Programiranje 1

Operatori i izrazi — drugi dio. Potpuna tablica prioriteta

Matej Mihelčić

Prirodoslovno-matematički fakultet
Matematički odsjek

13. studenoga 2023.

Operator inkrementiranja i dekrementiranja

Operator **inkrementiranja** (++) je **unarni** operator koji **povećava** vrijednost **varijable** za 1.

Operand **mora** biti **varijabla** (**ne smije** biti izraz) i mora biti **brojevnog** tipa (ili kompatibilan kao npr. znak, **pokazivač**).

`++x;` je ekvivalentno s `x = x+1;` (možemo koristiti i `x++;`).

Operator **dekrementiranja** (--) je **unarni** operator koji **smanjuje** vrijednost **varijable** za 1.
`--x;` je ekvivalentno s `x = x-1;` (možemo koristiti i `x--;`).

U **prefiks notaciji** (`++x`, `--x`): 1. **promijeni** vrijednost varijable, 2. koristi **izračunatu vrijednost** u složenijem izrazu.
U **postfiks notaciji** (`x++`,`x--`): 1. koristi **trenutnu vrijednost** u složenijem izrazu, 2. **promijeni** vrijednost varijable.

Operator inkrementiranja i dekrementiranja

Često se koriste za **povećavanje** ili **smanjivanje** brojača u petljama.

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

```
x = 4;
y = ++x; /* daje y = 5, x = 5 */
y = x++; /* daje y = 5, x = 6*/

i = 10;
printf("i␣=␣%d\n", --i); /* ispisuje i = 9 */
printf("i␣=␣%d\n", i--); /* ispisuje i = 9 */
printf("i␣=␣%d\n", i); /* ispisuje i = 8 */
```

Operator inkrementiranja i dekrementiranja

Kod **složenijih** izraza **nema** pravila o tome **kada** se mijenja vrijednost varijable obzirom na računanje cijelog izraza.

Nije opasno ukoliko se varijabla na koju djeluje ++ i -- pojavljuje samo **jednom** u izrazu.

Opasni primjeri izraza

```
j = i++ + i++ , k = ++i + ++i , a[i] = i++
```

Razlika između prefiksa i postfiksa

```
/* prefiks */
int x, z;
x = 3;
z = ++x - 2;
/* rezultati */
/* x = 4, z = 2 */
```

```
/* postfiks */
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 koji se **ne izračunava** ili **tip** zatvoren u zagrade `(tip)`.

Korištenje operatora sizeof

```
1 int i;
2 double x;
3 printf("Velicina tipa int = %u\n",
4 sizeof i); // moze i sizeof(i), sizeof(int)
5 printf("Velicina tipa double = %u\n",
6 sizeof x); // moze i sizeof(x), sizeof(double)
```

Najčešći rezultat je 4 i 8 bajtova.
`sizeof(char)` je uvijek 1. Kod složenijih tipova podataka dobivamo **ukupan** broj bajtova koji **podatak** tog tipa zauzima.

Operator sizeof

Korištenje operatora sizeof

```
1 char tekst[] = "Program";
2 printf("Broj␣znakova␣u␣varijabli␣tekst␣=␣%u\n",
3 sizeof(tekst));
4 /*Broj znakova u varijabli tekst = 8*/
```

Operator `sizeof` vraća cjelobrojnu vrijednost bez predznaka (zato koristimo format %u). Tip zadužen za reprezentaciju veličina ili duljina objekata se zove `size_t` iz zaglavlja `stddef.h`.

Operator `sizeof` je **unarni operator**, **asocijativnost** mu je $D \rightarrow L$, pripada istoj prioritetnoj grupi kao i ostali unarni operatori.

Tablica prioriteta operatora

Kategorija	Operatori	Asocijativnost
unarni	++ -- + - * & (type) sizeof	$D \rightarrow L$
aritm. mult.	* / %	$L \rightarrow D$
aritm. adit.	+ -	$L \rightarrow D$
pridruživanje	=	$D \rightarrow L$

Relacijski operatori

U C-u postoji **šest relacijskih** operatora:
`<`, `<=`, `>`, `>=`, `==`, `!=`. Podijeljeni su u **dvije** grupe po prioritetu.

1. **Standardni** ili **uređajni** operatori:
   < (strogo manje), <= (manje ili jednako), > (strogo veće), >= (veće ili jednako).
   Imaju međusobno isti prioritet, **niži** od prioriteta unarnih i numeričkih operatora.
2. Operatori **jednakosti**:
   == (jednako), != (različito)
   Imaju **manji** prioritet od relacijskih operatora.

Primjeri relacijskih operatora.

```
1 int a = 2, b = 15, c = 80;
2 int rezultat;
3 rezultat = a < b;
4 /* rezultat = 1 (istina), jer 2 < 15 */
5 rezultat = a == b;
6 /* rezultat = 0 (laz), jer 2 != 15 */
7 rezultat = (a + 10) >= c;
8 /* rezultat = 0 (laz), jer (2 + 10) < 80 */
```

Tablica prioriteta operatora

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

Logički izrazi i logički operatori

Relacijskim operatorima se formiraju **logički izrazi** koji poprimaju vrijednost 1 (**istina**) ili 0 (**laž**). Logičke vrijednosti su tipa `int`, iako u `C11` postoji tip `_Bool`.

Relacijski operatori imaju niži prioritet od aritmetičkih.

```
1  //primjer logickog izraza
2  i>='A' - 'a'+1
3  //ekvivalentno s
4  i>=('A'-'a'+1)
```

Složeniji logički izrazi tvore se pomoću **logičkih operatora**. `!` - logička negacija, `&&` - logičko I, `||` - logičko ILI.

Logički operatori kao parametre primaju **logičke vrijednosti** (najčešće rezultati izračunavanja logičkih izraza). Svaka vrijednost **različita od nula** interpretira se kao **istina** a **nula** se interpretira kao **laž**. **Rezultat** je logička vrijednost.

Tablica prioriteta operatora

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

Logički operatori - primjeri

Pretpostavimo:

- $i > 1$ i $c == 't'$ je istina.
- $j < 6$ je laž.

```
1 i > 1 || j < 6 //istinito (1)
2 i > 1 && j < 6 //neistinito (0)
3 !(i > 1) //neistinito (0)
4 i > 1 && (j < 6 || c != 't') //neistinito (0)
```

Zagrade koristimo ukoliko želimo **promijeniti prioritete izvođenja**. Npr. unarni operator `!` ima veći prioritet od relacijskog operatora >, stoga bi se izraz $!i > 1$ izvrijednio kao: $0 > 1$ što je neistina (0). Iako je konačni rezultat u ovom primjeru jednak, to je različito od $!(i > 1) = !1 = 0$. Isprobajte na primjeru $i = 2$, $!i > 4$, $!(i > 4)$.

Logički operatori - primjeri

```
1 int a = 0, b = 10, c = 50, d = 100;
2 int rezultat;
3
4 rezultat = !(c<d);
5 //rezultat = 0, !(50<100) = !1 = 0
6 rezultat = a-b && 1;
7 // rezultat = 1, -10 && 1 = 1
8 rezultat = d || b && a;
9 // rezultat = 1, 100 || 10 && a = 100 || 0 = 1
```

Operator negacije (!) se može koristiti i kao:
`if(!zadovoljava)`, ekvivalentno s `if(zadovoljava == 0)`.
`if(n%2 ==0)`, ekvivalentno s `if(!(n%2))`
`if(n%2!=0)`, ekvivalentno s `if(n%2)`, može se koristiti za testirati **neparnost broja** (`if(n%2 == 1)`).

Logički operatori - primjeri

Za varijablu `int n`, treba razlikovati: $30 >= n >= 10$ i logički izraz koji ispituje tvrdnju $n \in \{10, \ldots, 30\}$. Matematička tvrdnja se u C-u zapisuje kao $n >= 10$ && $n <= 30$.

Za $n = 20$, izraz $30 >= n >= 10$ se izvrijednjava kao $1 >= 10$ što je **laž**.

Skraćeno računanje logičkih izraza

Logički izrazi koji se sastoje od logičkih pod-izraza povezanih **binarnim** operatorima && i ||, op_1 && op_2, op_1 || op_2, računaju se po sljedećim pravilima:

- Prvo se izvrijednjava lijevi operand op_1 pa desni operand op_2. Poredak računanja je s lijeva na desno.
- Koristi se **skraćeno računanje** takvih izraza. Izraz se **prestaje** računati onog trenutka kada njegova **vrijednost** postane **poznata**.

Posljedica:

- op_1 && op_2 - ako je op_1 lažan, op_2 se ne računa.
- op_1 || op_2 - ako je op_1 istinit, op_2 se ne računa.

Skraćeno računanje logičkih izraza - primjer

Neka su a, b, c varijable tipa `int`. U izrazu $x = \ a++ \ || \ b++ \ \&\& \ c++$, operator && ima veći prioritet od ||. Stoga se izraz računa kao: $x = \ a++ \ || \ (b++ \ \&\& \ c++)$. Po pravilima izvrijednjavanja operatora ||, prvo računamo $a++$, a zatim $b++ \ \&\& \ c++$.

- Ako $a! = 0$ je **istina**, tada je sigurno $x = 1$.
  - $b++ \ \&\& \ c++$ se ne računa.
  - a se poveća za 1, b i c se ne mijenjaju.
- Ako je $a == 0$ (**laž**), tada $x = b++ \ \&\& \ c++$ i računa se $b++$.
  - Ako je $b == 0$ (**laž**), tada $x = 0$, c se ne mijenja.
  - Ako je $b! = 0$ (**istina**), računa se $c++$ i ako $c! = 0$ tada $x = 1$, inače $x = 0$.

Skraćeno računanje logičkih izraza

Skraćeno izračunavanje logičkih izraza se vrlo često koristi u programima, posebno ukoliko imamo pretrage s 2 uvjeta. **Bitno je kojim se redoslijedom pišu uvjeti!**

```
1 char x[130];
2 for (i = 0; i < 130 && x[i] != 'b'; ++i) { ... }
```

Kod uvjeta u gornjem primjeru za $i = 130$ neće doći do provjere $x[i]! = \text{'b'}$ zato što je $i < 130$ **laž**.

```
1 char x[130];
2 for (i = 0;  x[i] != 'b' && i < 130; ++i) { ... }
```

U gornjem primjeru dolazi do greške za $i = 130$, zato što $x[130]$ nije dio polja x (dohvaćamo memoriju izvan granica polja).

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:

- ~ - 1-komplement (negacija bit-po-bit).
- « - lijevi pomak bitova
- » - desni pomak bitova
- & - logičko I bit-po-bit
- ^ - ekskluzivno logičko ILI bit-po-bit
- | - logičko ILI bit-po-bit

Operator ~ je **unaran** a ostali operatori su **binarni**.

Razlikujete: operator & od operatora &&, te operator | od operatora ||.

Operatori nad bitovima

Operatori &, ^ i | uzimaju dva operanda (broja) i vrše operacije na bitovima koji se nalaze na odgovarajućim mjestima.

Neka su *a* i *b* cjelobrojne varijable duljine 16 bitova.

```
a      = 0x0005 /* = 0000 0000 0000 0101*/
b      = 0x0009 /* = 0000 0000 0000 1001*/
-------------------------------------
a & b = 0x0001 /* = 0000 0000 0000 0001*/

a      = 0x0005 /* = 0000 0000 0000 0101*/
b      = 0x0009 /* = 0000 0000 0000 1001*/
-------------------------------------
a | b = 0x000d /* = 0000 0000 0000 1101*/
```

Operatori nad bitovima

```
a      = 0x0005 /* = 0000 0000 0000 0101*/
b      = 0x0009 /* = 0000 0000 0000 1001*/
----------------------------------------
a ^ b = 0x000c /* = 0000 0000 0000 1100*/
```

Unarni operator 1-komplement (~) djeluje tako da **jedinice** u binarnom zapisu broja pretvara u **nule**, a **nule** pretvara u **jedinice** ($0 \leftrightarrow 1$).

Neka je *a* cjelobrojna varijabla duljine 16 bitova.

```
a  = 0x0005 /* = 0000 0000 0000 0101*/
----------------------------------------
~a = 0xfffa /* = 1111 1111 1111 1010*/
```

Operatori pomaka

Operatori pomaka (« , ») pomiču bitove binarnog zapisa broja **nalijevo** ili **nadesno**.

- Operatori pomaka **nalijevo** « i **nadesno** » primaju dva operanda **cjelobrojnog tipa** (uz promociju kratkih tipova):
  - nad **prvim** operandom se vrši operacija i rezultat je tipa tog operanda (nakon eventualne promocije),
  - **drugi** operand je **broj bitova** za koji treba izvršiti pomak (i njegov tip se promovira po potrebi).
- Drugi operand **ne smije** biti **negativan** ili **premašiti broj** bitova, inače rezultat nije definiran.

Operacija $e_1 << e_2$ pomiče bitove od e_1 za e_2 pozicije **ulijevo**.
Pomak **nije ciklički!**
Najznačajniji bitovi od e_1 se gube a **zdesna** se dodaju **nule**.
Rezultat operacije bi trebao biti množenje e_1 s 2^{e_2}.

Operatori pomaka

Prethodna tvrdnja vrijedi ako i samo ako prvi operand e_1 ima **cjelobrojni tip** i:

- **bez predznaka** je (`unsigned`)
- **s predznakom** je, vrijednost od e_1 je **nenegativna** i $e_1 \cdot 2^{e_2}$ je **prikaziv** u tipu od e_1 (**s predznakom**).

U protivnom je rezultat **nedefiniran** (ovisi o implementaciji). Razlog je što vodeći bit određuje predznak ($1 << 31$).

Operacija $e_1 >> e_2$ pomiče bitove od e_1 za e_2 pozicije **udesno** (<u>**ne ciklički**</u>). **Najmanje značajni** bitovi od e_1 se gube a **s lijeva** se dodaju **nule**. Prethodna tvrdnja vrijedi ako i samo ako prvi operand e_1 ima **cjelobrojni tip** i:

- **bez predznaka** je (`unsigned`)
- **s predznakom** je i vrijednost od e_1 je **nenegativna**.

Ukoliko je e_1 negativan, rezultat **ovisi o implementaciji**.

Operatori pomaka

Kod većine implementacija se kod negativnog e_1 s lijeva uvodi **bit predznaka** (1), ostali uvode **nule**. Rezultat operacije bi **uglavnom** trebao biti dijeljenje e_1 s 2^{e_2} (razlika: $-7 >> 3 = -1$ a **ne** 0).

```
a    = 0x60ac /* = 0110 0000 1010 1100*/
a<<6 = 0x2b00 /* = 0010 1011 0000 0000*/

a    = 0x60ac /* = 0110 0000 1010 1100*/
a>>6 = 0x0182 /* = 0000 0001 1000 0010*/
```

Operatori pomaka

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

~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 se često koriste za tzv. *maskiranje* (prekrivanje ili filtriranje) pojedinih bitova u operandu.

- logičko I (`&`) se može koristiti za **postavljanje** određenih bitova na 0 (*maska* ima vrijednost 0 na tim pozicijama),
- logičko ILI (`|`) se može koristiti za **postavljanje** određenih bitova na 1 (*maska* ima 1 na tim pozicijama).

Ostali bitovi se **ne mijenjaju**, tj. *propuštaju se kroz filter*.

Ekskluzivno ILI (`^`) se može koristiti za **nalaženje** mjesta na kojima operandi imaju različite bitove, negacija (`~`) se koristi za komplementiranje vrijednosti bitova ($1 \leftrightarrow 0$).

Maskiranje

Primjer: Postavite **deseti najmanje** značajan bit **cijelog nenegativnog** broja *a* na **nulu**.

- Konstruiramo varijablu **mask** koja na desetoj poziciji **s desna** ima vrijednost 0, a na ostalim pozicijama ima vrijednost 1. **mask** kreiramo tako da:
  - **krenemo** od broja 1 (`int` ili `unsigned int` konstanta). **Prvi** najmanje značajan bit je 1.
  - **pomaknemo** ga za devet mjesta **ulijevo** (deseti najmanje značajan bit je 1).
  - dobiveni rezutat 1-komplementiramo.
- primjenimo bit-po-bit I (`&`) između *a* i **mask**.

```
unsigned mask = ~(1 << 9); // bolje: 1u
a = a & mask; /* a &= mask */
```

Maskiranje

Primjer: Kopirajte šest **najmanje značajnih** bitova iz varijable *a* u varijablu *b*.

- Definiramo varijablu **mask** koja ima **šest najmanje značajnih** bitova jednakih 0, ostale 1.
- Bit-po-bit ILI (|) s **mask izdvaja** tražene bitove od *a*, ostale postavlja na 1.

```
mask = 0xffffffc0; /* = 1111 ... 1100 0000 */
b = a | mask;
//ovisno o duljini cjelobrojnog tipa varijable a

mask = ~0x3f; /* = ~11 1111 */ // ~0x3fU
//neovisno o duljini cjelobrojnog tipa varijable a
```

Tablica prioriteta operatora

Kategorija	Operatori	Asocijativnost
unarni	! ~ ++ -- + - * & (type) sizeof	$D \rightarrow L$
aritm. mult.	* / %	$L \rightarrow D$
aritm. adit.	+ -	$L \rightarrow D$
op. pomaka	« »	$L \rightarrow D$
relacijski	< <= > >=	$L \rightarrow D$
rel. jednakost	== !=	$L \rightarrow D$
bit-po-bit I	&	$L \rightarrow D$
bit-po-bit eks. ILI	^	$L \rightarrow D$
bit-po-bit ILI	|	$L \rightarrow D$
logičko I	&&	$L \rightarrow D$
logičko ILI	||	$L \rightarrow D$
pridruživanje	=	$D \rightarrow L$

Operatori pridruživanja

Osnovni operator pridruživanja je =. Ima niži prioritet od **svih operatora** osim operatora ,. **Razlog:** da bi se prvo izvrijednio `izraz`, a nakon toga se vrijednost pridružila varijabli `varijabla`.

```
varijabla = izraz;
```

Primjeri:

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

Pridruživanje `varijabla = izraz` je također izraz i vrijednost nakon izvrednjavanja tog izraza je **vrijednost** varijable na lijevoj strani. Pridruživanje **može biti dio složenijeg izraza**.

Operatori pridruživanja

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

U uvjetu `while` naredbe se prvo $x[i]$ pridruži varijabli a, onda se testira je li vrijednost tog izraza (vrijednost varijable a) različita od nula. Zagrade su **nužne** zbog prioriteta izvođenja operatora.

Pripaziti na:

```
if (varijabla = izraz) ...;
//vrijednost izraza se pridruzi varijabli pa se
//ispita je li ta vrijednost razlicita od nula
if (varijabla == izraz) ...;
//provjerava se ima li varijabla jednaku
//vrijednost  vrijednosti  izraza s desne strane
```

Operatori pridruživanja

Operatore pridruživanja možemo **ulančati**.

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

Izraz se računa **jednom** a zatim se ta vrijednost **pridružuje** varijablama `varijabla_n, ..., varijabla_1`.

```
x = y = cos(3.22)
//ekvivalentno kao
x = (y = cos(3.22));
//ekvivalentno kao
y = cos(3.22);
x = y;
```

Složeni operatori pridruživanja

Složeni operatori pridruživanja su:
`+=`, `-=`, `*=`, `/=`, `%=` (binarni aritmetički nakon kojih slijedi =),
`«=`,`»=`,`&=`,`^=` ,`|=` (binarni bitovni nakon kojih slijedi =).

Izraz oblika `izraz_1 op= izraz_2`, za
op $\in \{+, -, *, /, \%, <<, >>, \&, \hat{}, |\}$ ekvivalentan je s `izraz_1 = izraz_1 op (izraz_2)`. Prvi operand je `izraz_1` koji mora biti varijabla. Prvo trebamo izvrijedniti `izraz_2`.

Izraz	Ekvivalentan izraz
`i += 10`	`i = i + 10`
`i -= j`	`i = i - j`
`i *= j + 4`	`i = i * (j + 4)`
`i /= 2`	`i = i / 2`
`i %= 6`	`i = i % 6`
`i <<= 3`	`i = i << 3`
`i &= k`	`i = i & k`

Tablica prioriteta

Kategorija	Operatori	Asocijativnost
unarni	! ~ ++ -- + - * & (type) sizeof	$D \to L$
aritm. mult.	* / %	$L \to D$
aritm. adit.	+ -	$L \to D$
op. pomaka	« »	$L \to D$
relacijski	< <= > >=	$L \to D$
rel. jednakost	== !=	$L \to D$
bit-po-bit I	&	$L \to D$
bit-po-bit eks. ILI	^	$L \to D$
bit-po-bit ILI	|	$L \to D$
logičko I	&&	$L \to D$
logičko ILI	||	$L \to D$
pridruživanje	= += -= *= /= %= &= ^= |= «= »=	$D \to L$

Uvjetni operator ? :

Uvjetni izraz je izraz oblika: `izraz_1 ? izraz_2 : 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**.

Uvjetni operator je **ternarni** operator i **vraća vrijednost**.

```
double a, b;
//izraz koji vraca manji broj od a,b
(a < b) ? a : b;
//vrijednost mozemo pridruziti varijabli
min = (a < b) ? a : b
```

Operator zarez ,

Operator **zarez ,** separira **dva izraza**. Izrazi separirani zarezom se izvrijednjavaju s lijeva nadesno i rezultat čitavog izraza je vrijednost **desnog** izraza.

```
i = (i = 3, i + 4);
//i = 7
```

Operator **zarez** se uglavnom koristi u `for` naredbi. Ima najniži prioritet, zato su zagrade iz primjera **nužne**.

Tablica prioriteta

Kategorija	Operatori	Asocijativnost
unarni	! ~ ++ -- + - * & (type) sizeof	$D \to L$
aritm. mult.	* / %	$L \to D$
aritm. adit.	+ -	$L \to D$
op. pomaka	« »	$L \to D$
relacijski	< <= > >=	$L \to D$
rel. jednakost	== !=	$L \to D$
bit-po-bit I	&	$L \to D$
bit-po-bit eks. ILI	^	$L \to D$
bit-po-bit ILI	|	$L \to D$
logičko I	&&	$L \to D$
logičko ILI	||	$L \to D$
uvjetni	? :	$D \to L$
pridruživanje	= += -= *= /= %= &= ^= |= «= »=	$D \to L$

Tablica prioriteta

Kategorija	Operatori	Asocijativnost
operator zarez	,	$L \rightarrow D$

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**.

Do potpune tablice operatora još fale 4 operatora:

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

Obradit ćemo ih kasnije, kada ćemo govoriti o **složenijim** objektima.

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

Tablica prioriteta

Kategorija	Operatori	Asocijativnost
primarni	() [] -> .	$L \rightarrow D$
unarni	! ~ ++ -- + - * & (type) sizeof	$D \rightarrow L$
aritm. mult.	* / %	$L \rightarrow D$
aritm. adit.	+ -	$L \rightarrow D$
op. pomaka	« »	$L \rightarrow D$
relacijski	< <= > >=	$L \rightarrow D$
rel. jednakost	== !=	$L \rightarrow D$
bit-po-bit I	&	$L \rightarrow D$
bit-po-bit eks. ILI	^	$L \rightarrow D$
bit-po-bit ILI	|	$L \rightarrow D$
logičko I	&&	$L \rightarrow D$
logičko ILI	||	$L \rightarrow D$

Tablica prioriteta

Kategorija	Operatori	Asocijativnost
uvjetni	? :	$D \rightarrow L$
pridruživanje	= += -= *= /= %= &= ^= |= «= »=	$D \rightarrow L$
operator zarez	,	$L \rightarrow D$