



PROGRAMIRANJE I

P-04: Uvod u programski jezik C

prof. dr **Dražen Brđanin**
2023/24



P-04: Uvod u programski jezik C

■ Sadržaj predavanja

- istorijat i standardizacija
- struktura programa u jeziku C
- pretprocesorske direktive
- standardni izlaz i ulaz
- tipovi podataka
- promjenljive i konstante
- operatori
- konverzije tipova



Uvod u programski jezik C

O programskom jeziku C

- **Nastao 70-ih godina prošlog vijeka**
 - 1970. Ken Thompson, "B"
 - **1972. Denis Ritchie, "C"**
- **Zahtjevi koje je trebao da zadovolji:**
 - **viši programski jezik sa mogućnostima mašinskog jezika** (direktan pristup memoriji, veći skup operatora, ...)
 - prednosti: udobnost višeg programskog jezika, moć mašinskog jezika
 - **programski jezik sa malim skupom naredbi i bogatom bibliotekom funkcija**
 - prednosti: jednostavnost programiranja i pojednostavljeni prevodioci

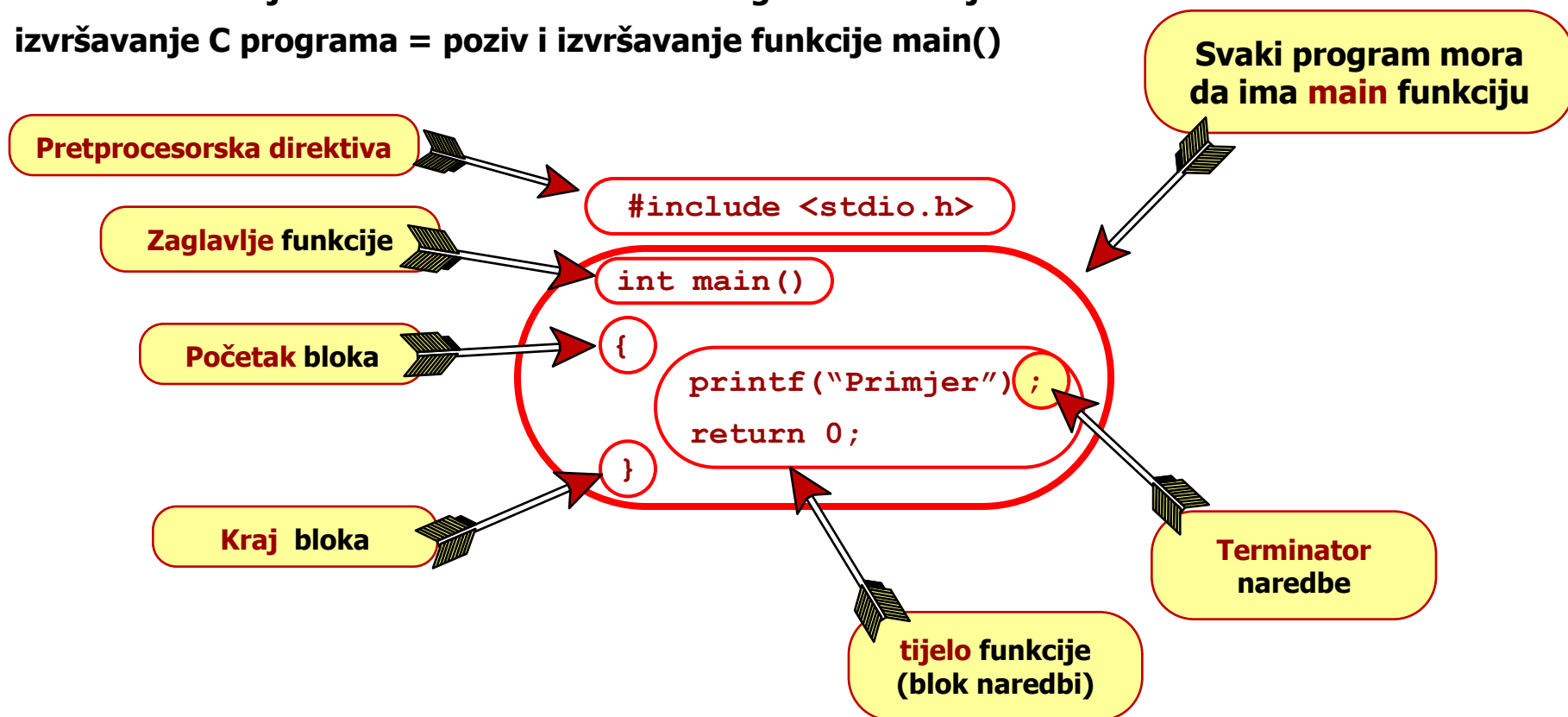
Standardizacija jezika C

- **K&R C (1978)**
 - B. Kerningen i D. Riči objavili 1978. prvo izdanje knjige "The C Programming Language" – godinama bila neformalna specifikacija jezika (poznata kao K&R ili white book)
 - K&R je dugo vremena služio kao standard, jer su uglavnom sve K&R elemente podržavali svi C prevodioci
- **ANSI C (C89) = ISO C (C90)**
 - 1989: ANSI objavio standard X3.159-1989 „Programming Language C“ – ANSI C ili C89.
 - 1990: ISO usvojila isti dokument pod oznakom ISO/IEC 9899:1990 – C90, pa C89=C90 (isti jezik)
 - C89/C90 predstavlja nadskup K&R jezika C
- **C99**
 - 1999: usvojen novi standard ISO/IEC 9899:1999, poznat kao C99, proširenje novim elementima
- **C11**
 - 2011: usvojen novi standard ISO/IEC 9899:2011, proširenje novim elementima

Struktura programa u jeziku C

Struktura programa u jeziku C

- Program u programskom jeziku C čini određen broj funkcija
- Jedna od funkcija obavezno se naziva main – glavna funkcija
- izvršavanje C programa = poziv i izvršavanje funkcije main()



Pretprocesorske direktive

Pretprocesorske direktive

- Obrađuje se prije prevođenja programa
- Obavezno počinje sa #



```
#include <stdio.h>
#define PI 3.14
```

```
int main()
{
    printf("pi=%5.2f", PI);
    return 0;
}
```

#include – najčešće korištena direktiva
uključuje u program sadržaj datoteke koja
se navodi kao parametar

```
#include <ime>
ili
#include "ime"
```

Standardna biblioteka

<stdio.h> funkcije za standardni I/O
<math.h> matematičke funkcije
...

#define – često korištena direktiva
definiše **simboličke konstante i makroe**

```
#define IME vrijednost
#define KVAD(x) (x)*(x)
```

Primjeri definicije simboličkih konstanti:

```
#define PI 3.14159
#define EPS 1e-6
#define MAXINT 32767
#define K_OCT 071
#define K_HEX 0x41
```

Standardni izlaz

Standardni izlaz

- Ispis podataka na standardni izlaz (monitor)
- U različitim programskim jezicima na različite načine (naredbe, funkcije, objekti, ...)
- U jeziku C, standardni izlaz se realizuje funkcijom **printf()** iz standardne biblioteke (<stdio.h>)

prototip funkcije:

```
int printf(const char* konverzioniString, ...);
```

poziv funkcije:

```
printf("konverzioni string", lista_argumenata);
```

Kontrolni (konverzioni) niz određuje način na koji se vrši ispis

Štampa se sve što se nađe unutar " " sve dok se ne dođe do znaka %

Kad se dođe do znaka % gleda se sljedeći znak (grupa znakova) koji se naziva **konverzioni karakter**

Konverzioni karakter definiše kako će se ispisati odgovarajući podatak iz liste argumenata

Mora biti onoliko znakova % u cijelom stringu koliko ima i argumenata u listi

Određuje šta treba da se ispiše (koji podaci treba da se ispišu)

Argumenti se odvajaju zapetama

Argumenti mogu da budu **promjenljive, konstante, izrazi**

Standardni izlaz

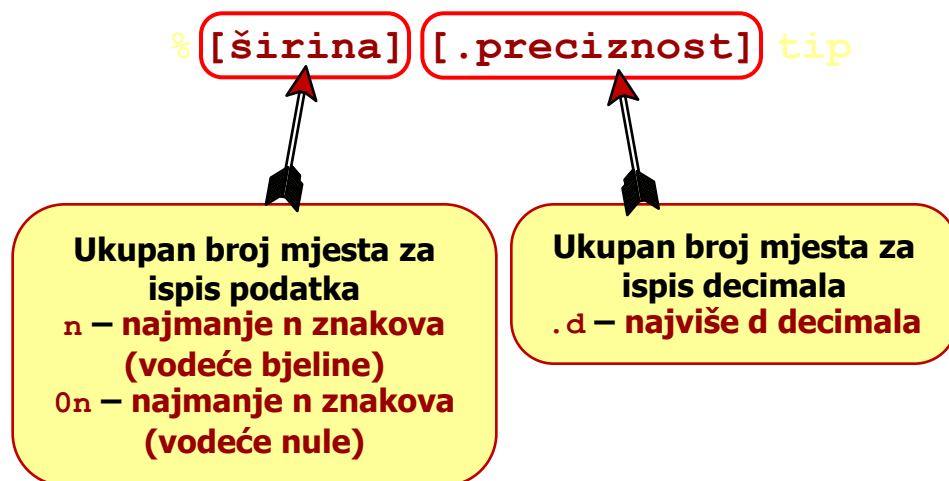
Standardni izlaz – konverzioni karakteri

%c	– znak	%ld	– long integer
%s	– string	%f	– float
%d	– integer (cijeli broj)	%lf	– double
%hd	– short integer	%o	– oktalni podatak
%u	– unsigned integer	%x	– heksadecimalni podatak
%hu	– unsigned short	%%	– ispis znaka %

Specijalne sekvence

\n	– novi red
\b	– jedno mjesto unazad
\a	– zvučni signal (beeper)
\f	– novi ekran (briše ekran)
\r	– vraćanje na početak reda
\t	– tabulator udesno
\'	– ispisuje znak '
\"	– ispisuje znak "
\\	– ispisuje znak \
\ddd	– ispisuje znak sa kodom <i>ddd</i> oktalno
\xdd	– ispisuje znak sa kodom <i>dd</i> heksadecimalno

Opšti oblik:



Standardni ulaz

Standardni ulaz

- Učitavanje podataka sa standardnog ulaza (tastatura)
- U različitim programskim jezicima na različite načine (naredbe, funkcije, objekti, ...)
- U jeziku C, standardni ulaz se realizuje funkcijom **scanf()** iz standardne biblioteke (<stdio.h>)

prototip funkcije:

```
int scanf(const char* konverzioniString, ...);
```

poziv funkcije:

```
scanf("konverzioni string", lista_argumenata);
```

Kontrolni (konverzioni) niz određuje način na koji se vrši unos podataka

Sličan konverzionom nizu kod `printf()`

Čine ga konverzioni karakteri koji definišu način unosa podataka

Određuje šta treba da se unese (koji podaci treba da se učitaju)

Argumenti se odvajaju zapetama

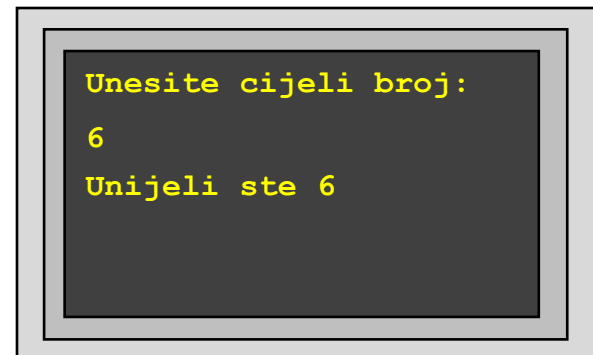
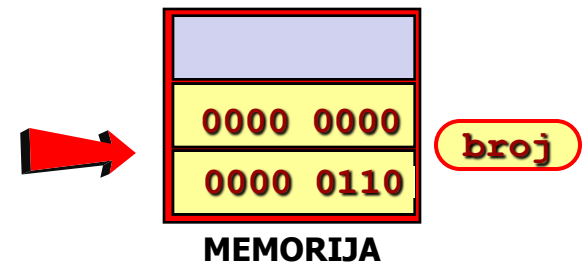
Argumenti su pokazivači na memorijske lokacije u koje se smještaju učitani podaci

Ako se učitava promjenljiva `x` tada je argument oblika `&x`

Standardni ulaz

Primjer:

```
#include <stdio.h>
int main()
{
    short int broj;
    printf( "Unesite cijeli broj:\n" );
    scanf( "%hd", &broj );
    printf( "Unijeli ste %hd\n", broj );
    return 0;
}
```





Promjenljive

Promjenljive

- Pored naredbi, promjenljive su osnovni elementi u imperativnoj paradigmi
- **Promjenljiva (varijabla) je imenovana memorijska lokacija namijenjena za privremeno skladištenje podataka**
- Promjenljive su tipično **smještene u memoriji (memorijske promjenljive)**, a izuzetno mogu da budu **smještene u registrima procesora (registarske promjenljive)**
- U zavisnosti od toga da li promjenljiva tokom svog životnog vijeka:
 - **služi za smještanje podataka isključivo jednog tipa, koji se ne mijenja tokom životnog vijeka – statički tipizirani jezici**
 - **služi za smještanje podataka različitih tipova (može da mijenja tip) – dinamički tipizirani jezici**
 - **C je statički tipiziran jezik**

➤ Imenovanje promjenljivih

- **ime promjenljive je identifikator (svaka promjenljiva ima jedinstveno ime u nekom bloku)**
- identifikator može da sadrži slova i cifre, kao i simbol `_`; ne može počinjati cifrom; ključne riječi jezika C (npr. `if`, `for`, `while`) ne mogu se koristiti kao identifikatori
- velika i mala slova se razlikuju, npr. promjenljive sa imenima `a` i `A` su dvije različite promjenljive
- uobičajeno malim slovima počinju imena promjenljivih i funkcija, a velikim slovima imena simboličkih konstanti
- imena promjenljivih treba da oslikavaju njihovo značenje u programu, ali za promjenljive kao što su indeksi obično se koriste kratka imena (npr. `i`)
- ukoliko ime promjenljive sadrži više riječi:
 - razdvajanje simbolom `_` (npr. `broj_dana`)
 - kamilja notacija (CamelCase) (npr. `brojDana`)
- različite konvencije za imenovanje promjenljivih, npr. mađarska notacija – početna slova imena promjenljivih predstavljaju kratku oznaku tipa te promjenljive (na primjer, `iBrojStudenata`)
- ANSI C – garantuje da se barem početnih 31 slovo imena smatra značajnim, a kod C99 to je 63

Promjenljive

Deklaracija (definicija) promjenljive

- Neki programski jezici ne zahtijevaju eksplicitnu deklaraciju promjenljivih
- **Jezik C zahtijeva eksplicitnu deklaraciju**
 - svaka promjenljiva mora biti eksplicitno deklarirana (tipično na početku bloka)
- Prevodiocu (kompajleru) mora se ukazati na to da će se u programu koristiti neka promjenljiva, kako bi on mogao da rezerviše potreban prostor u memoriji

Svaki izraz oblika
Tip imePromjeljive;
predstavlja deklaraciju, ali ako će na osnovu tog iskaza biti i rezervisan prostor u memoriji, tada govorimo o definiciji

Početna vrijednost
početna vrijednost ne mora se dodjeljivati
[] se koriste da bi se naglasilo da nije obavezno

Opšti oblik definicije promjenljive:

Tip imePromjenljive **[=vrijednost]** ;

Npr.

```
int i;           // definicija bez inicijalizacije
int i,j;         // definicija bez inicijalizacije
float a, e=2.71; // definicija sa inicijalizacijom
char znak='+' ,c; // definicija sa inicijalizacijom
```

Konstante

Definicija konstante

- Osim simboličkih konstanti

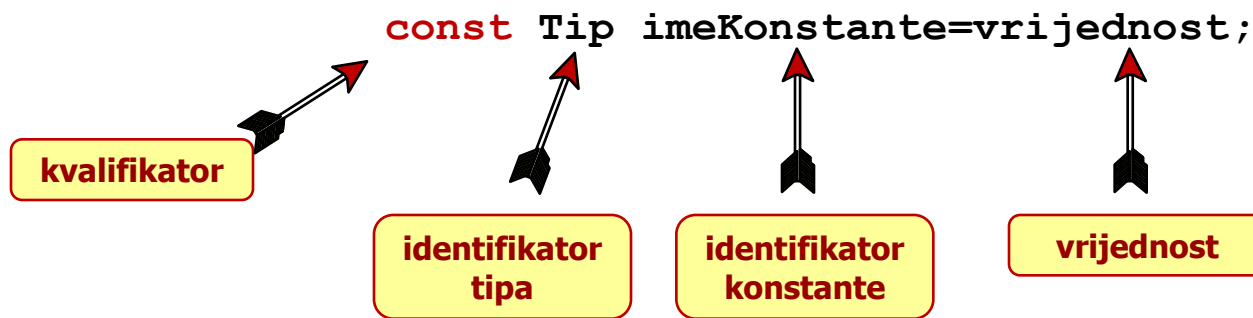
#define PI 3.14

moгуće je definisati konstante

(read only promjenljive)

**Vrijednost konstante ne može se mijenjati
(read only)**

Opšti oblik definicije konstante:



Npr.

```
const float pi=3.14159;
```

```
const int dim=100;
```



Tipovi podataka

Tipovi podataka

- Svaki podatak koji se koristi u programu je nekog tipa
- **Svaki tip podataka karakterišu:**
 - vrsta podataka koje opisuje
 - način reprezentacije i format
 - dozvoljene operacije
- S obzirom na to da li dati tip već postoji u jeziku ili ga mora definisati programer, razlikujemo:
 - **standardni (ugrađen u jezik, built-in)**
 - **korisnički definisan**
- S obzirom na to da li dati tip omogućava reprezentaciju jednostavnijih podataka (npr. Jedna cjelobrojna vrijednost) ili složenih podataka (npr. niz vrijednosti), tip podataka može biti:
 - **prosti**
 - **složeni**

Prosti ugrađeni tipovi podataka u C-u

Cjelobrojni

char	(1 bajt) koristi se i za znakove
short	(2 bajta)
int	(4 bajta)
long	(4 bajta)
long long	(8 bajtova)

Realni

float	(4 bajta)
double	(8 bajtova)

Kvalifikatori za cjelobrojne podatke

signed	- označeni
ako se ne navede podrazumijeva se za sve cjelobrojne tipove (osim za char)	
signed char	(-128..+127)
signed short	(-32768..+32767)
unsigned	- neoznačeni
unsigned char	(0..255)
unsigned short	(0..65535)

Logički

starije verzije standarda – nema poseban tip
ISTINA: svaka nenulta vrijednost
LAŽ: nula
novije verzije standarda – tip bool



Operatori


Operatori

- **reprezentuju operacije nad podacima**
- **s obzirom na broj podataka (operanada) nad kojima se primjenjuju**, operator može biti:
 - **unarni** – primjenjuje se nad jednim operandom
 - **binarni** – primjenjuje se nad dva operanda
 - **ternarni** – primjenjuje se nad tri operanda
- **s obzirom na poziciju u odnosu na operande unutar izraza**, operator može biti:
 - **prefiksni** – navodi se prije operanda
npr. ++i
 - **postfiksni** – navodi se poslije operanda
npr. i++
 - **infiksni** – navodi se između operanda
npr. a+b
- unarni operatori mogu biti prefiksni ili postfiksni, a binarni su tipično infiksni
- **jezik C ima veoma bogat skup operatora**

Prioritet operatora

- **operatori imaju prioritet** (kao i u matematici)
- **osnovna pravila** za određivanje prioriteta:
 - **male zagrade ()** omogućavaju grupisanje operatora unutar izraza i imaju **najviši prioritet**
 - **unarni operatori** imaju **prioritet nad binarnim operatorima**
 - **postfiksni unarni operatori** imaju **prioritet u odnosu na prefiksne**
 - **aritmetički operatori** imaju prioritet u odnosu na operatore poređenja, a **operatori poređenja** imaju prioritet u odnosu na **logičke operatore**
 - **operatori dodjele** imaju **veoma nizak prioritet**
- **prioritet operatora je standardizovan i ne može da se mijenja**, tipično prikazan u **tabeli prioriteta operatora**

Operatori

Prioritet	Operatori	Asocijativnost
Najviši  Najniži	++ (postf.) -- (postf.) () [] . ->	→
	++ (pref.) -- (pref.) + - ! ~ (type) * & sizeof	←
	* / %	→
	+ -	→
	<< >>	→
	< <= > >=	→
	== !=	→
	&	→
	^	→
		→
	&&	→
		→
	?:	←
	= += -= *= /= %= <<= >>= &= = ^=	←
	,	→

Operatori

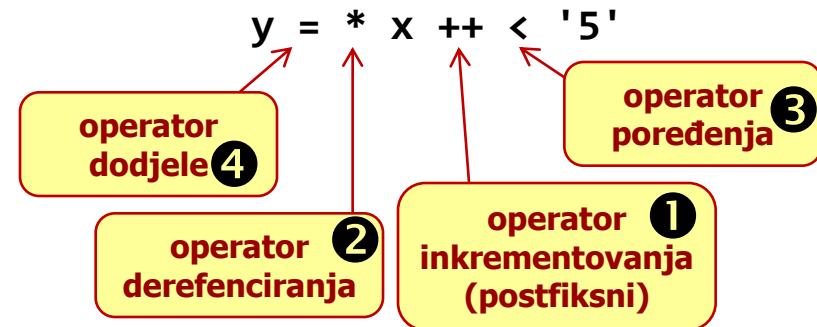
Asocijativnost operatora

- **operatori imaju asocijativnost** (kao i u matematici)
- **asocijativnost definiše koji redom se izvršavaju operatori istog prioriteta:**
 - **lijeva asocijativnost (slijeva udesno)**
 $A \text{ op1 } B \text{ op2 } C == (A \text{ op1 } B) \text{ op2 } C$
npr. binarni operator + ima lijevu asocijativnost
 $1+2+3 \Rightarrow 1+2+3 == (1+2)+3 \Rightarrow 6==6$
 - **desna asocijativnost (zdesna ulijevo)**
 $A \text{ op1 } B \text{ op2 } C == A \text{ op1 } (B \text{ op2 } C)$
npr. operator dodjele = ima desnu asocijativnost
 $a=b=10 \Rightarrow a=(b=10) \Rightarrow b=10, a=10$

Primjer:

U datom složenom izrazu odrediti redoslijed kojim će se izvršiti pojedini operatori

$y = *x++ < '5'$



①
 $y = * (x++) < '5'$

②
 $y = (* (x++)) < '5'$

③
 $y = ((* (x++)) < '5')$

④
 $(y = ((* (x++)) < '5'))$

Operatori

Operator dodjele

- Operatorom dodjele promjenljivoj se dodjeljuje neka vrijednost
- Osnovni operator dodjele (=)

Npr.

```
int n;  
n=100;
```

S lijeve strane operatora dodjele može da se nalazi sve ono čemu može da se dodijeli vrijednost (npr. promjenljiva).
Takvi "entiteti" nazivaju se "L-vrijednosti" (L-value ili left-value)

Desna strana operatora dodjele mora biti takvog tipa da se može dodijeliti entitetu s lijeve strane (istog tipa ili da se može konvertovati u dati tip)

S lijeve strane operatora dodjele ne može da se nalazi ono čemu ne može da se dodijeli vrijednost (npr. konstanta, konstantna vrijednost)!

Npr.

```
const int k=100;  
k=200;
```

✗

Npr.

```
5=a;
```

✗

Operator dodjele je desno asocijativan (asocijativnost zdesna ulijevo)

Npr.

```
a = b = c = 10;
```

↑ ↑ ↑
③ ② ①

$a = (b = (c = 10)); \Rightarrow$

```
c = 10;  
b = c;  
a = b;
```

Operatori

Aritmetički operatori

- aritmetičke operacije nad podacima
- osnovni aritmetički operatori

binarni operator sabiranja (+)

`a=3; c=a+2; ⇒ c ← 5`

binarni operator oduzimanja (-)

`b=3; c=b-2; ⇒ c ← 1`

binarni operator množenja (*)

`a=2; b=3; c=a*b; ⇒ c ← 6`

binarni operator dijeljenja (/)

`i=9; k=4; n=i/k; ⇒ c ← 2`

`p=9.0; q=4; r=p/q; ⇒ c ← 2.25`

binarni operator dijeljenja po modulu – ostatak cjelobrojnog dijeljenja (%)

`c=9%4; ⇒ c ← 1`

`p=6%8; ⇒ p ← 6`

unarni operator promjene znaka (-. +)

`c=4; d=-c; ⇒ d ← -4`

`c=4; d=+c; ⇒ d ← 4`

Asoc.

→

Asoc.

←

Aritmetički operatori imaju polimorfno ponašanje (kontekstno zavisno ponašanje)
– različito se ponašaju u zavisnosti od konteksta, tj. podataka na koje se primjenjuju.

npr. binarni operator sabiranja omogućava:
sabiranje cjelobrojnih podataka ⇒ rezultat je cjelobrojni podatak
sabiranje FP podataka ⇒ rezultat je FP podatak

U slučaju da su operandi različitih tipova, tada se primjenjuju standardna pravila za konverziju.

Jezici koji omogućavaju primjenu operatora nad operandima različitih (unaprijed nepoznatih) tipova, pri čemu se primjenjuju poznata pravila za konverziju tipova, nazivaju se slabo tipizirani jezici.

C je slabo tipizirani jezik.

Operatori

Aritmetički operatori

Operator uvećavanja za jedan – inkrementovanje (++)

Operator umanjivanja za jedan – dekrementovanje (--)

Mogu da se primjenjuju nad cjelobrojnim i FP podacima

prefiksni oblik

postfiksni oblik

Asoc. $\left\{ \begin{array}{l} ++prom \\ --prom \end{array} \right\} \leftarrow$ $\begin{array}{l} prom++ \\ prom-- \end{array} \rightarrow$ Asoc.

Npr.

$n=n+1 \equiv ++n \equiv n++$

$n=n-1 \equiv --n \equiv n--$

Ako ne postoji širi kontekst, tj. ako inkrementovanje čini čitavu naredbu, i ako se vrijednost izraza dalje ne koristi u nekom složenom izrazu, onda nema razlike između $n++$ i $++n$

U složenim izrazima, veoma se razlikuje ponašanje prefiksnog i postfiksno operatora.

Operatori

Aritmetički operatori

Operator uvećavanja za jedan – inkrementovanje (++)

Operator umanjivanja za jedan – dekrementovanje (--)

Operatori sa bočnim efektom!

U složenim izrazima, veoma se razlikuje ponašanje prefiksnog i postfiksno operatora.

U slučaju prefiksnog operatora, prvo inkrementuje/dekrementuje operand, a zatim se dalje koristi ta modifikovana vrijednost.

U slučaju postfiksno operatora, vrijednost operanda se inkrementuje/dekrementuje nakon što bude upotrijebljena u složenom izrazu.

U složenim izrazima, vrijednost izraza ++n je inkrementovana vrijednost promjenljive n

U složenim izrazima, vrijednost izraza n++ je originalna (stara) vrijednost promjenljive n

U složenim izrazima, vrijednost izraza --n je dekrementovana vrijednost promjenljive n

U složenim izrazima, vrijednost izraza n-- je originalna (stara) vrijednost promjenljive n

Npr.

```
i = 2;  
j = (++i)+5;  
-----  
i ← i+1 = 2+1 = 3  
j ← i+5 = 3+5 = 8
```

Npr.

```
i = 2;  
j = (i++)+5;  
-----  
j ← i+5 = 2+5 = 7  
i ← i+1 = 2+1 = 3
```

Kao sporedni – bočni efekat (eng. *side effect*) imamo inkrementovanje/dekrementovanje

Operatori

Aritmetički operatori

Operator uvećavanja za jedan – inkrementovanje (++)

Operator umanjivanja za jedan – dekrementovanje (--)

Postfiksni operatori imaju
prioritet nad prefiksним

Npr.

```
i = 1;  
j = 10;  
k = i +++ j;
```

operator
dodjele ③

postfiksni
operator
inkrementa ①

binarni
operator
sabiranja ②

```
k ← i+j = 1+10 = 11  
i ← i+1 = 1+1 = 2  
j nepromijenjeno
```

Operatori inkrementovanja i
dekrementovanja ne mogu da se
primjenjuju nad konstantama i
konstantnim vrijednostima (literali)!

Mogu da se primjenjuju samo nad
L-vrijednostima

Npr.

```
int p;  
p++;
```



Npr.

```
const int k;  
k++;
```



Npr.

```
int p;  
p = 2++;
```



Operatori

Operatori poređenja

➤ Alternativni nazivi:

relacijski operatori / relacioni operatori

> veće	}	Operatori poretka Asoc: → viši prioritet
< manje		
>= veće ili jednako		
<= manje ili jednako		

== jednako	}	Operatori jednakosti i različitosti Asoc: → niži prioritet
!= različito		

Rezultat primjene operatora poređenja je istinitosna vrijednost: istina=1 / laž=0

Npr.

3<5	=>	1 (istina)
3>5	=>	0 (laž)
3==5	=>	0 (laž)
3!=5	=>	1 (istina)
3<5<2	=>	1 (istina)
3=5	=>	greška

Operatori

Logički operatori

- Primjenjuju se nad istinitosnim (logičkim) vrijednostima

! - negacija / logičko NE (NOT)

&& - konjunkcija / logičko I (AND)

|| - disjunkcija / logičko ILI (OR)

Asoc: ←

Viši prioritet od svih binarnih operatora

Asoc: →

Niži prioritet od aritmetičkih i relacionih operatora

Rezultat primjene logičkih operatora je istinitosna vrijednost: istina=1 / laž=0

x	y	x && y
0	0	0
1	0	0
0	1	0
1	1	1

x	y	x y
0	0	0
1	0	1
0	1	1
1	1	1

x	!x
0	1
1	0

Npr.

5 && 1	=>	1 (istina)
1 && 1	=>	1 (istina)
0 && 1	=>	0 (laž)
!0	=>	1 (istina)
!9	=>	0 (laž)
!(k+1)	=>	0 (laž)
!(!(0))	=>	0 (laž)
!(!(5))	=>	1 (istina)
0 && (i=50)	=>	0 (laž)
1 (j=128)	=>	1 (istina)

Operatori

Logički operatori

Prilikom izračunavanja logičkih izraza primjenjuje se strategija lijenog izračunavanja (eng. *lazy evaluation*) – izračunava se samo ono što je neophodno!

Ako se tokom sračunavanja složenog logičkog izraza, u nekom trenutku postigne vrijednost na koju ne utiče ostatak izraza, prekida se sračunavanje izraza.

Npr.

`1<2 || a<b`

1<2 je ISTINA pa nije neophodno da se izračunava `a<b`, jer vrijednost `a<b` neće uticati na vrijednost kompletnog izraza, tj. vrijednost složenog izraza biće 1, bez obzira na vrijednost izraza `a<b`

Npr.

`1>2 && a<b`

1>2 je LAŽ pa nije neophodno da se izračunava `a<b`, jer vrijednost `a<b` neće uticati na vrijednost kompletnog izraza, tj. vrijednost složenog izraza biće 0, bez obzira na vrijednost izraza `a<b`

Npr.

```
int a=30, b=4, c;  
c = a<10 && b++;
```



Primjenjuje se



Ne primjenjuje se

`a←30, b←4, c←0`



Operatori

Bitski (bitovski) operatori

- Primjenjuju se nad pojedinačnim bitima (bitovima) unutar cjelobrojnih podataka

`~` - bitska negacija / bitsko NOT

`<<` - bitsko pomjeranje (šiftovanje) ulijevo

`>>` - bitsko pomjeranje (šiftovanje) udesno

`&` - bitska konjunkcija / bitsko AND

`^` - bitska ekskluzivna disjunkcija / bitsko XOR

`|` - bitska disjunkcija / bitsko OR

Asoc: ←
najviši prioritet (unarni operator)

Asoc: →
prioritet između aritmetičkih i
relacionih operatora

Asoc: →
prioritet između relacionih i
logičkih operatora

x	y	$x \wedge y$
0	0	0
1	0	1
0	1	1
1	1	0

Operatori

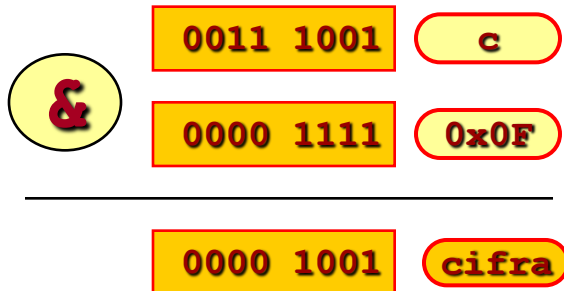
Bitski (bitovski) operatori

Primjenjuju se nad pojedinačnim bitima (bitovima) unutar cjelobrojnih podataka!

& - bitsko AND

Npr.

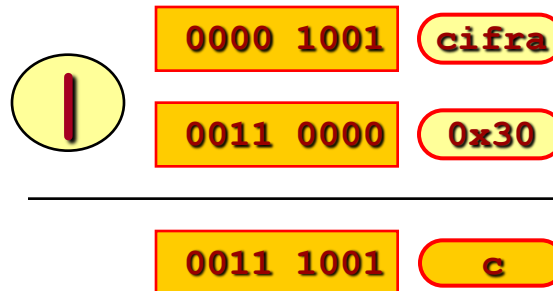
```
unsigned char c='9', cifra;  
cifra = c & 0x0F;  
printf( "%d\n", cifra );
```



| - bitsko OR

Npr.

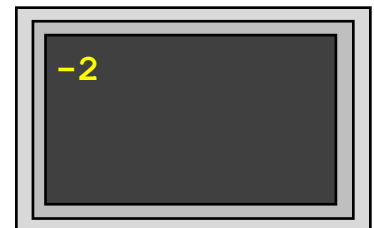
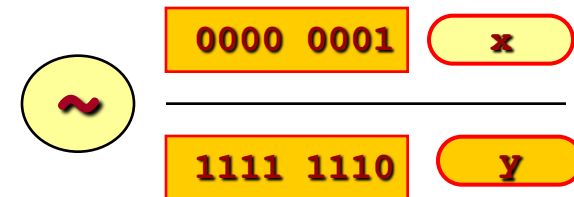
```
unsigned char cifra=9, c;  
c = cifra | 0x30;  
printf( "%c\n", c );
```



~ - bitsko NOT

Npr.

```
unsigned char x=1, y;  
y = ~x;  
printf( "%d\n", y );
```



Operatori

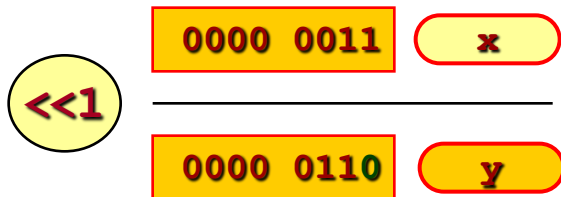
Bitski (bitovski) operatori

Primjenjuju se nad pojedinačnim bitima (bitovima) unutar cjelobrojnih podataka!

<< - shift ulijevo

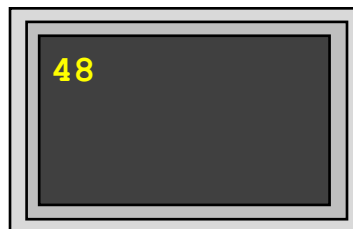
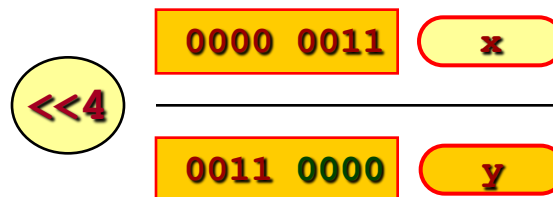
Npr.

```
unsigned char x=3, y;  
y = x<<1;  
printf( "%d\n", y );
```



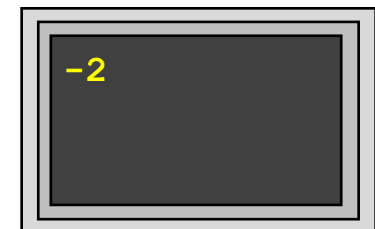
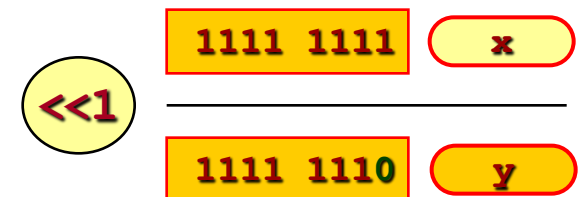
Npr.

```
unsigned char x=3, y;  
y = x<<4;  
printf( "%d\n", y );
```



Npr.

```
signed char x=-1, y;  
y = x<<1;  
printf( "%d\n", y );
```



Šiftovanje ulijevo ekvivalentno je množenju sa 2!
Primjenjuje se tzv. aritmetičko šiftovanje (zadržava se znak)

Operatori

Bitski (bitovski) operatori

>> - shift udesno

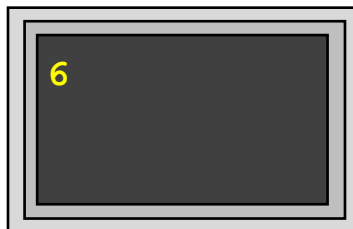
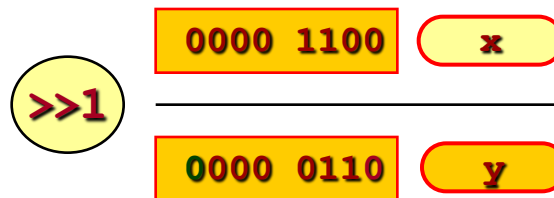
Šiftovanje udesno ekvivalentno je dijeljenju sa 2!

Primjenjuje se tzv. aritmetičko šiftovanje (zadržava se znak)



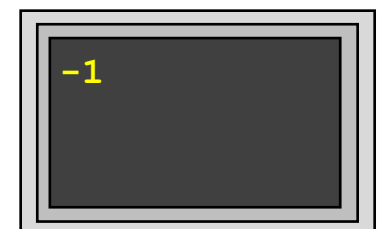
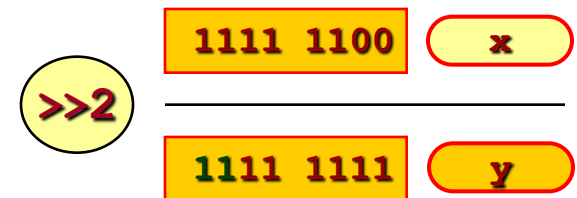
Npr.

```
unsigned char x=12, y;  
y = x>>1;  
printf( "%d\n", y );
```



Npr.

```
signed char x=-4, y;  
y = x>>2;  
printf( "%d\n", y );
```



Operatori

Složeni operatori dodjele

- Pored osnovnog operatora dodjele (=) postoje i složeni operatori dodjele
- Složeni operatori dodjele omogućavaju formiranje kompaktnijih izraza

$$lv = lv \text{ op } rv; \Leftrightarrow lv \text{ op} = rv;$$

Složeni operatori dodjele (za aritmetičke op.)

+= -= *= /= % =

$x = x + y \Rightarrow x += y$ Npr. $x += 3 \equiv x = x + 3$

$x = x - y \Rightarrow x -= y$ Npr. $x -= 3 \equiv x = x - 3$

$x = x * y \Rightarrow x *= y$ Npr. $x *= 3 \equiv x = x * 3$

$x = x / y \Rightarrow x /= y$ Npr. $x /= 3 \equiv x = x / 3$

$x = x \% y \Rightarrow x \% = y$ Npr. $x \% = 3 \equiv x = x \% 3$

Složeni operatori dodjele (za bitske op.)

<<= >>= &= ^= |=

$x = x << y \Rightarrow x << = y$ Npr. $x << = 1 \equiv x = x << 1$

$x = x >> y \Rightarrow x >> = y$ Npr. $x >> = 3 \equiv x = x >> 3$

$x = x \& y \Rightarrow x \& = y$ Npr. $x \& = 3 \equiv x = x \& 3$

$x = x \wedge y \Rightarrow x \wedge = y$ Npr. $x \wedge = 5 \equiv x = x \wedge 5$

$x = x | y \Rightarrow x | = y$ Npr. $x | = 5 \equiv x = x | 5$

**Prioritet svih operatora dodjele je isti,
a asocijativnost desna (zdesna ulijevo)**

Npr.

```
int x=10, y=100;
x += y += 2;    }    y=y+2;    // y←102
                  }    x=x+y;    // x←112
```

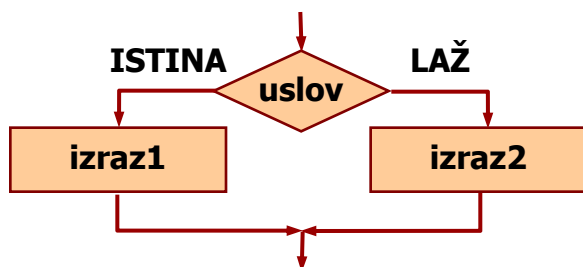
Operatori

Uslovni operator (?:)

- Uslovni operator (operator uslova) je **ternarni operator** (ima tri operanda)

Opšti oblik:

`uslov ? izraz1 : izraz2`



Prioritet uslovnog operatora je viši samo od operatora dodjele i operatora zapeta, a asocijativnost desna (zdesna ulijevo)

I kod uslovnog operatora se primjenjuje princip lijenog izračunavanja

Npr.

```
y = 5;  
x = y>0 ? 100 : 200;  
-----  
x ← 100
```

Npr.

```
y = -2;  
x = y>0 ? 100 : 200;  
-----  
x ← 200
```

Npr.

```
int x, y=5;  
x = y ? y?10:20 : y?100:200;  
printf("x=%d y=%d", x, y);  
-----  
x=10 y=5
```

Npr.

```
x = 3>5 ? n++ : 200;  
-----  
x ← 200
```



Operatori

Operator zapeta (,)

- Služi za spajanje više izraza
- Zapeta je binarni operator, najnižeg prioriteta
- Ima lijevu asocijativnost pa je vrijednost izraza

a , b ≡ b

Npr.

y=4 , x=5 ; ⇔ y=4 ; x=5 ;

Npr.

y=4 , x ; ⇔ y=4 ; x ;

Npr.

y= (1 , 2) ; ⇔ y=2 ;

Npr.

y= (4 , x) ; ⇔ y=x ;

Operator sizeof

- Vraća broj bajtova koje zauzima neki podatak

sizeof(podatak)

Npr.

```
int i,j,k;  
i=sizeof(int);  
k=sizeof(i+j);
```

} i←4
 k←4

**Vrijednost izraza se ne izračunava,
samo se određuje broj bajtova potreban
za reprezentaciju vrijednosti!**

Npr.

```
int i=10,k;  
k=sizeof(i++);
```

} k←4
 i←10





Konverzije tipova

Konverzija tipa

- **Konverzija tipa predstavlja pretvaranje vrijednosti jednog tipa u vrijednost drugog tipa.**
- Jezik C je veoma fleksibilan po pitanju konverzije tipova i dopušta **korištenje vrijednosti jednog tipa tamo gde se očekuje vrijednost drugog tipa (slabo tipiziran jezik)**.
- Konverzije tipova omogućavaju **miješanje podataka različitih tipova u okviru istog izraza**, ali **mogu dovesti do gubitka podataka ili njihove loše interpretacije**.
- Prilikom konverzije tipa **konvertuju se samo vrijednosti koje figurišu u izrazu**, a **promjenljive sve vrijeme ostaju istog onog tipa koji im je pridružen deklaracijom (statički tipiziran jezik)**.
- **Konverzija može biti:**
 - **eksplicitna** – izvršavaju se na zahtjev programera (specifikuje ih programer)
 - **implicitna** – izvršavaju se automatski, kada je to potrebno
- **Promocija (napredovanje u tipu)**
 - **konverzija vrijednosti „manjeg tipa“ u vrijednost „većeg tipa“**
 - Neki primjeri promocije:
char → int, int → float, float → double, ...
- **Democija (nazadovanje u tipu)**
 - **konverzija vrijednosti „većeg tipa“ u vrijednost „manjeg tipa“**
 - Neki primjeri democije:
int → short, int → char, float → int, ...

Konverzije tipova

Promocija (napredovanje u tipu)

- konverzija vrijednosti „manjeg tipa“ u vrijednost „većeg tipa“
(npr. char → int, int → float, float → double, ...)
- Prilikom promocije uglavnom ne dolazi do gubitka informacije, ali može...

Npr.

```
int x=2;  
long long y;  
y=x;
```

Promocija bez gubitka informacije
x=2, y=2

00 00 00 02

x

00 00 00 00

00 00 00 02

y

Npr.

```
int x=-2;  
long long y;  
y=x;
```

Promocija bez gubitka informacije
x=-2, y=-2

FF FF FF FE

x

FF FF FF FF

FF FF FF FE

y

Npr.

```
int x=2;  
float f;  
f=x;
```

Promocija bez gubitka informacije
(mantisa može da se predstavi)

x $2_{10} = 10_2 = 1.0 \cdot 2^1$

eksponent

Mantisa

Može da se predstavi sa 23 bita

Npr.

```
int x=0xffffffff;  
float f;  
f=x;
```

Promocija sa gubitkom informacija
(mantisa ne može da se predstavi)

x $0\text{ffffff}\text{ff}_{16}$

00001111 11111111 11111111 11111111₂
1.11111111111111111111111111111111 · 2²⁷

27 bita

ne može da se reprezentuje 23-bitnom mantisom

Konverzije tipova

Democija (nazadovanje u tipu)

- konverzija vrijednosti „većeg tipa“ u vrijednost „manjeg tipa“
(npr. int → short, int → char, float → int, ...)
- Prilikom democije može da dode do gubitka informacije (u slučaju da se polazna vrijednost ne može predstaviti u okviru novog tipa)

Npr.

```
int x=2.75;
```

2.75 je FP podatak
Konverzija FP podataka u cjelobrojne podatke vrši se odbacivanjem decimala!

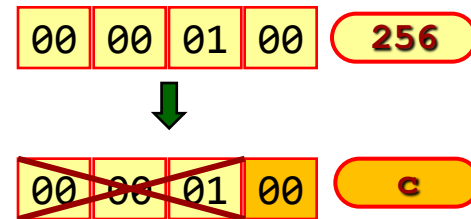
00 00 00 02

x

Democija cjelobrojnih vrijednosti vrši se "odsijecanjem" vodećih bitova!

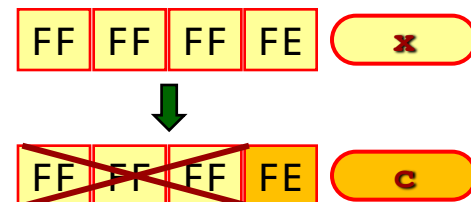
Npr.

```
unsigned char c=256;
```



Npr.

```
int x=-2;  
signed char c;  
c=x;
```





Konverzije tipova

Eksplisitne konverzije

- eksplisitne konverzije izvršavaju se na zahtjev programera, tj. programer ih eksplisitno specifikuje u programu
- Eksplisitne konverziju specifikuju se **operatorom eksplisitne konverzije tipa**
- Operator eksplisitne konverzije tipa alternativno se naziva **operator kastovanja** (eng. *type cast operator*)

eksplisitna konverzija specifikuje se tako što se **identifikator ciljnog tipa navede unutar malih zagrada ispred izraza koji se konvertuje**
(tip) podatak

Ako se operator kastovanja primjenjuje na promjenljivu, ne mijenja se njen tip (C je statički tipiziran jezik), mijenja se samo tip vrijednosti koja se koristi u izrazu!

Npr.

```
int x=5, y=2;  
double d = x/y;
```

Nema eksplisitne konverzije
x i y su cjelobrojne promjenljive
primjenjuje se cjelobrojno dijeljenje $5/2 \rightarrow 2$
pa nakon toga implicitna konverzija $d \leftarrow 2.0$

Npr.

```
int x=5, y=2;  
double d = (double)x / (double)y;
```

Vrijednost promjenljive x se kaste u FP podatak dvostruke preciznosti i vrijednost promjenljive y se kaste u FP podatak dvostruke preciznosti
Nakon eksplisitne konverzije imamo dijeljenje FP podataka ($5.0/2.0 \rightarrow 2.5$) pa nakon toga inicijalizaciju promjenljive $d \leftarrow 2.5$



Konverzije tipova

Implicitne konverzije

- izvršavaju se automatski, kada je to potrebno, pri čemu postoje jasno definisana pravila
- uobičajeno se primjenjuju prilikom:
 - **dodjele vrijednosti**
 - **aritmetičkih operacija**
 - **prenosa parametara u funkcije**
 - ...

➤ Implicitne konverzije kod dodjele

- **Prilikom primjene operatora dodjele vrši se implicitna konverzija vrijednosti desne strane u tip lijeve strane, pri čemu se za tip dodjele uzima tip lijeve strane.**

Npr.

```
int x;  
double y = (x = 2.5);
```

Prvo se FP vrijednost 2.5 implicitno konvertuje u cjelobrojnu vrijednost 2 i dodjeljuje promjenljivoj x, jer je promjenljiva x cjelobrojna.

Vrijednost promjenljive x je cjelobrojna pa se implicitno konvertuje u FP podatak dvostruke preciznosti, kojim se inicijalizuje promjenljiva y.



Konverzije tipova

➤ Implicitne konverzije u aritmetičkim izrazima

- Prilikom primjene nekih operatora vrše se implicitne konverzije koje obezbjeđuju da operandi postanu istog tipa pogodnog za primjenu operatora.
- Ove konverzije nazivaju se **uobičajene aritmetičke konverzije** i primjenjuju se kod:
 - aritmetičkih operatora (+, -, *, /)
 - relacionih operatora (<, >, <=, >=, ==, !=)
 - uslovnog operatora ?:
- Aritmetički operatori ne primjenjuju se na „male“ tipove tj. na podatke tipa char i short (jer će vjerovatno doći do prekoračenja formata), nego se **prije primjene operatora mali tipovi promovišu u tip int**. Ovo se naziva **cjelobrojna promocija** (*eng. integer promotion*)

Npr.

```
unsigned char rez, c1, c2, c3;  
c1 = 100;  
c2 = 3;  
c3 = 4;  
rez = c1 * c2 / c3;
```

Vrijednosti promjenljivih c1, c2 i c3 se implicitno promovišu u int prije sračunavanja izraza c1*c2/c3.

Rezultat sračunavanja izraza je int.

Na kraju se vrijednost izraza implicitno konvertuje (democija) u unsigned char i dodjeljuje promjenljivoj rez.

Ako ne bi bilo promocije, ne bismo dobili dobar rezultat, jer 100*3 ne može da se reprezentuje jednim bajtom pa bi došlo do gubitka informacije.



Konverzije tipova

➤ Pravila za aritmetičke konverzije

1. Ako je bar jedan od operandi tipa long double i drugi se promovira u long double;
2. inače, ako je bar jedan od operandi tipa double i drugi se promovira u double;
3. inače, ako je bar jedan od operandi tipa float i drugi se promovira u float;
4. inače, svi operandi tipa char i short se promoviraju u int;
5. ako je jedan od operandi tipa long long i drugi se prevodi u long long;
6. inače, ako je jedan od operandi tipa long i drugi se prevodi u long.

➤ Pravila za konverzije označenih i neoznačenih cjelobrojnih tipova

1. Ako je neoznačeni operand širi od označenog, označeni operand se konvertuje u neoznačeni širi tip;
2. inače, ako je tip označenog operanda takav da može da predstavi sve vrijednosti neoznačenog tipa, tada se neoznačeni tip prevodi u širi označeni;
3. inače, oba operanda se konvertuju u neoznačeni tip koji odgovara označenom tipu.



Konverzije tipova

➤ Pravila za konverzije označenih i neoznačenih cjelobrojnih tipova

1. Ako je neoznačeni operand širi od označenog, označeni operand se konvertuje u neoznačeni širi tip;

Npr.

```
unsigned int ui=1;
signed char sc=-2;
printf("%d", ui>sc);
```

0

neoznačeni int je širi od signed char pa se signed char konvertuje u unsigned int, tj. -2 postaje jako velika pozitivna vrijednost, pa 1 nije veće od jako velike pozitivne vrijednosti

2. inače, ako je tip označenog operanda takav da može da predstavi sve vrijednosti neoznačenog tipa, tada se neoznačeni tip prevodi u širi označeni;

Npr.

```
signed short ss=-1;
unsigned char uc=127;
printf("%d", ss<uc);
```

1

označeni short može da predstavi sve neoznačene char pa se neoznačeni char konvertuje u označeni short (ostaje 127) pa je -1 manje od 127

3. inače, oba operanda se konvertuju u neoznačeni tip koji odgovara označenom tipu.

Npr.

```
signed int si=-1;
unsigned int ui=1;
printf("%d", si<ui);
```

0

označeni int i neoznačeni int su iste širine pa se označeni int konvertuje u neoznačeni int, tj. -1 postaje jako velika pozitivna vrijednost, a to nije manje od 1