

# *Programiranje 1*

## *7. predavanje*

Saša Singer

`singer@math.hr`

`web.math.pmf.unizg.hr/~singer`

PMF – Matematički odsjek, Zagreb

# Sadržaj predavanja

- **Operatori i izrazi** (drugi dio):
  - Unarni operatori inkrementiranja i dekrementiranja.
  - `sizeof` operator.
  - Relacijski operatori.
  - Logički operatori.
  - Skraćeno računanje logičkih izraza.
  - Operatori nad bitovima.
  - Operator pridruživanja i složeni operatori pridruživanja.
  - Uvjetni operator `? : .`
  - Operator zarez `,`
  - Tablica prioriteta i asocijativnosti operatora.

# Informacije

## Anketa:

- Koliko vas je instaliralo Code::Blocks
- i probalo izvršiti neki program?

# Operatori i izrazi (prvi dio) ponavljanje

# Sadržaj

- **Operatori i izrazi** (prvi dio) — ponavljanje:
  - Aritmetički operatori.
  - Pretvaranje tipova u aritmetičkim izrazima.

# Aritmetički operatori

U programskom jeziku C postoji 5 aritmetičkih operatora:

operator	značenje
+	zbrajanje, unarni plus
-	oduzimanje, unarni minus
*	množenje
/	dijeljenje
%	ostatak (modulo)

Operatori - i + imaju dva različita značenja, ovisno o tome kako se operator napiše — obzirom na operand(e).

# Aritmetički operatori (nastavak)

Operator **promjene** predznaka **-** je **unarni** operator i

● piše se **ispred** operanda

(tzv. **prefiks** notacija).

---

`-operand`

---

Slično vrijedi i za **unarni +** (ne mijenja predznak).

Ostali **aritmetički** operatori su **binarni** i

● pišu se **između** dva operanda

(tzv. **infiks** notacija).

---

`operand_1 operacija operand_2`

---

# Aritmetički operatori (nastavak)

Aritmetički operatori djeluju na **numeričke** operande raznih tipova. **Operandi** mogu biti:

- nekog **cjelobrojnog** tipa,
- nekog **realnog** tipa,
- **znakovnog** tipa (**char** se prikazuje kao cijeli broj).

**Problem.** Kad operandi **nisu** istog tipa, kojeg **tipa** je **rezultat**?

- Tada dolazi do **konverzije** ili pretvaranja **tipova** po određenim pravilima (v. malo kasnije).

Za početak, pogledajmo kako **radi cjelobrojno dijeljenje!**



# Cjelobrojno dijeljenje

Operacija **dijeljenja**  $/$ , u slučaju kad su

• oba operanda cjelobrojna,

daje **cjelobrojan** rezultat (operacija **div** od ranije).

Po **C99** standardu, rezultat se uvijek dobiva **zaokruživanjem** kvocijenta **prema nuli**. Dakle, vrijedi:

---

$$3/2 = 1 \quad -3/2 = -1$$

---

Po **C90** standardu, to vrijedi za **pozitivne** operande. Inače, ako je bar jedan operand **negativan**, rezultat **ovisi o implementaciji**.

**Napomena.** Ako je **bar jedan** operand realan broj, dijeljenje je uobičajeno dijeljenje **realnih brojeva**, pa je  $3.0/2 = 1.5$ .

## Cjelobrojni ostatak (modulo)

Operator `%` (modulo) djeluje **samo** na **cjelobrojnim** operandima i kao rezultat daje

● **cjelobrojni ostatak** pri **cjelobrojnom** dijeljenju operanada.  
(operacija **mod** od ranije).

Primjer. Za  $x = 10$  i  $y = 3$  dobivamo

---

$$x / y = 3 \quad x \% y = 1$$

---

**Ostatak** se računa tako da uvijek **vrijedi** (osim za  $y == 0$ ):

---

$$(x / y) * y + x \% y == x$$

---

Dakle, **ostatak** ima predznak **prvog** operanda.

# Konverzije u aritmetičkim izrazima

Kad aritmetički operator ima dva operanda različitog tipa, onda dolazi do

- konverzije ili pretvaranja tipova.

U pravilu:

- prije operacije, operand “nižeg” ili “užeg” tipa se promovira, odnosno, pretvara u “viši” ili “širi” tip,

- zatim se izvršava operacija — sad na operandima istog tipa,

- rezultat operacije ima taj isti (zajednički) tip.

Ovo pretvaranje se radi onim redom kojim se izvršavaju operacije u izrazu, tj. po prioritetu operacija (v. kasnije).

# Konverzije u aritmetičkim izrazima (nastavak)

Na primjer:

- Operandi tipa `char` i `short` (s predznakom ili bez njega) **automatski** se pretvaraju u tip `int` ili `unsigned int`, i to **prije svake** aritmetičke operacije.
- Ako su operandi tipa `float` i `double`, onda se, **prije** izračunavanja,
  - `float` pretvara u `double`.
- Ako je jedan od operanada **realnog** tipa (`float`, `double`), a drugi **cjelobrojnog** (`int`, `unsigned`, `long`, ...), onda se
  - **cjelobrojni** tip pretvara u **realni**.

# Konverzije kod pridruživanja

U operaciji pridruživanja dolazi do konverzije, ako tip lijeve strane nije isti kao tip desne strane. Tada se:

- operand na desnoj strani konvertira u tip operanda na lijevoj strani.

Pri tome može doći do gubitka informacije,

- ako se širi tip konvertira u uži.

Najčešći primjer je pretvaranje realnog u cjelobrojni tip.

- To se radi “odbacivanjem”, tj. zaokruživanjem prema nuli.

Na primjer, ako je  $x$  varijabla tipa `float` i  $n$  varijabla tipa `int`, prilikom pridruživanja  $n = x$ , doći će do odsjecanja decimala u broju  $x$ .

# Operatori i izrazi (nastavak)

# Sadržaj

- **Operatori i izrazi** (drugi dio):
  - Unarni operatori inkrementiranja i dekrementiranja.
  - `sizeof` operator.
  - Relacijski operatori.
  - Logički operatori.
  - Skraćeno računanje logičkih izraza.
  - Operatori nad bitovima.
  - Operator pridruživanja i složeni operatori pridruživanja.
  - Uvjetni operator `? : .`
  - Operator zarez `,`.
  - Tablica prioriteta i asocijativnosti operatora.

# Operator inkrementiranja

Operator **inkrementiranja** `++` je **unarni** operator koji

• **povećava** vrijednost **varijable** za 1.

Operand **mora** biti **varijabla** (**ne smije** biti **izraz**) i mora biti nekog **brojevnog** ili kompatibilnog tipa (na pr. **pokazivač**).

Naredba

---

```
++x;
```

**ekvivalentna** je naredbi

---

```
x = x + 1;
```

Na ovom mjestu možemo koristiti i **postfiks** notaciju `x++`, s **istim** značenjem.



# Operator dekrementiranja

Operator **dekrementiranja** `--` je **unarni** operator koji

• **smanjuje** vrijednost **varijable** za 1.

Naredba

---

```
--x;
```

**ekvivalentna** je naredbi

---

```
x = x - 1;
```

Na ovom mjestu možemo koristiti i **postfiks** notaciju `x--`, s **istim** značenjem.

# Inkrementiranje i dekrementiranje (nastavak)

Razlika između **prefiks** i **postfiks** notacije ovih operatora pojavljuje se u **složenim** izrazima, s **više** operatora.

U **prefiks** notaciji (**++x**, **--x**)

- **prvo** se **promijeni** vrijednost varijable, a
- **onda** se ta vrijednost **iskoristi** u složenom izrazu.

U **postfiks** notaciji (**x++**, **x--**)

- **prvo** se (trenutna) vrijednost varijable **iskoristi** u složenom izrazu, a
- **onda** se **promijeni** vrijednost varijable.

# Inkrementiranje i dekrementiranje (nastavak)

Ova dva operatora (u obje forme) vrlo se često koriste za povećavanje ili smanjivanje brojača u petljama (na pr. `for`).

---

```
for (brojac = 0; brojac < n; ++brojac) ... ;  
for (brojac = n; brojac > 0; --brojac) ... ;
```

---

U ovakvim primjerima, potpuno je svejedno koju formu operatora koristimo — prefiks ili postfiks.

Inače, treba paziti na razliku između jedne i druge forme, tj. pažljivo izabrati koju formu stvarno želimo.

# Prefiks i postfiks forma — primjeri

## Primjer.

---

```
x = 3;  
y = ++x;    /* daje y = 4, x = 4 */  
y = x++;    /* daje y = 4, x = 5 */
```

---

## Primjer.

---

```
i = 7;  
printf("i = %d\n", --i); /* ispisuje i = 6 */  
printf("i = %d\n", i--); /* ispisuje i = 6 */  
printf("i = %d\n", i);   /* ispisuje i = 5 */
```

---

## Inkrementiranje i dekrementiranje (nastavak)

**Oprez.** Kod složenih izraza, **nema** pravila o tome **kad** se **mijenja** vrijednost varijable, obzirom na **računanje cijelog** izraza:

- prije (poslije) računanja **čitavog** izraza ili naredbe,
- **tik prije** (poslije) korištenja **trenutne** vrijednosti pojedine varijable u procesu računanja vrijednosti izraza.

To **nije** opasno ako se **varijabla** na koju djeluje **++** ili **--** pojavljuje samo **jednom** u izrazu ili naredbi.

Međutim, ako se takva **varijabla** javlja **više** puta, rezultat može biti **različit** na različitim C prevoditeljima (**Intel**, **Gnu**)!

Opasni primjeri (v. **KR2**, str. **53–54**):

•  $j = i++ + i++$ ,  $k = ++i + ++i$ ,  $a[i] = i++$ .

# Inkrementiranje i dekrementiranje (sažetak)

Operatori **inkrementiranja** i **dekrementiranja** pišu se u **prefiks** i **postfix** obliku.

**Značenje:** **++x** i **x++** znači “**povećaj x za 1**”, a **--x** i **x--** znači “**smanji x za 1**”.

**Primjer.** **Razlika** između **prefiksa** i **postfiksa** — **kad** se izvrši operacija inkrementiranja/dekrementiranja!

---

```
/* prefiks */
```

```
int x, z;
```

```
x = 3;
```

```
z = ++x - 2;
```

```
/* rezultati */
```

```
/* x = 4, z = 2 */
```

```
/* postfix */
```

```
int x, z;
```

```
x = 3;
```

```
z = x++ - 2;
```

```
/* x = 4, z = 1 */
```

---

# Operator sizeof

Operator `sizeof` vraća **veličinu** svog **operanda** u **bajtovima**.

- Operand može biti **izraz** (ne izračunava se) ili **tip podatka**, zatvoren u zagrade: **(tip)**.

Primjer.

```
int i;
double x;
printf("Velicina tipa int      = %u\n",
      sizeof i);    // moze i sizeof(i)
printf("Velicina tipa double = %u\n",
      sizeof x);    // moze i sizeof(x)
```

Na standardnim računalima rezultati su **4** i **8**.

Ovdje je operand **izraz** (sastavljen od jedne **varijable**).

## Operator sizeof (nastavak)

Isti efekt postizemo ako `sizeof` primijenimo na **tip podatka**:

---

```
printf("Velicina tipa int      = %u\n",
      sizeof(int));
printf("Velicina tipa double = %u\n",
      sizeof(double));
```

---

**Napomena.** Po “definiciji”, `sizeof(char)` je uvijek jednak **1**.

Kod **složenijih** tipova podataka dobivamo **ukupan** broj bajtova koji **podatak** (tog tipa) zauzima.

Na primjer, za **polje** dobivamo “**veličinu**” **cijelog** polja (ako smo u funkciji u kojoj je **rezervirana** memorija za polje, inače **ne** — v. kasnije).



# Operator sizeof (nastavak)

Primjer.

```
char tekst[] = "Program";  
printf("Broj znakova u varijabli tekst = %u\n",  
      sizeof(tekst));
```

ispisuje

```
Broj znakova u varijabli tekst = 8
```

Operator `sizeof` vraća cjelobrojnu vrijednost bez predznaka (zato `%u`) koja ovisi o implementaciji.

Pripadni tip definiran je u datoteci zaglavlja `<stddef.h>` i zove se `size_t` (tip za “veličine”, ili “duljine” objekata).

# Tablica prioriteta operatora (nepotpuna)

Operator `sizeof` je

• unarni operator, asocijativnost  $D \rightarrow L$ ,

i pripada istoj prioritetnoj grupi kao i ostali unarni operatori.

Kategorija	Operatori	Asoc.
unarni	<code>++</code> <code>--</code> <code>+</code> <code>-</code> <code>*</code> <code>&amp;</code> <code>(type)</code> <code>sizeof</code>	$D \rightarrow L$
aritm. mult.	<code>*</code> <code>/</code> <code>%</code>	$L \rightarrow D$
aritm. adit.	<code>+</code> <code>-</code>	$L \rightarrow D$
pridruživanje	<code>=</code>	$D \rightarrow L$

# Relacijski operatori

Jezik C ima šest relacijskih operatora:

  $<$  ,  $<=$  ,  $>$  ,  $>=$  ,  $==$  ,  $!=$  ,

podijeljenih u dvije grupe po prioritetu.

Standardni ili “uređajni” relacijski operatori su:

operator	značenje
$<$	strogo manje
$<=$	manje ili jednako
$>$	strogo veće
$>=$	veće ili jednako

Oni imaju isti prioritet, niži od prioriteta aritmetičkih i unarnih operatora.

# Operatori jednakosti

Operatori jednakosti su:

operator	značenje
<code>==</code>	jednako
<code>!=</code>	različito

Oni imaju zasebnu prioritetnu grupu s

- manjim prioritetom od standardnih relacijskih operatora.

Asocijativnost relacijskih operatora i operatora jednakosti je

- slijeva udesno,  $L \rightarrow D$ .

# Relacijski operatori — primjer

## Primjer.

---

```
int a = 1, b = 20, limit = 100;  
int rezultat;
```

```
rezultat = a < b;  
    /* rezultat = 1 (istina), jer 1 < 20 */
```

```
rezultat = a == b;  
    /* rezultat = 0 (laz), jer 1 != 20 */
```

```
rezultat = (a + 10) >= limit;  
    /* rezultat = 0 (laz), jer (1 + 10) < 100 */
```

---

Napomena. Ne smije se napisati  $\Rightarrow$  ili  $\Leftarrow$ !

# Tablica prioriteta operatora (nepotpuna)

Kategorija	Operatori	Asoc.
unarni	++ -- + - * & (type) sizeof	D → L
aritm. mult.	* / %	L → D
aritm. adit.	+ -	L → D
relacijski	< <= > >=	L → D
rel. jednakost	== !=	L → D
pridruživanje	=	D → L

# Logički izrazi

Relacijskim operatorima formiraju se tzv. **logički izrazi**.

Njihova vrijednost je

• **istina** (1) ili **laž** (0).

C90 nema poseban logički tip, pa je **vrijednost logičkih** izraza tipa **int**. (To vrijedi i u C11, iako postoji tip **\_Bool**.)

Primjer. Za  $i = 1$ ,  $j = 2$ ,  $k = 4$  imamo

izraz	istinitost	vrijednost
$i < j$	istinito	1
$(i + j) \geq k$	neistinito	0
$i == 2$	neistinito	0
$k != i$	istinito	1

## Logički izrazi (nastavak)

Prioritet relacijskih operatora **niži** je od prioriteta aritmetičkih operatora.

Zato je izraz

---

$$i \geq 'A' - 'a' + 1$$

---

ekvivalentan s

---

$$i \geq ('A' - 'a' + 1)$$

---

Prioritet je “podešen” tako

👉 da **zagrade ne treba** pisati,

iako je katkad korisno — za čitljivost, kao u  $(i + j) \geq k$ .



# Logički operatori

Složeniji logički izrazi tvore se pomoću logičkih operatora:

operator	značenje
!	logička negacija (unarna)
&&	logičko I
	logičko ILI

Operandi logičkih operatora su logičke vrijednosti (najčešće, logički izrazi), s tim da se

- svaka cjelobrojna vrijednost različita od nule interpretira kao istina,
- a samo nula se interpretira kao laž.

Vrijednost složenog logičkog izraza je 0 (laž) ili 1 (istina).

# Logički operatori (nastavak)

Svaki od logičkih operatora ima svoj poseban prioritet.

- Unarni operator **!** (logička negacija) spada u istu grupu kao i ostali unarni operatori.
  - Asocijativnost je, također,  $D \rightarrow L$ .
- Binarni operatori **&&** i **||** imaju niži prioritet od relacijskih i aritmetičkih operatora.
  - Asocijativnost je uobičajena, slijeva udesno,  $L \rightarrow D$ .
- Operator **&&** (logičko **i**, što odgovara množenju) ima viši prioritet od **||** (logičko **ili**, što odgovara zbrajanju).

# Logički operatori (nastavak)

Primjer. Ako je

• `i > 1` i `c == 't'` istinito, a

• `j < 6` lažno,

onda je:

izraz	istinitost	vrijednost
<code>i &gt; 1    j &lt; 6</code>	istinito	1
<code>i &gt; 1 &amp;&amp; j &lt; 6</code>	neistinito	0
<code>!(i &gt; 1)</code>	neistinito	0
<code>i &gt; 1 &amp;&amp; (j &lt; 6    c != 't')</code>	neistinito	0

Uočite da u tablici pišu samo one **zagrade** koje zaista **hoćemo** — za **promjenu** prioriteta.

# Logički operatori — primjer 1

## Primjer.

```
int a = 0, b = 10, c = 100, d = 200;
int rezultat;

rezultat = !(c < d);
    /* rezultat = 0, jer !(100 < 200) = !1
                                   = 0 */

rezultat = (a - b) && 1;
    /* rezultat = 1, jer -10 && 1 = 1 */

rezultat = d || b && a;
    /* rezultat = 1, jer 200 || (10 && 0)
                                   = 200 || 0 = 1 */
```

## Logički operatori — primjer 2

Primjer. Operator **negacije** **!** može se koristiti i ovako:

```
if (!zadovoljava) ...
```

što je ekvivalentno s

```
if (zadovoljava == 0) ...
```

Prvi oblik je **zgodan** za **logičke** testove (“ako ne zadovoljava, onda ...”).

Drugi oblik je **čitljiviji** za **numeričke** testove. Primjeri:

- 🔴 test parnosti: `if (n % 2 == 0)`, a **ne** `if (!(n % 2))`,
- 🔴 test neparnosti: `if (n % 2 == 1)`, a **ne** `if (n % 2)`.

## Logički operatori — primjer 3

Primjer. U C-u je dozvoljeno napisati izraz

---

$$30 \geq n \geq 10$$

---

Ako provjeravate je li  $n$  između 10 i 30, onda rezultat neće biti to što očekujete.

Za  $n = 20$ ,

- prvo se računa  $30 \geq 20$ , s rezultatom 1 (istina),
- a zatim se računa  $1 \geq 10$ , s rezultatom 0 (laž)!

Ispravno napisana provjera je

---

$$30 \geq n \ \&\& \ n \geq 10$$

---

# Tablica prioriteta operatora (nepotpuna)

Kategorija	Operatori	Asoc.
unarni	! ++ -- + - * & (type) sizeof	D → L
aritm. mult.	* / %	L → D
aritm. adit.	+ -	L → D
relacijski	< <= > >=	L → D
rel. jednakost	== !=	L → D
logičko I	&&	L → D
logičko ILI		L → D
pridruživanje	=	D → L

# Skraćeno računanje logičkih izraza — *standard*

Logički izrazi, koji se sastoje od pojedinačnih logičkih izraza povezanih **binarnim** operatorima `&&`, `||` — tj. izrazi oblika

`op1 && op2`, `op1 || op2`,

računaju se po sljedećim **pravilima**.

- **Prvo** računa **prvi** = **lijevi** operand `op1`, **prije** desnog `op2`, tj. poredak računanja **operanda** je “**slijeva nadesno**” ( $\rightarrow$ ).

**Važno.** U **C**-u se koristi tzv. **skraćeno** računanje takvih izraza.

- To znači da se izraz **prestaje** računati onog trena kad njegova **vrijednost** postane **poznata**.

**Posljedica pravila:** U navedenim izrazima, **prvo** se računa `op1` i

- `op1 && op2` — ako je `op1` **lažan**, `op2` se **ne računa**.
- `op1 || op2` — ako je `op1` **istinit**, `op2` se **ne računa**.



## Skraćeno računanje logičkih izraza — primjer

Primjer. Neka su  $a$ ,  $b$  i  $c$  varijable tipa `int`. U izrazu

$$x = a++ \ || \ b++ \ \&\& \ c++$$

operator `&&` ima **veći prioritet** od `||`, tj. prvo djeluje `&&`. No, `b++ && c++` je **desni** operand za `||`  $\implies$  **prvo** se računa `a++`.

- Ako je  $a \neq 0$  (**istina**), onda je sigurno  $x = 1$  (istina).
  - Drugi operand za `||`, tj. `b++ && c++` se **ne računa**.
  - Dakle,  $a$  se poveća za  $1$ , a  $b$  i  $c$  se **ne mijenjaju!**
- Ako je  $a == 0$  (**laž**), onda je  $x = b++ \ \&\& \ c++$  i računa se `b++`. Ako je  $b == 0$  (**laž**), onda je sigurno  $x = 0$  (laž), pa se `c++` **ne računa** i  $c$  se **ne mijenja!**

Dakle, cijeli izraz se računa **slijeva nadesno** ( $\rightarrow$ ), iako `&&` ima **veći prioritet** od `||`. Pogledajte slične primjere u **odatku**.

# Skraćeno računanje logičkih izraza — svrha

Skraćeno izračunavanje logičkih izraza se vrlo često koristi u programima (pretrage s 2 uvjeta). Pri tome treba biti oprezan

• kojim redom pišemo pojedine izraze, jer može doći do grešaka koje se relativno teško otkrivaju!

Primjer. U odsječku kôda (= “nađi znak različit od 'a'”),

```
char x[128];  
for (i = 0; i < 128 && x[i] != 'a'; ++i) {  
    ...  
}
```

za  $i = 128$  (na kraju petlje), neće doći do ispitivanja  $x[i] \neq 'a'$  — što je korektno, jer  $x[128]$  ne postoji!

# Skraćeno računanje logičkih izraza (nastavak)

Za **razliku** od prethodnog, kôd s **obratnim** poretком izraza

```
char x[128];          /* GRESKA */
for (i = 0; x[i] != 'a' && i < 128; ++i) {
    ...
}
```

nije korektan — za  $i = 128$ ,

• **prvo** ispituje je li  $x[128]$  različito od 'a',  
što je **greška** (“gazimo” po memoriji).

**Napomena.** Operatori  $\&\&$  i  $\|\|$  jesu **komutativni** — u smislu konačne **vrijednosti** izraza. Međutim, za **postupak** računanja,

• **poredak** pisanja operanada je **bitan!**

# Operatori nad bitovima

Operatori nad bitovima mogu se primijeniti na cjelobrojne tipove podataka `char`, `short`, `int`, `long`, ... (s predznakom ili bez njega), a djeluju na bitove u prikazu podatka:

operator	značenje
<code>~</code>	1–komplement (negacija bit-po-bit)
<code>&lt;&lt;</code>	lijevi pomak bitova
<code>&gt;&gt;</code>	desni pomak bitova
<code>&amp;</code>	logičko I bit-po-bit
<code>^</code>	ekskluzivno logičko ILI bit-po-bit
<code> </code>	logičko ILI bit-po-bit

Operator `~` je unarni, a svi ostali operatori su binarni.

Pazite na razliku između `&`, `|` (po bitovima) i `&&`, `||` (logički).

# Logički operatori nad bitovima

Operatori  $\&$ ,  $\wedge$  i  $\mid$  uzimaju dva operanda i vrše operacije na bitovima koji se nalaze na odgovarajućim mjestima.

Definicije operacija za odgovarajući par bitova dane su u sljedećoj tablici:

b1	b2	b1 & b2	b1 ^ b2	b1   b2
1	1	1	0	1
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Svaki ima svoj prioritet, redom:  $\&$ ,  $\wedge$ ,  $\mid$  — ispod relacijskih jednakosti, iznad logičkog  $\&\&$ . Asocijativnost je  $L \rightarrow D$ .

**Oprez!** C nema binarnih konstanti!

# Logički operatori nad bitovima (nastavak)

**Primjer.** Uzmimo da su **a** i **b** cjelobrojne varijable duljine 16 bitova (za lakše “crtanje”).

**Logičko I** (oba bita = 1):

$$\begin{array}{rcl} \mathbf{a} & = & \mathbf{0x0003} \quad /* = 0000\ 0000\ 0000\ 0011 */ \\ \mathbf{b} & = & \mathbf{0x0009} \quad /* = 0000\ 0000\ 0000\ 1001 */ \\ \hline \mathbf{a \ \& \ b} & = & \mathbf{0x0001} \quad /* = 0000\ 0000\ 0000\ 0001 */ \end{array}$$

**Logičko ILI** (bar jedan bit = 1):

$$\begin{array}{rcl} \mathbf{a} & = & \mathbf{0x0003} \quad /* = 0000\ 0000\ 0000\ 0011 */ \\ \mathbf{b} & = & \mathbf{0x0009} \quad /* = 0000\ 0000\ 0000\ 1001 */ \\ \hline \mathbf{a \ | \ b} & = & \mathbf{0x000b} \quad /* = 0000\ 0000\ 0000\ 1011 */ \end{array}$$

# Logički operatori nad bitovima (nastavak)

Ekskluzivno logičko ILI (točno jedan bit = 1):

$$a = 0x0003 \quad /* = 0000 \ 0000 \ 0000 \ 0011 \ */$$

$$b = 0x0009 \quad /* = 0000 \ 0000 \ 0000 \ 1001 \ */$$

---

$$a \wedge b = 0x000a \quad /* = 0000 \ 0000 \ 0000 \ 1010 \ */$$

---

# Logički operator 1–komplement

Unarni operator 1–komplement ( $\sim$ ) djeluje tako da jedinice u zapisu pretvara u nule i obratno, nule u jedinice ( $0 \leftrightarrow 1$ ).

Primjer. Neka je  $a$  cjelobrojna varijabla duljine 16 bitova (za lakše “crtanje”).

---

1–komplement (svaki bit u suprotni):

$$\begin{array}{l} a = 0x0c03 \quad /* = 0000\ 1100\ 0000\ 0011 \ */ \\ \hline \sim a = 0xf3fc \quad /* = 1111\ 0011\ 1111\ 1100 \ */ \end{array}$$

---



# Operatori pomaka

Operatori pomaka  $\ll$  i  $\gg$  pomiču binarni zapis broja **nalijevo** (**ulijevo**) ili **nadesno** (**udesno**).

- Operatori pomaka **nalijevo**  $\ll$  i **nadesno**  $\gg$  uzimaju **dva** operanda **cjelobrojnog tipa** (uz promociju kratkih tipova):
  - nad **prvim** operandom se vrši operacija i **rezultat** ima njegov **tip** (nakon eventualne početne promocije),
  - a **drugi** operand je **broj bitova** za koji treba izvršiti pomak (i njegov tip se prvo promovira, ako treba).
- Drugi operand **ne smije** biti **negativan** ili **premašiti broj bitova** u prvom operandu — inače rezultat **nije definiran!**

**Prioritet** operatora  $\ll$  i  $\gg$  je isti, **ispod** aritmetičkih aditivnih, **iznad** relacijskih. **Asocijativnost** je  $L \rightarrow D$ .

## Operator $\ll$ — množenje potencijom od 2

Operacija  $e1 \ll e2$  pomiče bitove od  $e1$  za  $e2$  mjesta ulijevo i to **ne ciklički**, tj.

- najznačajniji bitovi od  $e1$  se **gube**,
- a **zdesna** se dodaju **nule**.

Ideja operacije: **rezultat** = (cjelobrojno) **množenje**  $e1$  s  $2^{e2}$ .

To vrijedi ako (i samo ako) prvi operand  $e1$  ima cjelobrojni tip

- **bez predznaka** (tj. **unsigned ...**), ili
- **s predznakom**, vrijednost od  $e1$  je **nenegativna** i  $e1 \times 2^{e2}$  je **prikaziv** u tipu od  $e1$  (**s predznakom**).

U protivnom, rezultat je **nedefiniran**, tj. ovisi o implementaciji.

Razlog: vodeći bit rezultata određuje predznak ( $1 \ll 31$ ).

## Operator $\gg$ — dijeljenje potencijom od 2

Operacija  $e1 \gg e2$  pomiče bitove od  $e1$  za  $e2$  mjesta **udesno** i to **ne ciklički**, tj.

- najmanje značajni bitovi od  $e1$  se **gube**.

Ideja operacije: **rezultat** = cjelobrojno **dijeljenje**  $e1$  s  $2^{e2}$ .

To vrijedi ako (i samo ako) prvi operand  $e1$  ima cjelobrojni tip

- **bez predznaka** (tj. **unsigned ...**), ili

- **s predznakom** i vrijednost od  $e1$  je **nenegativna**.

Drugim riječima, **slijeva** se tada dodaju **nule**.

U protivnom ( **$e1$  negativan**), rezultat **ovisi** o implementaciji.

- **Većina slijeva** uvodi **bit predznaka (1)**, ostali uvode **nule**.

Ponašanje **većine**  $\approx$  **dijeljenje** s  $2^{e2}$  (ali  $-7 \gg 3 = -1$ , a **ne 0**).

## Operatori pomaka — primjeri

Primjer.  $b = a \ll 6$  radi sljedeće:

---

$$\begin{aligned} a &= 0x60ac \quad /* = 0110\ 0000\ 1010\ 1100 */ \\ a \ll 6 &= 0x2b00 \quad /* = 0010\ 1011\ 0000\ 0000 */ \end{aligned}$$

---

Svi bitovi pomiču se 6 mjesta **ulijevo**  $\iff$  množenje s  $2^6$ .

Primjer.  $b = a \gg 6$  radi sljedeće:

---

$$\begin{aligned} a &= 0x60ac \quad /* = 0110\ 0000\ 1010\ 1100 */ \\ a \gg 6 &= 0x0182 \quad /* = 0000\ 0001\ 1000\ 0010 */ \end{aligned}$$

---

Svi bitovi pomiču se 6 mjesta **udesno**  $\iff$  dijeljenje s  $2^6$ .

Ako je  $a$  cjelobrojnog tipa **s predznakom** i **negativan**, rezultat **ovisi** o implementaciji — slijeva se uvodi **bit predznaka** ili **nule**.

## Jednostavni primjer

Primjer. Ako je u programu definirano:

```
int x = 3;    /* ... 0011 */
int y = 5;    /* ... 0101 */
```

onda bitovni operatori daju sljedeće rezultate:

```
~x = -4      /* x + ~x + 1 = 0, v. ranije. */
x & y = 1    /* ... 0001 */
x | y = 7    /* ... 0111 */
x ^ y = 6    /* ... 0110 */
x << 2 = 12  /* ... 1100 */
y >> 2 = 1   /* ... 0001 */
```

# Maskiranje

Bitovni logički operatori najčešće služe tzv. “maskiranju” (prekrivanju ili filtriranju) pojedinih bitova u operandu:

- logičko I ( $\&$ ) služi postavljanju određenih bitova na 0 — “maska” ima 0 na tim mjestima,
- logičko ILI ( $|$ ) služi postavljanju određenih bitova na 1 — “maska” ima 1 na tim mjestima.

Ostali bitovi se ne mijenjaju, tj. “propuštaju se kroz filter”.

Slično tome,

- ekskluzivno ILI ( $\wedge$ ) služi nalaženju mjesta na kojima operandi imaju različite bitove,
- negacija ( $\sim$ ) služi “okretanju” bitova u suprotne.

## Maskiranje (nastavak)

**Primjer.** Postavimo deseti najmanje značajan bit u **a** na **nulu**.

Kad u varijabli **a** želimo postaviti neki bit na **nulu**, dovoljno je napraviti **logičko I** s varijablom **mask** koja na **tom** mjestu ima **nulu**, a na svim ostalim **jedinice**.

Varijablu **mask** najlakše je napraviti ovako:

- **krenemo** od broja **1** (**prvi** najmanje značajni bit je **1**),
- **pomaknemo** ga za **9** mjesta **ulijevo** (**deseti** najmanje značajni bit je **1**),
- dobiveni **rezultat** **1-komplementiramo** (suprotni bitovi).

---

```
unsigned mask = ~(1 << 9);    // jos bolje: 1u
a = a & mask;    /* a &= mask, v. malo kasnije */
```

---

## Maskiranje (nastavak)

Primjer. Šest najmanje značajnih bitova treba iz varijable **a** kopirati u **b**. Sve ostale bitove u **b** treba staviti na 1.

Prvo definiramo varijablu **mask** koja ima šest najmanje značajnih bitova jednakih 0, a ostale 1. Logički Ili s **mask** izdvaja tražene bitove u **a**, a sve ostale postavlja na 1.

---

```
mask = 0xfffffc0; /* = 1111 ... 1100 0000 */  
b = a | mask;
```

---

Ovako napisano, rezultat u **mask** ovisi o duljini tipa **int**.

To se može izbjeći korištenjem 1–komplementa.

---

```
mask = ~0x3f; /* = ~11 1111 */ // ~0x3fU
```

---



# Tablica prioriteta operatora (nepotpuna)

Kategorija	Operatori	Asoc.
unarni	! ~ ++ -- + - * & (type) sizeof	D → L
aritm. mult.	* / %	L → D
aritm. adit.	+ -	L → D
op. pomaka	<< >>	L → D
relacijski	< <= > >=	L → D
rel. jednakost	== !=	L → D
bitovni I	&	L → D
bitovni eks. ILI	^	L → D
bitovni ILI		L → D

## Tablica prioriteta operatora (nepotpuna)

Kategorija	Operatori	Asoc.
logičko I	&&	L → D
logičko ILI		L → D
pridruživanje	=	D → L

# Operatori pridruživanja

Osnovni operator pridruživanja je `=`. Prioritet mu je

niži od većine ostalih operatora (`,` je izuzetak).

To je zato da naredba pridruživanja, oblika

---

```
varijabla = izraz;
```

---

prvo izračuna `izraz` na desnoj strani, a onda tu vrijednost pridruži varijabli na lijevoj strani operatora.

Primjer.

---

```
i = 2;      j = 3 * i;
x = 3.17;   y = x + 5.342;
a = a + 1;  c = 'm';
```

---

# Operatori pridruživanja (nastavak)

Pridruživanje `varijabla = izraz` je, ujedno, i `izraz`.

- **Vrijednost** tog izraza je **vrijednost** varijable na lijevoj strani, **nakon** što se izvrši pridruživanje.

Stoga, pridruživanje može biti dio **složenijeg** izraza.

Primjer.

---

```
while ((a = x[i]) != 0) {  
    ...  
    ++i;  
}
```

---

U testu `while` naredbe, **prvo** se `x[i]` pridruži varijabli `a`, a **onda** se testira je li vrijednost tog izraza (a to je vrijednost varijable `a`) različita od nule. Zagrade su **nužne!**

## Operatori pridruživanja (nastavak)

Ovakva mogućnost **može** uzrokovati **teško** uočljive **greške**.

**Primjer.** Pogledajmo što će se dogoditi ako napišemo

---

```
if (varijabla = izraz) ...;
```

---

**umjesto** uobičajenog

---

```
if (varijabla == izraz) ...;
```

---

U prvoj **if** naredbi: **prvo** se vrijednost izraza **pridruži** varijabli, a **zatim** se **izvršava** tijelo **if** naredbe ako je varijabla **različita od nule**.

U drugoj **if** naredbi: **vrijednost** varijable se **ne mijenja**, već se samo **uspoređuje** s vrijednošću izraza. Tijelo **if** naredbe se **izvršava** ako su te dvije vrijednosti **jednake**.

# Operatori pridruživanja (nastavak)

Asocijativnost operatora pridruživanja je

- zdesna nalijevo,  $D \rightarrow L$ .

Zato operatore pridruživanja možemo **ulančati**:

---

```
varijabla_1 = varijabla_2 = ...  
              = varijabla_n = izraz;
```

---

To znači da se

- **izraz jednom** izračuna,
- a **zatim** se ta vrijednost, redom, **pridružuje** varijablama:  
**varijabla\_n, ..., varijabla\_2, varijabla\_1.**

# Operatori pridruživanja (nastavak)

Primjer.

---

$$x = y = \cos(3.22);$$

---

je ekvivalentno s

---

$$x = (y = \cos(3.22));$$

---

odnosno,

---

$$y = \cos(3.22);$$
$$x = y;$$

---

# Složeni operatori pridruživanja

Složeni operatori pridruživanja su:

•  $+=$ ,  $-=$ ,  $*=$ ,  $/=$ ,  $\%=$  (binarni aritmetički, pa =).

•  $<<=$ ,  $>>=$ ,  $\&=$ ,  $\^=$ ,  $\|=$  (binarni bitovni, pa =).

Općenito, izraz oblika

---

$$\text{izraz\_1 op= izraz\_2}$$

---

gdje je **op** jedna od **operacija**  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$ ,  $<<$ ,  $>>$ ,  $\&$ ,  $\^$ ,  $\|$ ,  
ekvivalentan je s

---

$$\text{izraz\_1} = \text{izraz\_1 op} (\text{izraz\_2})$$

---

Tj., **lijeva** strana **izraz\_1**, koji mora biti **varijabla**, je, ujedno, i **prvi** operand. Uočite **zagrade** oko **izraz\_2**.



## Složeni operatori pridruživanja (nastavak)

Ovi složeni operatori pridruživanja spadaju u istu prioritetnu grupu s osnovnim operatorom pridruživanja = i imaju istu asocijativnost zdesna nalijevo,  $D \rightarrow L$ .

Primjer.

Izraz	Ekvivalentan izraz
<code>i += 5</code>	<code>i = i + 5</code>
<code>i -= j</code>	<code>i = i - j</code>
<code>i *= j + 1</code>	<code>i = i * (j + 1)</code>
<code>i /= 4</code>	<code>i = i / 4</code>
<code>i %= 2</code>	<code>i = i % 2</code>
<code>i &lt;&lt;= 2</code>	<code>i = i &lt;&lt; 2</code>
<code>i &amp;= k</code>	<code>i = i &amp; k</code>

# Tablica prioriteta operatora (nepotpuna)

Kategorija	Operatori	Asoc.
unarni	! ~ ++ -- + - * & (type) sizeof	D → L
aritm. mult.	* / %	L → D
aritm. adit.	+ -	L → D
op. pomaka	<< >>	L → D
relacijski	< <= > >=	L → D
rel. jednakost	== !=	L → D
bitovni I	&	L → D
bitovni eks. ILI	^	L → D
bitovni ILI		L → D

# Tablica prioriteta operatora (nepotpuna)

Kategorija	Operatori	Asoc.
logičko I	&&	L → D
logičko ILI		L → D
pridruživanje	= += -= *= /= %= &= ^=  = <<= >>=	D → L

# Uvjetni operator ? :

Uvjetni izraz je izraz oblika

---

$$\text{izraz\_1} \text{ ? } \text{izraz\_2} \text{ : } \text{izraz\_3}$$

---

U njemu se **prvo** izračunava **izraz\_1**.

- Ako je on **istinit** (različit od nule), onda se izračunava **izraz\_2** i on postaje **vrijednost** čitavog uvjetnog izraza. U ovom slučaju, **izraz\_3** se **ne računa**.
- Ako je **izraz\_1** **lažan** (jednak nuli), onda se izračunava **izraz\_3** i on postaje **vrijednost** čitavog uvjetnog izraza. U ovom slučaju, **izraz\_2** se **ne računa**.

“**Paket**” od dva simbola “**? :**” je, zapravo, **ternarni** operator, tj. ima **3** operanda. **Prioritet** mu je **nizak** — odmah **iznad =**.

## Uvjetni operator ? : (nastavak)

Primjer. Izraz koji daje **manji** od brojeva **a** i **b**.

---

```
double a, b;  
...  
(a < b) ? a : b;
```

---

Vrijednost **uvjetnog** izraza može se pridružiti nekoj varijabli:

---

```
double a, b, min;  
...  
min = (a < b) ? a : b;
```

---

Korisno za razne **inicijalizacije!**

Zagrade ovdje **nisu** potrebne, zbog prioriteta (v. malo kasnije).

## Operator zarez ,

Operator **zarez ,** separira **dva izraza**. Izrazi separirani zarezom **izračunavaju se slijeva nadesno** ( $\rightarrow$ ) i rezultat čitavog izraza je vrijednost **desnog** izraza.

Primjer.

---

$$i = (i = 3, i + 4);$$

---

daje rezultat  **$i = 7$** . Operator zarez, **uglavnom**, se koristi u **for** naredbi (v. kasnije).

**Prioritet** operatora **,** je **niži** od operatora **pridruživanja**, tj. nalazi se **na dnu** tablice prioriteta. Zato su **nužne** zagrade na desnoj strani **prvog** operatora **=** u gornjem primjeru.

**Asocijativnost** je, naravno (po ideji), slijeva nadesno,  **$L \rightarrow D$** .

# Tablica prioriteta operatora (nepotpuna)

Uvjetni operator ima **nizak** prioritet, odmah **iznad** operatora pridruživanja, tako da zagrade oko prvog, drugog i trećeg izraza, najčešće, **nisu** potrebne.

Operator `,` ima **najniži** prioritet.

Kategorija	Operatori	Asoc.
unarni	! ~ ++ -- + - * & (type) sizeof	D → L
aritm. mult.	* / %	L → D
aritm. adit.	+ -	L → D
op. pomaka	<< >>	L → D
relacijski	< <= > >=	L → D
rel. jednakost	== !=	L → D

# Tablica prioriteta operatora (nepotpuna)

Kategorija	Operatori	Asoc.
bitovni I	&	L → D
bitovni eks. ILI	^	L → D
bitovni ILI		L → D
logičko I	&&	L → D
logičko ILI		L → D
uvjetni	? :	D → L
pridruživanje	= += -= *= /= %= &= ^=  = <<= >>=	D → L
operator zarez	,	L → D



# Tablica prioriteta operatora (potpuna)

Sad smo obradili skoro sve operatore u jeziku C.

Do potpune tablice operatora fale nam još samo tzv. primarni operatori. Ima ih 4:

- ( ) — za poziv funkcije,
- [ ] — za pristup elementima polja,
- . — za pristup članovima strukture,
- -> — za pristup članovima strukture preko pokazivača.

Obradit ćemo ih kasnije, kad navedeni “objekti” stignu na red.

Primarni operatori su grupa s najvišim prioritetom, a asocijativnost im je uobičajena,  $L \rightarrow D$ .

# Tablica prioriteta operatora (potpuna)

Kategorija	Operatori	Asoc.
primarni	() [] -> .	L → D
unarni	! ~ ++ -- + - * & (type) sizeof	D → L
aritm. mult.	* / %	L → D
aritm. adit.	+ -	L → D
op. pomaka	<< >>	L → D
relacijski	< <= > >=	L → D
rel. jednakost	== !=	L → D

# Tablica prioriteta operatora (potpuna)

Kategorija	Operatori	Asoc.
bitovni I	&	L → D
bitovni eks. ILI	^	L → D
bitovni ILI		L → D
logičko I	&&	L → D
logičko ILI		L → D
uvjetni	? :	D → L
pridruživanje	= += -= *= /= %=	D → L
	&= ^=  = <<= >>=	
operator zarez	,	L → D

# Dodatak — primjeri za operatore

# Skraćeno računanje logičkih izraza — primjer 1

Primjer 1. Računamo tablicu vrijednosti logičkih izraza

$$x = a++ \ || \ b++ \ \&\& \ c++ \quad y = b++ \ \&\& \ c++ \ || \ a++$$

Vrijednosti su iste (komutativnost  $\ || \$ ), a različito se računaju!

Nepromijenjena vrijednost  $\iff$  taj izraz se ne računa:

početni			$x$	novi			$y$	novi		
$a$	$b$	$c$		$a$	$b$	$c$		$a$	$b$	$c$
0	0	0	0	1	1	0	0	1	1	0
0	0	1	0	1	1	1	0	1	1	1
0	1	0	0	1	2	1	0	1	2	1
0	1	1	1	1	2	2	1	0	2	2
1	0	0	1	2	0	0	1	2	1	0
1	0	1	1	2	0	1	1	2	1	1
1	1	0	1	2	1	0	1	2	2	1
1	1	1	1	2	1	1	1	1	2	2

# Skraćeno računanje logičkih izraza — primjer 2

Primjer 2. Računamo tablicu vrijednosti logičkih izraza

$$x = a++ \ \&\& \ b++ \ || \ c++ \qquad y = b++ \ \&\& \ a++ \ || \ c++$$

Vrijednosti su iste (komutativnost  $\&\&$ ), a različito se računaju!

Nepromijenjena vrijednost  $\iff$  taj izraz se ne računa:

početni				novi				novi		
<i>a</i>	<i>b</i>	<i>c</i>	<i>x</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>y</i>	<i>a</i>	<i>b</i>	<i>c</i>
0	0	0	0	1	0	1	0	0	1	1
0	0	1	1	1	0	2	1	0	1	2
0	1	0	0	1	1	1	0	1	2	1
0	1	1	1	1	1	2	1	1	2	2
1	0	0	0	2	1	1	0	1	1	1
1	0	1	1	2	1	2	1	1	1	2
1	1	0	1	2	2	0	1	2	2	0
1	1	1	1	2	2	1	1	2	2	1

# Operatori — zadaća

**Primjer.** Koristi makro naredbu `PR(x)` za **ispis** parametra `x`.

```
#include <stdio.h>
#define PR(x) printf("%d\n", (x));

int main(void) {
    int x, y, z;
    x = -4 % 4 / 4 + -4;          PR(x);
    y = 4 / -x ++ -4;           PR(x); PR(y);
    y *= z = x + 4 == 4 / -y;   PR(y); PR(z);
    x = x || y && --z;           PR(x); PR(y); PR(z);
    PR(++x && ++y || ++z) ;     PR(x); PR(y); PR(z);
    return 0; }
```

**Ispis:** `-4`, `-3`, `-3`, `-3`, `1`, `1`, `-3`, `1` (ne 0), `1`, `2`, `-2`, `1` (ne 2).