

## Uvod u programiranje pojam algoritma

## Algoritam

---

- Pravilo (ili skup pravila) kojim se opisuje kako riješiti neki problem i koje posjeduje sljedeća svojstva:
  - algoritam je precizan
  - algoritam je jednoznačan
  - algoritam obuhvaća konačni broj koraka; svaki korak je opisan instrukcijom
  - za algoritam su definirani početni objekti (koji pripadaju nekoj klasi objekata) nad kojima se obavljaju operacije
  - ishod obavljanja algoritma je skup završnih objekata (rezultat), tj. algoritam je djelotvoran (*effective*)
- Postupak obavljanja algoritma je **algoritamski proces**

## Primjer algoritma – kiseljenje krastavaca

---

- Početni objekti:
  - 5 kg krastavaca, 1 l alkoholnog octa (9%), 30 dag šećera, 10 dag soli, kopar, papar
- Krastavce i kopar oprati i posložiti u čiste staklenke
- U 2 l vode dodati ocat, šećer, sol i papar
- Zakuhati uz miješanje
- Vruću otopinu uliti u staklenke
- Staklenke zatvoriti celofanom i gumicom
- Složiti staklenke u široki lonac napunjen vodom do grla staklenki
- Zagrijati vodu do 80 stupnjeva. Ako toplomjer nije raspoloživ, zagrijavati dok se s dna ne počnu dizati mjehurići zraka
- Staklenke izvaditi, obrisati i složiti na police u smočnicu
- Ostaviti stajati barem 24 sata
- Završni objekti:
  - kiseli krastavci á la FER

## Primjer algoritma – ručno zbrajanje dvaju višeznamenkastih dekadskih brojeva

---

- Početni objekti:
  - 2 pribrojnika
- Napisati dekadске brojeva tako da znamenke jedinica budu potpisane jedna ispod druge, znamenke desetica i sve ostale znamenke također. Ako u manjem broju ne postoji znamenka koje u većem ima, zamisliti da na tom mjestu piše 0.
- Zamisliti broj koji ćemo nazvati *prijenos*, i postaviti ga na vrijednost 0
- Za svako težinsko mjesto, krenuvši s desna na lijevo, učiniti
  - Zbrojiti dvije odgovarajuće znamenke i *prijenos*
  - Na odgovarajuće težinsko mjesto rezultata upisati znamenku jedinica tako dobivene sume
  - Broju *prijenos* dodijeliti vrijednost znamenke desetica tako dobivene sume
- Ako je *prijenos* > 0, upisati ga ispred rezultata
- Završni objekt:
  - zbroj

## Algoritam

---

- Instrukcije moraju biti izvedive i jednoznačne
  - Primjeri za nedopuštene instrukcije:
    - izračunaj  $5/0$
    - uvećaj  $x$  za 6 ili 7

## Algoritam

---

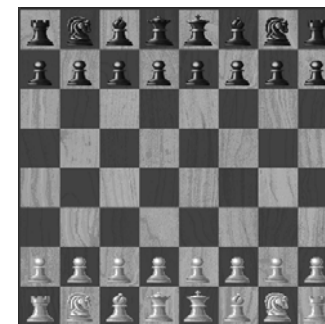
- Algoritam je **učinkovit** (*efficient*) ako se rezultat dobije u prihvatljivom ili "razumnom" vremenu.

*Primjer:* Algoritam koji bi izabirao potez igrača šaha tako da ispita sve moguće posljedice poteza, zahtijevao bi milijarde godina na najbržem zamislivom računalu. Zašto?

- 20 mogućih prvih poteza bijelog
- 20 mogućih prvih poteza crnog
- > 20 mogućih drugih poteza bijelog
- > 20 mogućih drugih poteza crnog
- itd...

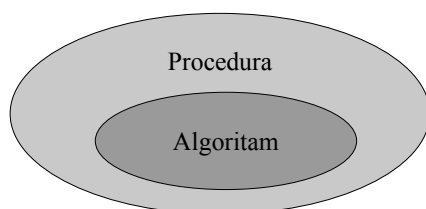
Za 10 poteza svakog igrača, barem  $20^{20}$  kombinacija  $\sim 10^{26}$

Kad bi se 1 kombinacija analizirala 1  $\mu$ s, to je 3170979198376 godina!



## Procedura

---



- Postupak koji ima sva svojstva kao i algoritam, ali ne mora završiti u konačnom broju koraka jest računalna procedura.
- Primjeri za proceduru:
  - Operacijski sustav računala
  - Uređivač teksta

## Algoritmi i programi

---

- *Program* - opis algoritma koji u nekom programskom jeziku jednoznačno određuje što računalo treba napraviti.
- *Programiranje* – proces opisivanja algoritma nekim od programskih jezika
- Algoritmi + strukture podataka = PROGRAMI (*Wirth*)
  - kako osmisлити algoritme
  - kako strukturirati podatke
  - kako formulirati algoritme
  - kako verificirati korektnost algoritama
  - kako analizirati algoritme
  - kako provjeriti (testirati) program
- Postupci izrade algoritama nisu jednoznačni te zahtijevaju i kreativnost. Inače bi već postojali generatori algoritama. Znači da se (za sada?) gradivo ovog predmeta ne može u potpunosti algoritmizirati. Koristit će se programski jezik C. Za sažeti opis složenijih algoritama može se koristiti pseudokod.

## Vrste programske podrške

---

- **Sistemska programska podrška**
  - Operacijski sustavi (MS-DOS, UNIX/Linux, Windows)
  - Uslužni (utility) programi (prevodioci, uređivači teksta):
  - Servisi (Internet poslužitelj, poslužitelj baze podataka)
  
- **Aplikativna (namjenska, primijenjena)**
  - Rješava probleme specifične za neku struku
  - Tablični kalkulatori (npr. Excel), obrađivači teksta (npr. Word), mrežno planiranje (npr. Project)...

## Redoslijed rješavanja manjih programa

---

1. Uočavanje (identifikacija) problema i postavljanje programskog zadatka
2. Oblikovanje programa (razvoj algoritma)
3. Konverzija algoritma u logiku razumljivu računalu
4. Kodiranje
5. Upis programskog kôda u računalo
6. Prevođenje (kompilacija) programa
7. Ispravljanje formalnih pogrešaka
8. Kolekcija programa (stvaranje izvršnog programa)
9. Izvođenje programa s test podacima
10. Ispravljanje uočenih logičkih pogrešaka
11. Korištenje programa s aktualnim podacima

## Oblikovanje većih programa

---

- **Organizacija posla (tima)**
- **Primjena prikladne metodologije**
  - Pristup od vrha prema dolje (*top-down*)
    - oblikovanje počinje specifikacijom na najvišoj razini, a zatim se dijelovi specifikacije u koracima dijele i opisuju na sve nižim razinama, dok se ne dospije do razine u kojoj specifikacija dovoljno precizno definira programsko rješenje
  - Pristup od dna prema gore (*bottom-up*)
    - oblikovanje počinje rješavanjem detaljnih problema na najnižoj razini, a zatim se dobivena rješenja udružuju na višim hijerarhijskim razinama
- **Izbor programerskih pomagala**
  - Dijagrami toka
  - Struktogrami
  - Pseudokodiranje
  - ...

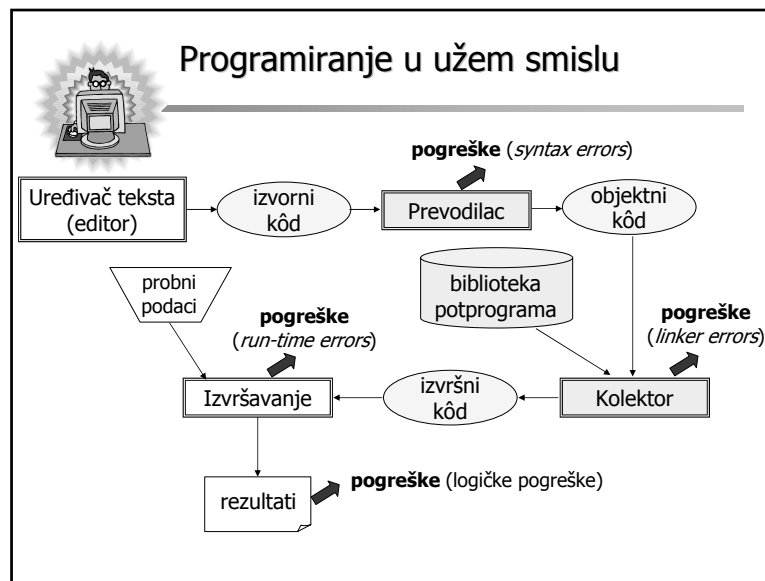
## Oblikovanje većih programa

---

- **Izrada specifikacija programa**
  - Prikupljanje zahtjeva
  - Oblikovanje ulaza, izlaza i interakcije programa s korisnikom
  - Specifikacija mora biti neovisna o korištenom programskom jeziku
  
- **Izrada programske dokumentacije**
  - Dokumentacija o svakoj fazi razvoja
  - Programska dokumentacija
  - Komentari u kôdu
  
- **Verifikacija svake faze oblikovanja**

## Izbor programskog jezika

- Izbor programskog jezika
  - FORTRAN, PASCAL, COBOL
  - C/C++
  - Visual Basic
  - 4GL (4<sup>th</sup> Generation Language)
  - Java, C#



## Programiranje u užem smislu

- Unos izvornog programa (*source code*)
  - ASCII uređivač teksta (EDIT, vi, ...)
  - Uređivač teksta ugrađen u radnu okolinu programera (MS Visual Studio)
- Prevođenje izvornog programa u objektni program
  - Poziv prevodioca (*compiler*)
  - Prevodilac otkriva sintaktičke (pravopisne, formalne) pogreške
    - programer ispravlja izvorni kôd i ponovo pokreće prevođenje
- Kolekcija (povezivanje) prevedenog programa u izvršni (apsolutni) program
  - Poziv kolektora (*linker*)
  - Povezuju se potrebne potprogramske biblioteke (`stdio.h`, `math.h`, ...)
  - Kolektor otkriva pogreške
    - programer ispravlja izvorni kôd i ponovno pokreće prevođenje

## Programiranje u užem smislu

- Izvođenje izvršnog programa
  - Definiranje skupova ulaznih probnih podataka i očekivanih rezultata
  - Izvršavanje programa na osnovi probnih podataka
    - Pogreške koje se otkrivaju prilikom izvršavanja (*run-time errors*)
      - npr. Division by zero
    - Logičke pogreške
      - program "radi" (ne dojavljuje pogreške), ali daje pogrešne rezultate
    - programer ispravlja izvorni kôd i ponovo pokreće prevođenje/izvođenje

## PRIMJER

- Programski zadatak  
pronaći najveći od tri zadana broja
- Pseudokod koji koristi isključivo termine govornog jezika

```
pročitaj tri realna broja
ispiši pročitane brojeve
odredi najveći broj
ispiši nađeni broj
```

## PRIMJER - nastavak

- Pseudokod koji koristi uobičajene simbole

```
pročitaj (x,y,z)
ispiši (x,y,z)
{odredi najveći broj}
  ako je x > y tada
    ako je x > z tada
      rez := x
    inače
      rez := z
  inače
    ako je y > z tada
      rez := y
    inače
      rez := z
ispiši (rez)
kraj
```

## PRIMJER - nastavak

- Unapređenje prethodnog rješenja

```
pročitaj (x,y,z)
ispiši (x,y,z)
{odredi najveći broj}
  rez := z
  ako je x > y tada
    ako je x > z tada
      rez := x
    inače
      ako je y > z tada
        rez := y
  ispiši (rez)
kraj
```

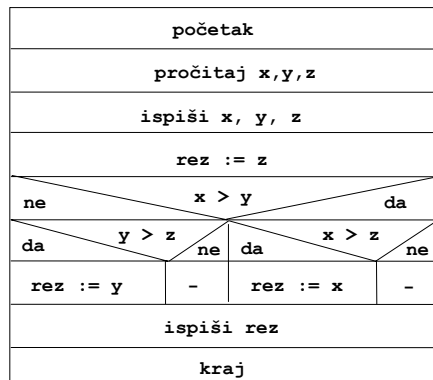
## Tablica traga

- Očekivani sadržaj memorijskih varijabli tijekom izvršenja programa (testiranje na papiru)

x	y	z	rez
1	-2	4	4
4	1	-2	-2
1	4	-2	-2

## PRIMJER - nastavak

### ☐ Struktogram



## PRIMJER - nastavak

### ☐ Kôd u programskom jeziku C

```
#include <stdio.h>
int main() {
    float x, y, z, rez;
    scanf("%f %f %f", &x, &y, &z);
    printf("%f %f %f \n", x, y, z);
    /* odredi najveći broj */
    rez = z;
    if ( x > y ) {
        if ( x > z) rez = x;
    } else {
        if ( y > z) rez = y;
    }
    printf("%f \n", rez);
    return 0;
}
```

Utorak 29.9.2005

## Uvod u C-programiranje opća pravila pisanja C-programa

## Opća pravila pisanja C programa

- C razlikuje velika i mala slova. Npr:  
`sum`  
`Sum`  
`SUM`
- C je jezik slobodnog formata (nema pravila koja propisuju stil pisanja)
- mjesto početka naredbe u retku je proizvoljno
- dopušteno je stavljanje više naredbi u istom retku. Npr:  
`int i,n; printf("Unesite n: "); scanf("%d", &n);`
- poželjno je umetanje praznina i praznih redova

## PRIMJER - što radi ovaj program?

```
#include <stdio.h>
int main() {float x, y, z, rez;scanf("%f %f %f",
&x, &y, &z);
printf
("%f %f %f \n", x, y
, z); rez
= z ; if( x
> y ) {if ( x>z) rez
= x;} else{if (
y >
z) rez=y
;}printf("%f \n"
, rez);
return 0;}
```

## PRIMJER - što radi ovaj program?

```
#include <stdio.h>
int main() {
float x, y, z, rez;
scanf("%f %f %f", &x, &y, &z);
printf("%f %f %f \n", x, y, z);
rez = z;
if (x > y) {
if (x > z) rez = x;
} else {
if (y > z) rez = y;
}
printf("%f \n", rez);
return 0;
}
```

Različiti stilovi

```
if (x > y) {
rez = x;
}
```

```
if (x > y)
{
rez = x;
}
```

```
if (x > y)
{
rez = x;
}
```

```
if (x > y)
{
rez = x;
}
```

## Ključne riječi

- predefinirani identifikatori koji za prevodioca imaju posebno značenje
- ključne riječi se pišu malim slovima
- Prema ANSI standardu C ima sljedeće 32 ključne riječi:

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

ANSI - American National Standards Institute

## Struktura C programa

- C program se sastoji od imenovanih blokova, deklaracija/definicija varijabli, direktiva preprocesoru
  - imenovani blokovi se nazivaju **funkcije**. Za nazive funkcija se ne smiju koristiti ključne riječi.
  - deklaracijom se opisuje naziv i tip varijable. Definicija je deklaracija kojom se osim opisa varijable, rezervira prostor u memoriji
- blok započinje znakom **{** i završava znakom **}**
- blok obuhvaća deklaracije/definicije, naredbe (*statement*) i neimenovane blokove
- svaka naredba i deklaracija/definicija mora završavati znakom **;**
  - blok NE završava znakom **;** tj. iza znaka **}** ne stavlja se **;**

## Struktura C programa

```
float x;                                definicija varijable
extern float y;                          deklaracija varijable

int suma(int i, int j) {                 imenovani blok (funkcija)
    int k;                                definicija varijable
    {                                     neimenovani blok
        int m;                            definicija varijable
        {                                 neimenovani blok
            m = i + j;                    naredba
            k = m;                        naredba
        }
    }
    return k;
}

int produkt(int i, int j) {              imenovani blok (funkcija)
    ...
}
```

## Struktura C programa

- u C programu mora postojati glavna (main) funkcija koja predstavlja mjesto gdje počinje izvršenje programa:

```
int main() {
    programski blok
    return 0;
}
```

ili

**POGREŠNO!!!**

```
void main() {
    programski blok
}
```

## Struktura C programa

```
/* Racunanje sume el. polja */          komentar
#include <stdio.h>                       direktive pretprocesoru
#define MAX 5
float suma (int n);                      definicija varijabli
float dat[MAX] = {2.5,2.3,0.,1.,5.};
int main() {                              funkcija main
    int n;                                definicija varijabli
    float sum;
    printf("Unesi broj elemenata");
    scanf("%d", &n);                      tijelo funkcije main
    sum = suma(n);
    printf("%9.2f\n", sum);
    return 0;
}                                          kraj funkcije main
```

## Struktura C potprograma

```
float suma(int n) {                       funkcija suma
    int i;                                definicija varijabli
    float s = 0;

    for(i = 0; i < n; i++)
        s += dat[i];                     tijelo funkcije suma
    return s;
}                                          kraj funkcije suma
```

## Komentari

---

- mogu se protezati kroz više linija
- izbjegavati komentar oblika:  
`printf("Unesi n: ");/* Ispis na zaslonu */`  
zato što program postaje nečitkiji
- nije dopušteno koristiti komentar unutar komentara:  
`/* definicija */ funkcije */ sume */`

## Pretprocesorske naredbe

---

- `#include <stdio.h>` uključuje u program prije prevođenja standardno zaglavlje `<stdio.h>` koje sadrži definicije tipova i funkcijskih prototipova (na primjer `printf`, `scanf` i druge).
- `#define MAX 5` definira simboličku konstantu MAX i pridjeljuje joj vrijednost (npr. 5) što je veoma pogodno kod parametrizacije programa

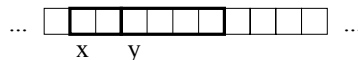
## Varijable

---

- Općenito: promjenljiv podatak (lat. *variabilis*-promjenljiv)
- U programiranju: varijabla je prostor u memoriji računala, poznate veličine, kojemu je dodijeljeno ime i čiji se sadržaj može mijenjati
- Simbolički se prikazuje pravokutnikom uz koji stoji ime

x       y

- Smještaj u memoriji računala



## Varijable

---

- svaka varijabla se obavezno mora definirati/deklarirati prije korištenja
- imena varijabli i funkcija su sastavljena od slova i brojki, a prvi znak mora biti slovo ili znak potcrtavanja `_`

```
suma god_rod    x1    pripremni_dio_studija  
94god   novi+datum   x1.1   matieni broj
```

- velika i mala slova se razlikuju (imena varijabli i funkcija se obično pišu malim slovom, imena simboličkih konstanti velikim)
- duljina može biti proizvoljna (značajno prvih 31 znakova)
- ključne riječi se **ne smiju** koristiti za imena varijabli

---

## Uvod u C-programiranje osnovni tipovi podataka

## Osnovni tipovi podataka

---

### ▪ Osnovni tipovi podataka

- `int` - cjelobrojni tip
- `float` - realni tip
- `double` - realni tip u dvostrukoj preciznosti
- `char` - znakovni tip (ili mali cijeli broj)

## Cjelobrojni tip podatka i prefiksi

---

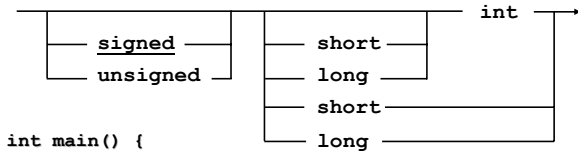
- `int` - cjelobrojni tip
- Prefiksi ili kvalifikatori
  - `short` - smanjuje raspon cjelobrojnih vrijednosti koje varijabla može sadržavati
  - `long` - povećava raspon cjelobrojnih vrijednosti koje varijabla može sadržavati
  - `unsigned` - dopušta pridruživanje samo pozitivnih vrijednosti
  - `signed` - dopušta pridruživanje pozitivnih i negativnih vrijednosti (zato dopušta manji raspon brojeva od *unsigned*)

## Cjelobrojni tip podatka i preciznost

---

- Cjelobrojni se tip podatka (integer), s obzirom na preciznost (broj znamenki), može deklarirati kao `short` ili `long`.
- U programskom jeziku C ne postoji ograničenje na preciznost `short`-a ili `long`-a, ali vrijede sljedeća pravila:
- `short` ne može biti precizniji od `int`
- `int` ne može biti precizniji od `long`
- odnosno za preciznost se može napisati:  
$$\text{short} \leq \text{int} \leq \text{long}$$
- U cjelobrojni tip podatka također pripada: `char` (kada predstavlja brojevnu, a ne znakovnu vrijednost).

## Formalna deklaracija cjelobrojnog tipa



```
int main() {
    signed int i;
    i = 123;
    return 0;
}
```

signed short int i; ⇔ signed short i; ⇔ short int i; ⇔ short i;  
 signed int i; ⇔ int i;  
 signed long int i; ⇔ signed long i; ⇔ long int i; ⇔ long i;  
 unsigned short int i; ⇔ unsigned short i;  
 unsigned int i;  
 unsigned long int i; ⇔ unsigned long i;

## Zašto je to važno znati?

- Primjer u programskom jeziku C
- Pretpostavka: *short int* koristi dva okteta

### Brojanje

```
int main () {
    short int i;
    i = 0;
    while (i < 100000) {
        i = i + 1;
    }
    printf ("Gotovo!");
    return 0;
}
```

- Što se i zašto dogodilo?

## Binarni brojevni sustav

- Pouzdano i neosjetljivo na manje promjene napona:
  - npr. 0 – 2,5 V ⇔ znamenka 0
  - 2,51 – 6 V ⇔ znamenka 1
- Znamenke su **0** i **1**, dakle baza brojanja **B=2** što određuje **binarni brojevni sustav**
- Iz engleskog **Bi**nary **di**git nastalo je ime za najmanju količinu informacije, znamenku binarnog brojevnog sustava **BIT**.
- Broj od  $n$  znamenki u brojevnom sustavu s bazom 2:
 
$$z_{n-1} \cdot 2^{n-1} + z_{n-2} \cdot 2^{n-2} + \dots + z_1 \cdot 2^1 + z_0 \cdot 2^0, \quad z_i \in \{0, 1\}$$

## Pretvorba dekadskog broja u binarni

$$N = z_{n-1} \cdot 2^{n-1} + z_{n-2} \cdot 2^{n-2} + \dots + z_1 \cdot 2^1 + z_0 \cdot 2^0$$

Izlučimo li iz svih pribrojnika, osim posljednjeg, zajednički faktor 2:

$$N = 2 \cdot (z_{n-1} \cdot 2^{n-2} + z_{n-2} \cdot 2^{n-3} + \dots + z_1 \cdot 2^0) + z_0 \quad \Leftrightarrow$$

$$N = 2 \cdot q_1 + z_0$$

$z_0$  jest, dakle, ostatak dijeljenja  $N$  s 2

### Pogledajmo sada količnik $q_1$

$$q_1 = z_{n-1} \cdot 2^{n-2} + z_{n-2} \cdot 2^{n-3} + \dots + z_1 \cdot 2^0$$

Izlučimo li iz svih pribrojnika, osim posljednjeg, zajednički faktor 2:

$$q_1 = 2 \cdot (z_{n-1} \cdot 2^{n-3} + z_{n-2} \cdot 2^{n-4} + \dots) + z_1 \quad \Leftrightarrow$$

$$q_1 = 2 \cdot q_2 + z_1$$

$z_1$  jest, dakle, ostatak  $q_1$  s 2

...

sve dok uzastopnim dijeljenjem s 2 ne postignemo 0.

## Pretvorba dekadskog broja u binarni

$$13_{10} = ?_2$$

$$N = 13 = 2 \cdot q_1 + z_0 \cdot 2^0 = 2 \cdot 6 + 1 \cdot 2^0 \quad \Rightarrow z_0 = 1, q_1 = 6$$

$$q_1 = 6 = 2 \cdot q_2 + z_1 \cdot 2^0 = 2 \cdot 3 + 0 \cdot 2^0 \quad \Rightarrow z_1 = 0, q_2 = 3$$

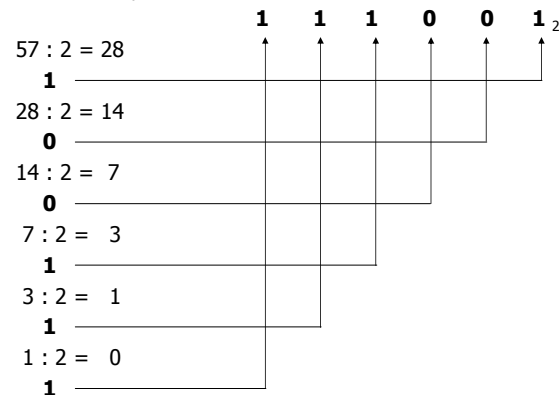
$$q_2 = 3 = 2 \cdot q_3 + z_2 \cdot 2^0 = 2 \cdot 1 + 1 \cdot 2^0 \quad \Rightarrow z_2 = 1, q_3 = 1$$

$$q_3 = 1 = 2 \cdot q_4 + z_3 \cdot 2^0 = 2 \cdot 0 + 1 \cdot 2^0 \quad \Rightarrow z_3 = 1, q_4 = 0$$

$$13_{10} = 1110_2$$

## Pretvorba dekadskog broja u binarni

$$57_{10} = ?_2$$



## Binarni brojevni sustav

- Općenito: najveći dekadski broj s  $d$  znamenaka iznosi  $10^d - 1$ 
  - Primjer:  $d=2$ , najveći broj  $99 = 10^2 - 1$
- Općenito: najveći **binarni** broj s  $b$  znamenaka iznosi  $2^b - 1$ 
  - Primjer:  $b=4$ , najveći broj  $1111 = 2^4 - 1$
- Koliko binarnih znamenaka treba za prikaz dekadskog broja?
  - $10^d - 1 = 2^b - 1 \Rightarrow$
  - $10^d = 2^b \Rightarrow$
  - $d \cdot \log 10 = b \cdot \log 2 \Rightarrow$
  - $d = b \log 2 \Rightarrow$
  - $b = d : \log 2 \Rightarrow$
  - $b = d : 0,30102999566398119521373889472449 \Rightarrow$
  - $b \approx d : 0,3 \Rightarrow b \approx d \cdot (1 : 0,3) \Rightarrow b \approx d \cdot 3,33$
  - Primjer: ako treba prikazati broj  $\leq 99$ ,  $b \approx 2 \cdot 3,33 = 6,66$

## Binarni brojevni sustav

- Koliki se najveći broj može prikazati sa 6 binarnih znamenaka?
  - $111111_2 = 32+16+8+4+2+1 = 63 = 64-1 = 2^6-1$
- Koliki je najveći broj prikazan sa 7 binarnih znamenaka?
  - $1111111_2 = 64+32+16+8+4+2+1 = 127 = 128-1 = 2^7-1$
- Očito, broj znamenaka iz prethodnog izraza treba zaokružiti na viši cijeli broj tj:
  - $b \approx \lceil d \cdot 3,33 \rceil$

## Uvećavanje binarnog broja za 1

- Primjer 1:

$$\begin{array}{r} 100 \\ + 1 \\ \hline 101 \end{array}$$

- Primjer 2:

$$\begin{array}{r} 111 \\ + 1 \\ \hline 1000 \end{array}$$

- Što se događa ako se najveći broj u registru s ograničenim brojem bita uveća za 1?

$$\begin{array}{r} 1111 \\ + 1 \\ \hline 10000 \end{array}$$

Prejev (overflow)

## Negativni binarni brojevi

- Budući da se u registar može pohraniti samo 0 ili 1, za pohranu negativnog predznaka je potreban dogovor (konvencija).
- Jedna od mogućih konvencija *mogla bi biti* npr. postaviti krajnji lijevi bit na 1 ako je broj negativan, ili na 0 ako je broj pozitivan.
- Primjer, 8-bitni registar
  - +24  
0 0 0 1 1 0 0 0
  - 24  
1 0 0 1 1 0 0 0
- Je li predložena konvencija praktična za računanje?

## Negativni binarni brojevi

- Ne, jer bi trebalo ostvariti i zaseban sklop za oduzimanje, različit od sklopa za zbrajanja
- Pokušajmo stoga broj 24 jednostavno komplementirati:

$$\begin{array}{r} +24 \quad 00011000 \\ -24 \quad 11100111 \\ \hline \end{array}$$

i zatim ta dva broja zbrojiti:

$$11111111$$

- Bilo bi dobro da je rezultat 0, ali nije. Međutim, kad bismo sada rezultatu dodali 1, zbog preljeva dobit će se 0.

$$\begin{array}{r} 11111111 \\ + 1 \\ \hline 10000000 \end{array}$$

## Negativni binarni brojevi

- Dakle, negativni brojevi se prikazuju tzv. tehnikom dvojnog komplementa. Nule pretvaramo u jedinice i jedinice u nule (komplement do baze - 1), a zatim tom komplementu dodajemo 1 (komplement do baze - dvojni komplement).

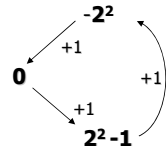
- Primjer: -37 u registru s 8 bita

$$\begin{array}{r} 37 \quad 00100101 \\ \quad 11011010 \\ \hline + \quad 1 \\ -37 \quad 11011011 \\ \hline + \\ 37 \quad 00100101 \\ \hline 10000000 \end{array}$$

## Primjer svih sadržaja u registru od 3 bita (ako je prvi bit predznak)

U registru s 3 bita, ako je prvi bit predznak mogu se prikazati sljedeći brojevi:

Dekadski broj	Binarni broj
0	000
1	001
2	010
3	011
-4	100
-3	101
-2	110
-1	111



Za  $n=3$  dobije se interval  $[-2^2, 2^2 - 1]$ , općenito  $[-2^{n-1}, 2^{n-1} - 1]$

Za  $n=8$  dobije se interval  $[-2^7, 2^7 - 1]$ , tj.  $[-128, 127]$

## Oktalni brojevni sustav

- Baza sustava je **B=8** a znamenke su **0,1,2,3,4,5,6,7**
  - Koristi se za skraćeno zapisivanje binarnih sadržaja kada je to spretno
  - Zapis se može dobiti iz dekadskog uzastopnim dijeljenjem s 8 i zapisivanjem ostataka s desna na lijevo, ali i izravno iz binarnog zapisa:
 
$$N = z_5 \cdot 2^5 + z_4 \cdot 2^4 + z_3 \cdot 2^3 + z_2 \cdot 2^2 + z_1 \cdot 2^1 + z_0 \cdot 2^0$$
 grupiramo li tri po tri pribrojnika i izlučimo zajednički faktor:
 
$$N = (z_5 \cdot 2^2 + z_4 \cdot 2^1 + z_3 \cdot 2^0) \cdot 2^3 + (z_2 \cdot 2^2 + z_1 \cdot 2^1 + z_0 \cdot 2^0) \cdot 2^0$$

$$N = (z_5 \cdot 2^2 + z_4 \cdot 2^1 + z_3 \cdot 2^0) \cdot 8^1 + (z_2 \cdot 2^2 + z_1 \cdot 2^1 + z_0 \cdot 2^0) \cdot 8^0$$

$$\Rightarrow$$

$$o^1 = (z_5 \cdot 2^2 + z_4 \cdot 2^1 + z_3 \cdot 2^0), o^0 = (z_2 \cdot 2^2 + z_1 \cdot 2^1 + z_0 \cdot 2^0)$$

Primjer:

36-bitni broj    001 110 000 101 111 001 010 011 111 000 100 001<sub>2</sub>  
 oktalni ekvivalent    **1 6 0 5 7 1 2 3 7 0 4 1**<sub>8</sub>

## Heksadekadski brojevni sustav

- Baza sustava je **B = 16**, a znamenke su **0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F**
  - Koristi se za skraćeno zapisivanje binarnog sadržaja.
  - Zapis se može dobiti iz dekadskog uzastopnim dijeljenjem s 16 i zapisivanjem ostataka s desna na lijevo, ali i izravno iz binarnog zapisa
- Primjer:

16-bitni broj                    0111 1011 0011 1110<sub>2</sub>  
 heksadekadski ekvivalent    **7 B 3 E**<sub>16</sub>

## Cjelobrojne konstante u jeziku C

- Konstante pisane u dekadskoj notaciji:
 

7 20 64 -110 8092 65535 34567821  
 34567821L -176987941 (pripaziti, 1≠l)
- Konstante pisane u oktalnoj notaciji:
 

07 024 0100 0156 017634 0177777
- Konstante pisane u heksadekadskoj notaciji
 

0x7 0x14 0x40 0x6E 0x1F9C 0xFFFF  
 0xFFFFFFFF

## Cijeli broj s predznakom i bez predznaka

- Kvalifikator `signed` (s predznakom) označava da se u varijabli može pohraniti pozitivna i negativna vrijednost.
- Varijabla deklarirana kao `unsigned` (bez predznaka) može pohraniti samo pozitivne vrijednosti, što udvostručuje maksimalnu pozitivnu vrijednost koja u njoj može biti pohranjena u odnosu na `signed`.

### Primjer:

- Najveći pozitivni broj prikazan u 16-bitnom registru (`signed`):  
 $0111111111111111_2 = 32767_{10}$
- Najveći pozitivni broj prikazan u 16-bitnom registru (`unsigned`):  
 $1111111111111111 = 65535_{10}$

## Cijeli broj s predznakom i bez predznaka

### Primjer:

- Broj **5** prikazan u 16-bitnom registru:  
 $0000000000000101$
- Broj **-5** prikazan metodom dvojnog komplementa u 16-bitnom registru, gdje se vodeća jedinica uvjetno može smatrati predznakom broja (1 → negativan broj):  
 $1111111111111011$
- Ako se isti niz binarnih znamenki pohrani u varijablu cjelobrojnog tipa **bez** predznaka, tada je vodeća jedinica **dio** podatka (**nije** predznak), te je na taj način predstavljen broj 65531

## Cijeli broj s predznakom i bez predznaka

Konstante bez predznaka pišu se s **U** ili **u** na kraju

- u dekadskoj notaciji:  
**7U 20u 64u**
- u oktalnoj notaciji:  
**07u 024U 0100u**
- u heksadekadskoj notaciji:  
**0x7u 0x14u 0xFFFFFFFFu**

## Decimalni brojevi u binarnom sustavu

- Decimalni binarni brojevi sadrže "binarnu točku", analogno decimalnom zarezu, odnosno točki u anglo-američkoj notaciji.
- Primjer prikaza decimalnih brojeva u binarnom sustavu:  
 $5.75_{10} = 5 * 10^0 + 7 * 10^{-1} + 5 * 10^{-2} =$   
 $= 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 1 * 2^{-1} + 1 * 2^{-2} =$   
 $= 101.11_2$

### Općenito:

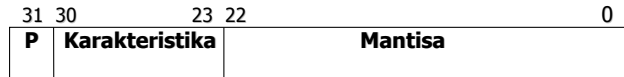
$$R = z_{n-1} \cdot 2^{n-1} + z_{n-2} \cdot 2^{n-2} + \dots + z_1 \cdot 2^1 + z_0 \cdot 2^0 \\ + z_{-1} \cdot 2^{-1} + z_{-2} \cdot 2^{-2} + \dots$$





## Realni brojevi standardne preciznosti

- Tip podatka u programskom jeziku C: `float`
- Koristi se 4 okteta (32 bita)
- Realni broj se pohranjuje u obliku



- P je predznak (P = 1: negativan broj; P = 0: pozitivan broj)
- Karakteristika: binarni eksponent + 127 (time se omogućuje pohrana negativnog eksponenta bez upotrebe tehnike dvojnog komplementa)  
Raspon karakteristike:  $K \in [0, 255]$   
Raspon binarnog eksponenta  $BE \in [-126, 127]$
- Pohranjuje se mantisa iz koje je uklonjena vodeća jedinica (skriveni bit)

## Skriveni bit mantise

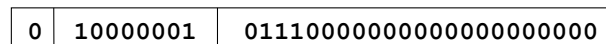
- U binarnom brojevnom sustavu, jedina znamenka koja se u normaliziranom broju (osim za broj 0) može pojaviti ispred decimalne točke je 1 u registru od 8 bitova, moguće je prikazati broj s ukupno 8 znamenaka  
 $1.xxxxxxx$

- Ta jedinica se ne pohranjuje i zato se naziva **skrivenim bitom (hidden bit)**. Na taj se način štedi jedan bit, a time povećava preciznost. u registru od 8 bitova, moguće je prikazati broj s ukupno 9 znamenaka  
 $1.xxxxxxxx$

1 ispred točke se podrazumijeva, stoga ga ne treba pohraniti

## Primjer: Prikazati broj 5.75 kao realni broj

1. Realni dekadski broj prikazati u obliku realnog binarnog broja  
 $5.75_{10} = 101.11_2$
2. Odrediti predznak: broj je pozitivan, stoga je P = 0
3. Normalizirati binarni broj  
 $101.11_2 \cdot 2^0 = 1.0111_2 \cdot 2^2$
4. Izračunati karakteristiku i izraziti ju u binarnom obliku  
 $K = 2_{10} + 127_{10} = 129_{10} = 1000\ 0001_2$
5. Izbaciti vodeću jedinicu iz mantise (skriveni bit)  
Mantisa (bez skrivenog bita i decimalne točke) =  $0111_2$
6. *Prepisati* predznak, karakteristiku i mantisu bez skrivenog bita u registar



**P Karakteristika Mantisa**

0100 0000 1011 1000 0000 0000 0000 0000<sub>2</sub>  
**4 0 B 8 0 0 0 0**<sub>16</sub>

## Primjeri pohrane realnih brojeva

$$2 = 10_2 \cdot 2^0 = 1_2 \cdot 2^1 = 0100\ 0000\ 0000\ 0000 \dots 0000\ 0000 = 4000\ 0000_{16}$$

P = 0, K = 1 + 127 = 128 (10000000), M = (1.) 000 0000 ... 0000 0000

$$-2 = -10_2 \cdot 2^0 = -1_2 \cdot 2^1 = 1100\ 0000\ 0000\ 0000 \dots 0000\ 0000 = C000\ 0000_{16}$$

Jednako kao 2, ali P = 1

$$4 = 100_2 \cdot 2^0 = 1_2 \cdot 2^2 = 0100\ 0000\ 1000\ 0000 \dots 0000\ 0000 = 4080\ 0000_{16}$$

Jednaka mantisa, BE = 2, K = 2 + 127 = 129 (10000001)

$$6 = 110_2 \cdot 2^0 = 1.1_2 \cdot 2^2 = 0100\ 0000\ 1100\ 0000 \dots 0000\ 0000 = 40C0\ 0000_{16}$$

$$1 = 1_2 \cdot 2^0 = 0011\ 1111\ 1000\ 0000 \dots 0000\ 0000 = 3F80\ 0000_{16}$$

K = 0 + 127 (01111111).

$$.75 = 0.11_2 \cdot 2^0 = 1.1_2 \cdot 2^{-1} = 0011\ 1111\ 0100\ 0000 \dots 0000\ 0000 = 3F40\ 0000_{16}$$

## Posebni slučajevi: prikaz broja 0

- Kada bi vodeća znamenka normaliziranog broja uvijek bila 1, ne bi bilo moguće prikazati broj 0
- Koristi se sljedeći dogovor: kada je  $K=0$  i svi bitovi mantise postavljeni na 0, radi se o prikazu realnog broja 0
- U računalu mogu postojati brojevi "+0" i "-0"  
0000 0000 0000 0000 0000 0000 0000 0000 → +0  
1000 0000 0000 0000 0000 0000 0000 0000 → -0
- Međutim, pri usporedbi tih dviju vrijednosti, smatra se da su jednake.

## Posebni slučajevi: denormalizirani broj

- Kada je  $K=0$  i postoje bitovi mantise koji nisu 0, radi se o "denormaliziranom broju". Ne podrazumijeva se skriveni bit, te se smatra da je vodeći bit mantise 0. Vrijednost eksponenta je fiksirana na -126 (ne koristi se izraz  $K=\text{binarni eksponent}+127$ ).

0000 0000 0110 0000 0000 0000 0000 0000

→  $0.11 \cdot 2^{-126}$

0000 0000 0000 0000 0000 0000 0000 1101

→  $0.000\ 0000\ 0000\ 0000\ 0000\ 1101 \cdot 2^{-126}$

- Primijetiti: 0 se može promatrati kao poseban slučaj denormaliziranog broja

0000 0000 0000 0000 0000 0000 0000 0000

→  $0.000\ 0000\ 0000\ 0000\ 0000\ 0000 \cdot 2^{-126}$

## Posebni slučajevi: prikaz $+\infty$ i $-\infty$

- Kada je  $K=255$  i svi bitovi mantise su postavljeni na 0, radi se o prikazu  $+\infty$  ili  $-\infty$ .
- Takvi brojevi se dobiju npr. prilikom dijeljenja s nulom:

```
float x, y, z, w;  
x = 5.;  
y = -5.;  
z = x / 0.;  
w = y / 0.;
```

0111 1111 1000 0000 0000 0000 0000 0000 →  $+\infty$

1111 1111 1000 0000 0000 0000 0000 0000 →  $-\infty$

## Posebni slučajevi: NaN

- Ako je  $K=255$  i postoje bitovi mantise koji nisu 0, radi se o NaN (*not a number*), tj. ne radi se o prikazu broja. NaN je posljedica obavljanja operacije čiji je rezultat nedefiniran ili se prilikom obavljanja operacije dogodila pogreška, npr.

```
float x, y, z;  
x = 0.;  
y = 0.;  
z = x / y;
```

0111 1111 1100 0000 0000 0000 0000 0000 → NaN



## Realni brojevi dvostruke preciznosti

- Tip podatka u programskom jeziku C: `double`
- Koristi se 8 okteta (64 bita)
- Realni broj se pohranjuje u obliku



- P je predznak (P = 1: negativan broj; P = 0: pozitivan broj)
- Karakteristika: binarni eksponent + 1023 (11 bita)  
Raspon karakteristike:  $K \in [0, 2047]$ .  
Raspon binarnog eksponenta  $BE \in [-1022, 1023]$
- Mantisa (52+1 bit).

## Realni brojevi dvostruke preciznosti

### ▪ Posebni slučajevi

- Kada je  $K = 0$  i svi bitovi mantise su nula radi se o broju nula
- Kada je  $K = 0$  i postoje bitovi mantise koji nisu 0, tada se radi o denormaliziranom broju
- Kada je  $K = 2047$  i svi bitovi mantise su 0 radi se o  $+\infty$  ili  $-\infty$
- Kada je  $K = 2047$  i postoje bitovi mantise koji nisu 0, tada se ne radi o prikazu broja (NaN)

## Raspon i preciznost realnih brojeva dvostruke preciznosti

- Najmanji pozitivni broj  $\neq 0$  koji se može prikazati je:  
 $0.0000 \dots 001 \cdot 2^{-1022}$  što je  $4.9406 \cdot 10^{-324}$
- Najveći pozitivni broj koji se može prikazati je:  
 $1.1111 \dots 11111111_2 \cdot 2^{1023} \approx 2^{1024} = 1.797693134862316 \cdot 10^{308}$
- Preciznost pohrane broja s mantisom veličine 53 binarne znamenke je:  
 $2^{53} \approx 10^x \Rightarrow 53 \log 2 \approx x \log 10 \Rightarrow x \approx 53 \log 2 = 15.95458977019$   
(podrazumijeva se **15** prvih važećih znamenki).

## Razlika između preciznosti i točnosti

- Preciznost (*precision*): broj znamenki koji opisuje neku veličinu
- Točnost (*accuracy*): točnost je bliskost stvarnoj (nepoznatoj) vrijednosti
- Za dovoljnu točnost potrebna je adekvatna preciznost, ali preciznost ne implicira automatski točnost jer su iskazane znamenke mogle nastati na temelju npr. pogrešnog mjerenja.

## Tip long double

- U C-u još postoji tip podatka **long double**.
  - Njegova veličina ovisi o platformi, pa se tako mogu naći implementacije u kojima je njegova veličina 64, 80, 96 ili 128 bita.
  - ANSI standard propisuje da nije manji od **double** broja.
- Primjer raspodjele ako je njegova veličina 80 bita:

Karakteristika: 15 bita

Binarni eksponent: Karakteristika – 16383

## Realne konstante

- Primjer realnih konstanti za različite tipove realnih brojeva:

1.	2.34	9e-8	8.345e+25	double
2f	2.34F	-1.34e5f		float
1.L	2.34L	-2.5e-37L		long double

## Veličina osnovnih tipova podataka

- Veličina osnovnih tipova podataka, odnosno količina memorije koju zauzima jedna varijabla osnovnog tipa ovisi o konkretnoj implementaciji prevodioca.

Tip	Veličina
char, unsigned char, signed char	1 okteta
short, unsigned short	2 okteta
int, unsigned int	4 okteta
long, unsigned long	4 okteta
float	4 okteta
double	8 okteta
long double	8 (10, 12, 16) okteta

## Osobitosti C prevodilaca

- Veličina prostora za pohranu podatka određenog tipa nije propisana standardom. Stoga je programeru na raspolaganju operator **sizeof**, koji za zadani operand izračunava veličinu prostora za pohranu izraženu u oktetima.
- Operand može biti naziv tipa podatka ili naziv varijable
- Primjer:

```
int var;  
printf("%d",sizeof(char)); // ispisuje 1  
printf("%d",sizeof(var)); // ispisuje 4
```

## Rasponi različitih tipova cijelih brojeva

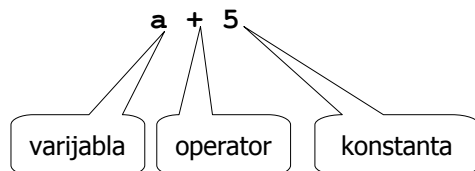
Deklaracija	Broj bita	Interval brojeva
<b>int</b>	32	[-2147483648, 2147483647]
<b>signed int</b>	32	[-2147483648, 2147483647]
<b>short</b>	16	[-32768, 32767]
<b>short int</b>	16	[-32768, 32767]
<b>signed short int</b>	16	[-32768, 32767]
<b>unsigned int</b>	32	[0U, 4294967295U]
<b>unsigned short int</b>	16	[0U, 65535U]
<b>long</b>	32	[-2147483648, 2147483647]
<b>long int</b>	32	[-2147483648, 2147483647]
<b>signed long int</b>	32	[-2147483648, 2147483647]
<b>unsigned long int</b>	32	[0U, 4294967295U]

## Operatori i izrazi

## Izrazi

- Izraz jest kombinacija operatora, operanada (konstante, varijable, ...) i zagrada, koja po evaluaciji daje rezultat. Može biti dio većeg izraza.

- Primjer:



- Primjer: `(b + c) / ((d + e) * 4)`

## Pridruživanje vrijednosti varijablama

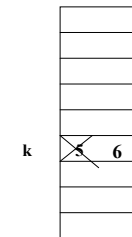
- Pridruživanje vrijednosti:

- simbol u pseudokodu `:=`
- u C-u `=`

- Na primjer:

- `k := 5`  $\Leftrightarrow$  `k = 5;`

- Što znači naredba ? `k = k + 1;`



**Primjer pridruživanja:** Zadano je  $X=14.5$  i  $Y=-9.9$ . U programu treba ispisati vrijednosti X i Y, a zatim u X staviti vrijednost od Y, a u Y staviti vrijednost od X. Ispisati ponovno X i Y.

```

ZamjenaVarijabli
int main () {
    float x, y, p;
    x= 14.5;
    y= -9.9;
    printf ("x=%f, y=%f\n", x, y);
    /* zamijeniti vrijednosti x i y */
    p=x;
    x=y;
    y=p;
    printf ("Nakon zamjene: x=%f, y=%f\n", x, y);
    return 0;
}

```

	x	y	p
	???	???	???
x= 14.5;	14.5	???	???
y= -9.9;	14.5	-9.9	???
p=x;	14.5	-9.9	14.5
x=y;	-9.9	-9.9	14.5
y=p;	-9.9	14.5	14.5

```

x=14.500000, y=-9.900000
Nakon zamjene: x=-9.900000, y=14.500000

```

## Osnovni aritmetički operatori

### Operator Značenje

+	zbrajanje
-	oduzimanje
*	množenje
/	dijeljenje
%	ostatak kod cjelobrojnog dijeljenja (modulo, modulus)

## Djelovanje aritmetičkih operatora na cjelobrojne operande

```

int a, b;
a = 10; b = 3;

```

<u>Izraz</u>	<u>Rezultat</u>
a + b	13
a - b	7
a * b	30
a / b	3
a % b	1

## Djelovanje aritmetičkih operatora na realne operande

```

float a, b;
a = 12.5; b = 2.;

```

<u>Izraz</u>	<u>Rezultat</u>
a + b	14.5
a - b	10.5
a * b	25.0
a / b	6.25
a % b	pogreška

## Aritmetika s različitim tipovima operanada

```
int i;
float f;
i = 2;
f = 2.99;
```

Što je rezultat operacije  $i + f$

4 ili 4.99

Kada su operandi različitog tipa, prije obavljanja operacije obavlja se implicitna (automatska) pretvorba tipa rezultata u "veći (važniji)" od tipova operanada koji sudjeluju u operaciji.

U prikazanom primjeru, prije nego se obavi operacija zbrajanja, vrijednost koja se nalazi u varijabli  $i$  pretvara se u 2.0 (pri tome sadržaj varijable  $i$  ostaje nepromijenjen).

## Implicitna (automatska) pretvorba tipova podataka

Pretvorba tipova podataka u izrazima obavlja se prema jednom od sljedećih 5 pravila. Treba iskoristiti prvo po redu pravilo koje se može primijeniti na konkretan slučaj!

1. Ako je jedan od operanada tipa **long double**, preostali operand se pretvara u tip **long double**
2. Ako je jedan od operanada tipa **double**, preostali operand se pretvara u tip **double**
3. Ako je jedan od operanada tipa **float**, preostali operand se pretvara u tip **float**
4. Ako je jedan od operanada tipa **long**, preostali operand se pretvara u tip **long**
5. Operande tipa **short** i **char** pretvoriti u tip **int**

Kada se u izrazima pojavljuju *unsigned* tipovi, pravila pretvorbe su složenija i ovise o implementaciji. Zato se ovdje neće razmatrati.

## Primjeri: implicitna pretvorba tipova podataka

```
char c;      int i;      float f;      long double ld;
short s;     long li;     double d;
```

Operacija	Pretvorba tipa prije obavljanja operacije	Prema pravilu
$ld * i$	sadržaj od $i \rightarrow$ long double	1.
$c \% i$	sadržaj od $c \rightarrow$ int	5.
$s / f$	sadržaj od $s \rightarrow$ float	3.
$f - ld$	sadržaj od $f \rightarrow$ long double	1.
$li \% i$	sadržaj od $i \rightarrow$ long	4.
$i * f$	sadržaj od $i \rightarrow$ float	3.
$d + c$	sadržaj od $c \rightarrow$ double	2.
$s - c$	sadržaj od $s \rightarrow$ int    sadržaj od $c \rightarrow$ int	5.

## Primjeri: implicitna pretvorba tipova podataka

```
float f1, f2;
int i1, i2;          /* pretpostavka int: 4 okteta */
char c1, c2;
f1 = 32000.5; f2 = 1.0;
i1 = 2147483647; i2 = 1;
c1 = 127; c2 = 1;
```

Izraz	Rezultat	tip rezultata
$f1 + f2$	32001.5	float
$f1 + i2$	32001.5	float
$i1 + i2$	-2147483648	int
$i1 + f2$	2147483648.0	float
$c1 + c2$	128	int

Kako to da ima 10 točnih znamenaka?

## Pretvorba tipova kod pridruživanja

- Vrijednost s desne strane pretvara se u tip podatka varijable ili izraza s lijeve strane znaka za pridruživanje
- Primjer:

```
int i;  
float f;  
i = 2.75; /* 2.75 se pretvara u int */  
f = 2147483638;
```
- Potrebno je obratiti pozornost da se pri promjeni tipa podatka može "izgubiti" manji ili veći dio informacije. U gornjem primjeru:
  - "izgubljene" su decimale 0.75 kod prvog pridruživanja, tj. ispisom vrijednosti varijable `i` dobilo bi se 2
  - `f` sadrži vrijednost za 10 veću od one koja je pridružena, tj. ispisom vrijednosti varijable `f` dobilo bi se 2147483648.

## EksPLICITNA (ZADANA) pretvorba tipa podataka

- Opći oblik zadane (eksplicitne) pretvorbe (eng. *cast operator*) glasi:

`(tip_podatka) operand`

- Operand može biti varijabla, konstanta ili izraz.
- Zadana pretvorba podataka ima viši prioritet od automatske.

- Primjer:

```
int i = 2000000000;  
double d1, d2;  
d1 = 2 * i;  
d2 = 2 * (double)i;
```

## Cjelobrojno dijeljenje

- Potrebno je obratiti pozornost na "neželjene" rezultate kod cjelobrojnog dijeljenja. Ukoliko se na primjer u realnu varijablu `a` želi pridružiti vrijednost  $\frac{1}{2}$ , sljedeća naredba pridruživanja **neće** varijabli `a` pridružiti vrijednost 0.5:

```
a = 1 / 2;
```
- U izrazu s desne strane jednakosti **oba su operanda cjelobrojnog tipa**, pa će se obaviti cjelobrojno dijeljenje. Rezultat tog izraza je 0 (uz ostatak 1).
- Za izbjegavanje cjelobrojnog dijeljenja potrebno je koristiti zadanu pretvorbu tipa (dovoljno samo na jednom operandu) ili zadati konstante tako da je barem jedna realna:

```
a = (float) 1 / 2; ili  
a = 1. / 2; ili  
a = 1 / 2.;
```

Napomena: U prvom slučaju korištena je zadana pretvorba tipa operanda (mogla se uporabiti i nad drugim operandom).

U drugom i trećem slučaju, dodavanjem točke, konstanta je postala realna te se dijeljenje obavlja u realnoj domeni.

## Cjelobrojno dijeljenje

- Primjer: Koliko iznosi:

a)  $9 / 4$       Rješenje: 2 (cjelobrojno dijeljenje)

b)  $9 \% 4$       Rješenje: 1 (ostatak cjelobrojnog dijeljenja  $9 : 4 = 2$  i ostatak 1)

## Prioritet osnovnih aritmetičkih operatora

---

1. \* / %
2. + -

Ako u izrazu ima više operatora jednakog prioriteta, izračunavaju se *slijeva nadesno*. Na primjer:

$$\begin{array}{r} 2 + \underbrace{3 / 2} * 4 - \underbrace{5 * 6 \% 8} \\ 2 + \underbrace{1 * 4} - \underbrace{30 \% 8} \\ \underbrace{2 + 4} - 6 \\ 6 - 6 \\ \underbrace{\phantom{6 - 6}}_0 \end{array}$$

## Korištenje okruglih zagrada

---

Kada treba koristiti okrugle zagrade ?

- a) ako se želi promijeniti ugrađeni redoslijed izvođenja operacija
- b) u slučaju vlastite nesigurnosti
- c) radi bolje čitljivosti programa

$$2 + 3 / (2 * 4) - 5 * (6 \% 8) \Rightarrow -28$$

## Primjer

---

- Primjer: Koliki je rezultat sljedećeg izraza:

$$\begin{array}{l} 5 + 10 / 3 * (8 - 6) \\ 5 + 10 / 3 * 2 \\ 5 + 3 * 2 \\ 5 + 6 \\ 11 \end{array}$$

## Primjer

---

- Primjer: Koju će vrijednost poprimiti varijable i, x, c

```
int i;  
double x, c, d;
```

nakon naredbi:

```
d = 6.0;  
i = (int)(d + 1.73);  
x = i / 2;  
c = (double)i / 2;
```

- Rješenje:

```
i = 7, x = 3, c = 3.5
```

Primjer: Učitati vrijednosti za cjelobrojne varijable i, j i k te ispisati njihove vrijednosti i aritmetičku sredinu.

▪ Struktogram

početak
definiraj cjelobrojne varijable i, j, k
definiraj realnu varijablu sredin
pročitaj vrijednosti i, j, k
$sredin = (i + j + k) / 3$
ispiši("Aritm. sredina je x.xxxxxx", sredin);
svršetak

## Rješenje primjera u C-u

📄 AritmetickaSredina

```
#include <stdio.h>
int main () {
    int i, j, k;
    float sredin;
    scanf ("%d %d %d", &i, &j, &k);
    sredin = (i + j + k) / 3.;
    printf ("Aritm. sredina je %f", sredin);
    return 0;
}
```

## Rezultat izvođenja programa

Ulazni podaci:

1 2 4

Ispis na zaslonu:

```
Aritm. sredina je 2.333333
```

## Primjer: Površina kruga

▪ Pseudokod

```
početak
definiraj PI=3.14
definiraj realne varijable radijus, povrsina
ispiši ("Zadajte radijus kruga. . . ")
učitaj (radijus)
povrsina = (radijus * radijus) * PI
ispiši ("Površina kruga radijusa x.xxx je x.xxx", radijus, povrsina)
kraj
```

## Primjer: Površina kruga

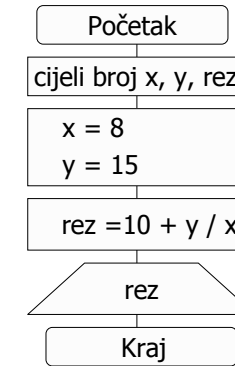
PovrsinaKrugJednostavna

```
#include <stdio.h>
/* primjer koristenja simbolickih konstanti */
#define PI 3.14 /*definicija simbolicke konstante*/

int main ( ) {
    float radijus, povrsina;
    printf("Zadajte radijus kruga >");
    scanf("%f", &radijus);
    povrsina = (radijus * radijus) * PI;
    printf("Povrsina kruga radijusa %.3f je %.3f\n",
        radijus, povrsina);
    return 0;
}
```

## Primjer: Dijeljenje dva cijela broja

▪ Dijagram toka



## Primjer: Dijeljenje dva cijela broja

DijeljenjeDvaCijelaBroja

```
#include <stdio.h>
/* primjer zadavanja cjelobrojnih konstanti*/
int main ( ) {
    int x, y, rez;
    /*oktalno zadan cijeli broj*/
    x = 010;
    /*heksadekadski zadan cijeli broj*/
    y = 0xF;
    rez = 10 + y / x;
    printf("Rezultat = %d\n", rez);
    return 0;
}
```