rechts nach links
= += -= *= /= %= &= ^= = <<= >>=	rechts nach links
,	links nach rechts

Die unären +, -, *, & haben

höhere Präzedenz als die binäre

Formen. Die Assoziativität regelt

für den Fall, dass ein Operator

zwischen zwei Operatoren gleich

Priorität steht, ob der linke oder

der rechte dieser Operatoren a

erster genommen wird, falls bei

Möglichkeiten sinnvoll sind:

a - b + c bedeutet also:

(a - b) + c, nicht etwa:

a - (b + c), während

a = b += 3 die Bedeutung

a = (b += 3) hat, oder,

noch ausführlicher:

b = b+3, a = b.

Beispiel: Testen Sie z. B. die Wirkungsweise des `?:`-Operators (cf. Nr. 3 oben) und des `<<`-Operators (cf. 2.3 oben) mit folgendem Programm:

```
/* test.c */
#include <stdio.h>

main() {
    int a;

    while (1) {
        printf("Eingabe von a und b: ");
        scanf("%d %d", &a, &b);
        printf("Maximum von %d, %d ist: %d\n", a, b, (a>b ? a : b));
        printf("%d << %d liefert: %d\n", a, b, a<b);
    }
}
```

Aufgabe 2.1: Was bedeutet -a++ ? Gibt es einen Unterschied zu ++a ? Machte (-a)++ einen Sinn?

Aufgabe 2.2: Testen Sie mit einem kleinen Programm alle oben angegebene Operatoren auf ihre Wirkungsweise, indem Sie Werte von Ausdrücken ausgeben, in denen die Operatoren auftauchen.

Arrays (oder Felder) in C

Mit der Deklaration `char flags[1000]`; deklariert man ein Array von 1000 Variablen vom Typ `char`, die die Namen `flags[0]`, `flags[1]`, ..., `flags[999]` haben. Analoges gilt auch für andere Datentypen:

`int primzahlen[200]`; deklariert ein Array von 200 Variablen vom Typ `int` mit den Namen `primzahlen[0]`, ..., `primzahlen[199]`, usw.

Man verwendet diese Variablenamen genau wie gewöhnliche Variablenamen; mit

```
primzahlen[0] = 2;
primzahlen[1] = 3;
primzahlen[2] = 5;
....
```

weist man ihnen Werte zu, mit `printf("%d", primzahlen[199])`; druckt man sie aus, mit

```
int i = 17;
int z = primzahlen[i];
```

weist man der `int`-Variablen `z` den Wert von `primzahlen[17]` zu, usw.

Ein Beispiel für die Verwendung von Arrays liefert das folgende Programm:

```
/* primzahlen.c
/* Das zweitaetelste Computerprogramm der Welt -: von anno -250.
/* Copyright: ERATOSTHENES, Philologe, Informatiker und Geograph
*/

#include <stdio.h>

#define TRUE 1          /* Vorabdeklaration einiger Werte */
#define FALSE 0
#define MAX 4000000

char flags[MAX];

/* Deklaration eines Feldes ("array") von .. */
/* MAX Variablen flags[0], ... flags[MAX - 1] */
/* vom Typ "character"; 1 byte */
```

```
main() {
    int i, prime, k, count, size;

    printf("Bestimmung aller Primzahlen bis "); scanf("%d", &size);
    size = (size+1)/2-2; count = 1; /* Nur ungerade Zahlen werden gesiebt */

    for (i = 0; i <= size; i++) /* Initialisiere das ... */
        flags[i]=TRUE; /* ... ganze Feld */

    for (i = 0; i <= size; i++) {
        if (flags[i]) { /* Ist flag[i]=TRUE, so ist i+i+3 prim */
            prime = i+i+3; k=i+prime;
            while (k<size) { /* also streiche alle ... */
                flags[k] = FALSE; k += prime; /* seine nichttrivialen */
            } /* Vielfachen */
            count++; /* und zaehle. */
        }
    }

    printf("\n%d Primzahlen\n", count);
}
```

Aufgabe 2.3: Modifizieren Sie das Programm primzahlen.c derart, dass die Primzahlen der Reihenfolge nach ausgegeben werden.

Aufgabe 2.4: Das Programm primzahlen.c profitiert davon, dass es die gemunden Zahlen, abgesehen von der 2, gar nicht erst in Betracht zieht, weil dies ja sowieso keine Primzahlen sind. Auf diese Weise reicht ein flags-Array etwa der Größe 1000, um alle Primzahlen bis 2000 zu bestimmen. Modifizieren Sie das Programm derart, dass von vornherein auch die durch 3 teilbaren Zahlen nicht betrachtet werden. Wie weit gelangt man dann mit einem flags-Array der Größe 1000?

Das folgende Programm zeigt, wie man ein Array von Strings (= Zeichenketten oder Wörter) definieren und verwenden kann.

```
/* tage.c */
#include <stdio.h>

char *tage[] = { "Sonntag", "Montag", "Dienstag",
                 "Mittwoch", "Donnerstag", "Freitag", "Sonabend" };

main() {
    int i;
    for (i=0; i<7; i++) {
        printf("Der %d-te Wochentag ist %s\n", i, tage[i]);
    }
}
```

Aufgabe 2.5: Schreiben Sie ein Programm, das folgendes leistet: Nach Eingabe dreier Zahlen, etwa 13, 10, 2000, erfolgt die Ausgabe: 13. Oktober 2000, und analog für andere Daten des laufenden Jahres.

Hinweis: Definieren Sie ein Array `char *monat[]` derart, dass z. B. `monat[10]` den Wert "Oktober" hat. Für Einzelzeile: Erweitern Sie das Programm, so dass die Ausgabe lautet: Freitag, 13. Oktober 2000 usw. (Andere Jahre?)