Uvod u programiranje

- predavanja -

listopad 2020.

Tipovi podataka u programskom jeziku C

- 2. dio -

Osnovni tipovi podataka

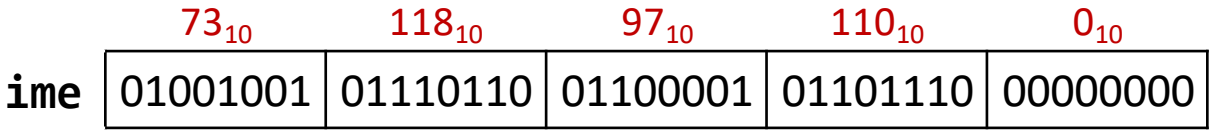
Digresija: nizovi znakova

Niz znakova - *string*

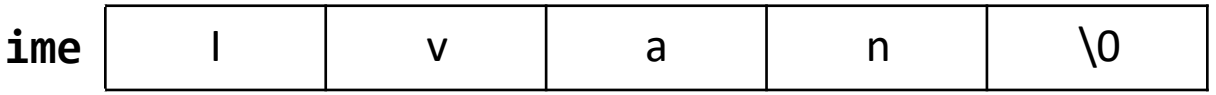
- U programskom jeziku C ne postoji osnovni tip podatka koji podržava rad s nizovima znakova
- za pohranu niza znakova koristi se jednodimenzijsko polje čiji su članovi tipa char, pri čemu se kraj niza obavezno označava članom polja koji sadrži nul-znak '\0' (kod iz ASCII tablice numeričke vrijednosti 0)

```
char ime[5];

ime[0] = 'I';
ime[1] = 'v';
ime[2] = 'a';
ime[3] = 'n';
ime[4] = '\0';



printf("%s", ime);
```

Ivan



- radi pojednostavljenja, na slikama će se umjesto kodova znakova prikazivati znakovi

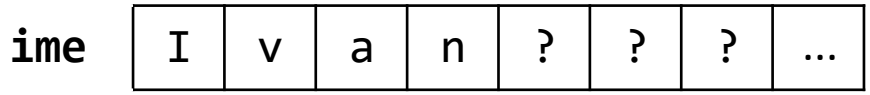


- kao konverzijsku specifikaciju koristiti %s
- kao vrijednost koju treba ispisati koristiti naziv polja u kojem je niz znakova pohranjen

Čemu služi terminator niza '\0' ?

- Pomoću znaka '\0' može se zaključiti gdje se nalazi kraj niza znakova. Npr.
  - funkcija printf prema konverzijskoj specifikaciji %s ispisuje znakove iz zadanog niza znakova, znak po znak, sve dok ne dođe do člana polja u kojem se nalazi vrijednost '\0'
  - ako kraj niza nije ispravno označen znakom '\0', funkcija će nastaviti ispisivati znakove čiji ASCII kodovi odgovaraju vrijednostima koji se nalaze u memoriji iza kraja niza

```
char ime[4];

ime[0] = 'I';
ime[1] = 'v';
ime[2] = 'a';
ime[3] = 'n';

printf("%s", ime);
```



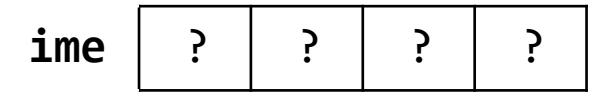
- ispis će se nastaviti sve dok se negdje u memoriji ne naiđe na bajt u kojem je upisana vrijednost nula

```
Ivan*)%&/!)=()Z)(B#DW=)$/")#*'@!"/...
```

Definicija niza znakova uz inicijalizaciju

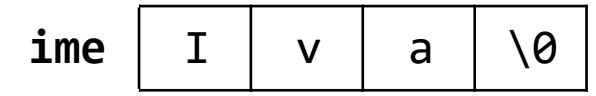
- Jednako kao kod polja ostalih tipova podataka:
  - bez inicijalizacije, članovi polja sadrže nedefinirane vrijednosti

```
char ime[4];
```



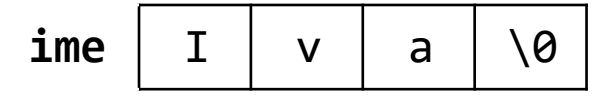
  - početne vrijednosti se mogu redom navesti u tzv. inicijalizatoru

```
char ime[4] = {'I', 'v', 'a', '\0'};
```



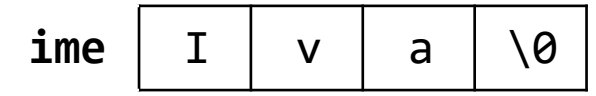
  - ili

```
char ime[] = {'I', 'v', 'a', '\0'};
```



  - ili

```
char ime[4] = {'I', 'v', 'a'};
```



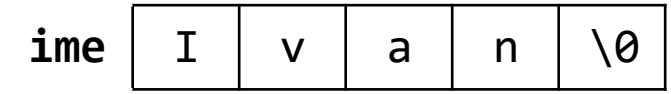
Definicija niza znakova uz inicijalizaciju

- Obratiti pažnju
  - ovakav način inicijalizacije jednodimenzijskog polja nije ispravan ako se njegov sadržaj namjerava koristiti kao niz znakova (string)

```
char ime[4] = {'I', 'v', 'a', 'n'};
```

  - ispravno:

```
char ime[4+1] = {'I', 'v', 'a', 'n'};
```



Jednostavniji način inicijalizacije niza znakova

- Umjesto:

```
char ime[4] = {'I', 'v', 'a', '\0'};
```

  - može se (i bolje je) koristiti drugačiji oblik inicijalizatora

```
char ime[4] = "Iva";
```

znak **\0** će biti dodan, programer **mora** osigurati prostor za barem jedan znak više!

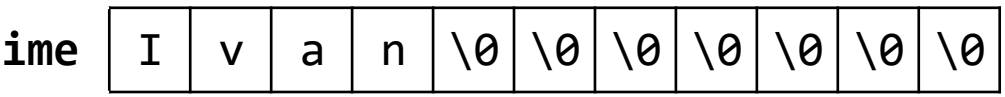
  - ili

```
char ime[] = "Iva";
```

potrebnu veličinu polja odredit će prevodilac, znak **\0** će biti dodan

  - uobičajeno se nizovi znakova definiraju uz pomoć simboličkih konstanti. Pri tome, dobra je praksa koristiti notaciju "+ 1". Time se naglašava da nije zaboravljen prostor za znak kojim se terminira niz

```
#define MAX_IME 10
char ime[MAX_IME + 1] = "Ivan";
```



Konstantni znakovni niz

- U programu se konstantni znakovni niz označava dvostrukim navodnicima

inicijalizator. Ovo nije naredba za pridruživanje!

```
#define MAX_IME 5
char ime[MAX_IME + 1] = "Iva";
printf("%s\n", ime);
printf("%s\n", "Marko");
```

ispis niza znakova pohranjenog u varijabli

ispis konstantnog znakovnog niza

konstantni znakovni niz

```
Iva↵
Marko↵
```

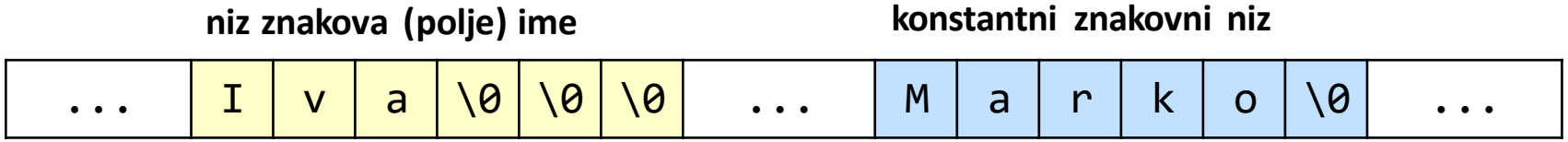
```
char ime[10 + 1];
ime = "Iva";
```

Nije dopušteno! Polje je ***non-modifiable*** *lvalue*!

Konstantni znakovni niz

```
#define MAX_IME 5
char ime[MAX_IME + 1] = "Iva";
printf("%s\n", ime);
printf("%s\n", "Marko");
```

- konstantni znakovni niz "Marko" u memoriji je pohranjen slično nizu znakova ime



- ali postoji bitna razlika: sadržaj niza znakova može se mijenjati

```
ime[1] = 'd';
printf("%s\n", ime);
```

```
Ida↵
```

Konstantni znakovni niz

- Konstantni znakovni niz se u programima ne smije lomiti kroz više redaka

```
printf("%s", "Fakultet
 u Unskoj 3");
```

prevodilac dojavljuje pogrešku

- ako je niz predugačak da bi se mogao pregledno napisati u jednom retku, treba koristiti jednostavno nadovezivanje

```
printf("%s", "Fakultet"
             " u Unskoj 3");
```

- konstantni znakovni niz napisan u dva ili više dijelova, u memoriji će biti pohranjen na isti način kao da je napisan u jednom dijelu

```
printf("%s", "Fakultet u Unskoj 3");
```

...	F	a	k	u	l	t	e	t		u		U	n	s	k	o	j		3	\0	...

Konverzijska specifikacija %s za printf

- Za ispis niza znakova (također i konstantnog znakovnog niza) koristi se konverzijska specifikacija %s

```
printf("Ime: %s\nPrezime: %s\nAdresa:\n%s\n",
       "Ana",
       "Novak",
       "Ilica 1\n10000 Zagreb\\Centar");
```

```
Ime: Ana↵
Prezime: Novak↵
Adresa:↵
Ilica 1↵
10000 Zagreb\Centar↵
```

Konverzijska specifikacija %s za scanf

- Konverzijska specifikacija %s omogućuje čitanje niza znakova do pojave prve praznine, oznake novog retka ili tabulatora

```
char ime[10 + 1], prezime[10 + 1];
scanf("%s %s", ime, prezime);
printf("%s, %s", prezime, ime);
```

```
Anamarija Horvat Novak↵
Horvat, Anamarija
```

- Čitanje konverzijskom specifikacijom %s na kraj niza ugrađuje '\0'
- Čitanje niza znakova pomoću funkcije scanf i konverzijske specifikacije %s je potencijalno opasno
  - što će se dogoditi ako korisnik upiše predugo ime ili prezime (u odnosu na duljinu niza znakova navedenu u definiciji)

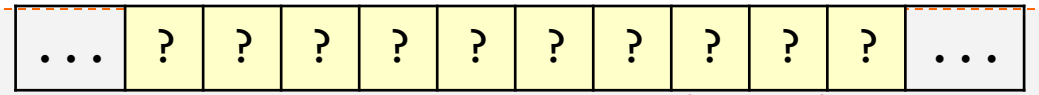
```
Anamarija Horvat-Novak↵
```

Učitavanje niza znakova na siguran način

- Ako je potrebno onemogućiti unos predugog niza znakova ili je potrebno omogućiti čitanje niza znakova koji sadrži praznine ili tabulatore, tada za čitanje niza znakova s tipkovnice umjesto funkcije scanf treba koristiti funkciju fgets
  - funkcija fgets će detaljnije biti objašnjena kasnije. Ovdje je naveden skraćeni opis, koji će biti dovoljan za njeno ispravno korištenje
  - funkcija fgets prima tri argumenta
    - ime polja (varijabla tipa niza znakova) `niz` u koje s tipkovnice treba učitati niz znakova
    - najveći dopušteni broj znakova n koje funkcija fgets smije upisati u to polje (za vrijeme dok učitava znakove s tipkovnice)
    - *tok podataka* iz kojeg se učitavaju znakovi. Za sada uvijek napisati `stdin` (skraćenica dolazi od *standard input* - vrlo pojednostavljeno: tipkovnica). Detaljnije objašnjenje slijedi u poglavlju o datotekama

Učitavanje niza znakova

```
char tekst[10];
fgets(tekst, 10, stdin);
```



- u zadani niz učitava znakove s tipkovnice sve dok ne pročita oznaku novog retka ili učita n-1 znakova (u ovom primjeru 9). Iza zadnjeg učitanog znaka u niz dodaje znak '\0'
  - zašto se ovoj funkciji mora zadati najveći dopušteni broj znakova?

↵	... \n \0 ? ? ? ? ? ? ? ? ...
Pas↵	... P a s \n \0 ? ? ? ? ? ...
Pet pasa↵	... P e t p a s a \n \0 ...
Dva macka↵	... D v a m a c k a \0 ...
Jedanaest pasa↵	... J e d a n a e s t \0 ...

Primjer

- Programski zadatak
  - s tipkovnice učitati niz znakova iz jednog retka. Niz znakova, uključujući oznaku novog retka (ako bude učitana), ne smije biti dulji od 10 znakova.
  - ako učitani niz sadrži znak za prelaz u novi redak, izbaciti ga iz niza. Sva mala slova u nizu pretvoriti u velika. Ispisati novi sadržaj niza i odmah iza kraja niza uskličnik.

```
Upisite niz znakova > Kvaka 22↵
KVAKA 22!
```

```
Upisite niz znakova > Put u svemir↵
PUT U SVEM!
```

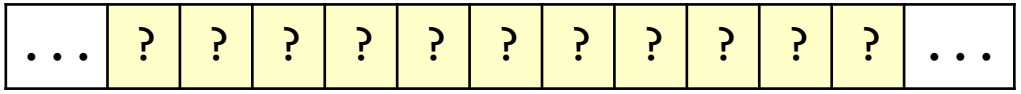
```
Upisite niz znakova > ↵
!
```

Rješenje

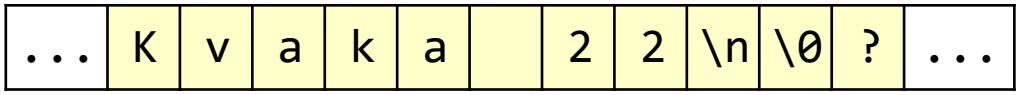
```
#include <stdio.h>
#define MAX_NIZ 10

int main(void) {
   char niz[MAX_NIZ + 1];
   int i = 0;
   printf("Upisite niz znakova > ");
   fgets(niz, MAX_NIZ + 1, stdin);
   while (niz[i] != '\0') {
      if (niz[i] >= 'a' && niz[i] <= 'z') {
         niz[i] = niz[i] - ('a' - 'A');
      } else if (niz[i] == '\n') {
         niz[i] = '\0';
      }
      i = i + 1;
   }
   printf("%s!", niz);
   return 0;
}
```

Objašnjenje

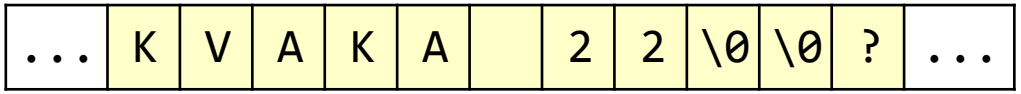
```
char niz[10];
```



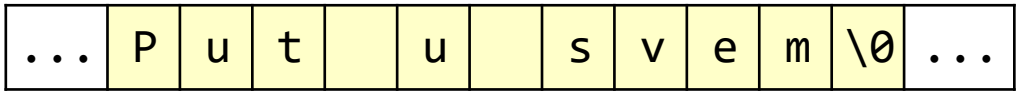

```
fgets(niz, 10, stdin);
```



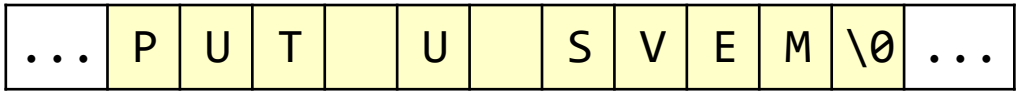
nakon petlje while



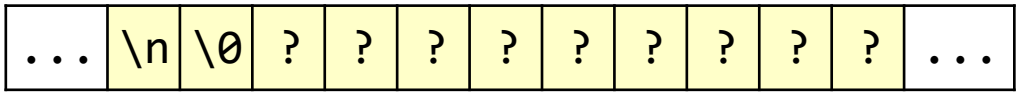

```
fgets(niz, 10, stdin);
```



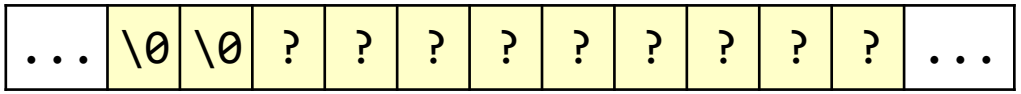
nakon petlje while




```
fgets(niz, 10, stdin);
```



nakon petlje while



Cjelobrojni tipovi podataka

Tip podatka _Bool

Tip podatka _Bool

- cjelobrojni tip podatka koji se koristi za pohranu logičke vrijednosti *istina* ili *laž*
  - interno se vrijednost u varijablu tipa _Bool pohranjuje kao cijeli broj
    - logička vrijednost *istina* predstavljena je cjelobrojnom vrijednošću 1, a logička vrijednost *laž* cjelobrojnom vrijednošću 0
    - npr. za gcc i arhitekturu x86_64, koristi se prostor veličine 1 bajt
  - po čemu se tip _Bool razlikuje od ostalih cjelobrojnih tipova?
    - ako se varijabli tipa _Bool pridruži bilo koja numerička vrijednost različita od nule, varijabla će poprimiti vrijednost 1

```
_Bool padaKisa;
padaKisa = 37;
printf("%hhd", padaKisa);
```

```
1
```

Logičke vrijednosti u programskom jeziku C

- svaki podatak, bilo kojeg tipa, u programskom jeziku C može se (ne znači da je poželjno) koristiti kao logička vrijednost. To znači da svaka vrijednost koja je numerički
  - jednaka nuli odgovara logičkoj vrijednosti *laž*
  - različita od nule odgovara logičkoj vrijednosti *istina*
- sljedeći primjer nije smjernica kako treba pisati programe (upravo suprotno) nego ilustracija prethodno navedenog

ispravno, ali nepotrebno nejasno

```
float a, b;
scanf("%f %f", &a, &b);
if (a + b) {
   printf("Zbroj nije nula");
}
```

ispravno i jasno

```
float a, b;
scanf("%f %f", &a, &b);
if (a + b != 0.f) {
   printf("Zbroj nije nula");
}
```

Rezultat logičkih izraza

- u programskom jeziku C rezultat logičkih izraza jest uvijek cjelobrojna vrijednost 0 (ako je rezultat *laž*) ili cjelobrojna vrijednost 1 (ako je rezultat *istina*). Rezultat po tipu odgovara tipu podatka int

```
float x = 5.0f;
printf("%d\n", x > 10.f || x < 0.f);
printf("%d", x != 20.f);
```

```
0
1
```

- to što je rezultat logičkog izraza cijeli broj 0 ili 1, nije opravdanje za pisanje ovakvog koda:

VRLO LOŠE!

```
if (x != 20.0 == 1)
```

Rezultat logičkih izraza

- rezultat logičkog izraza može se pohraniti u varijablu tipa _Bool, a potom koristiti u daljnjim logičkim izrazima
- Sljedeći primjer je ilustracija, a ne smjernica kako treba pisati

```
float x;
_Bool x_je_veci_od_10, x_je_manji_od_20;

scanf("%f", &x);
x_je_veci_od_10 = x > 10.f;
x_je_manji_od_20 = x < 20.f;

if (x_je_veci_od_10 && x_je_manji_od_20)
   printf("x je iz intervala <10, 20>");
```

VRLO LOŠE!

```
if (x_je_veci_od_10 == 1 && x_je_manji_od_20 == 1)

if ((x_je_veci_od_10 && x_je_manji_od_20) == 1)
```

Primjer

- unapređenje primjera s prim brojevima

```
...
int i, n, djeljiv = 0;
_Bool djeljiv = 0;                          // hipoteza: nije djeljiv
printf("Upisite prirodni broj > ");
scanf("%d", &n);
i = 2;
while (i <= n - 1 && djeljiv == 0 !djeljiv) {
   if (n % i == 0) {
      djeljiv = 1;                          // hipoteza je bila pogresna
   }
   i = i + 1;
}
if (djeljiv == 1 djeljiv || n == 1)         // jer broj 1 je specijalan slucaj
   printf("%d nije prim broj\n", n);
else
   ...
```

Zamjenski *naziv* za tip *_Bool*, konstante *false* i *true*

- u `<stdbool.h>` se nalaze:
  - *makro* definicija *bool*
    - omogućuje da se umjesto naziva tipa _Bool koristi zamjenski naziv bool
    - naziv *bool* prikladniji je od *_Bool* jer je više u duhu jezika C: ostali ugrađeni tipovi podataka, npr. tipovi int, float, itd. nemaju u svom nazivu znak _ i ne sadrže velika slova
  - *makro* definicije *true* i *false*
    - omogućuju korištenje simboličkih konstanti *true* i *false*, umjesto cjelobrojnih konstanti 0 i 1
    - povećava se jasnoća programa

Makro definicije će biti detaljno objašnjene u kasnijim predavanjima. Za sada je dovoljno znati da se nakon uključivanja datoteke <stdbool.h>, u programu mogu koristiti nazivi *bool*, *true* i *false* kao zamjena za *_Bool*, *1* i *0*.

Primjer

- unapređenje primjera s prim brojevima

```
#include <stdbool.h>
...
int i, n;
_Bool djeljiv = 0;
bool djeljiv = false;                    // hipoteza: nije djeljiv

printf("Upisite prirodni broj > ");
scanf("%d", &n);
i = 2;
while (i <= n - 1 && djeljiv == 0 !djeljiv) {
   if (n % i == 0) {
      djeljiv = 1 true;                  // hipoteza je bila pogresna
   }
...
```

Realni tipovi podataka

Tip podatka float

Tip podatka float

- Primjer definicije varijable

```
float temperatura;
```

- Primjeri konstanti tipa float

```
2f                  = 2.0
-2.34F              = -2.34
-1.34e5F            = -1.34 · 10^5
9.1093E-31f         = 9.1093 · 10^-31
```

- Na koji način su u računalu pohranjene vrijednosti tipa float?

Binarni razlomci

- racionalni broj **q** je *decimalni* razlomak ako se može zapisati u obliku

$$\mathbf{q} = \pm (c_n \cdot 10^n + c_{n-1} \cdot 10^{n-1} + \ldots c_1 \cdot 10^1 + c_0 \cdot 10^0 + r_1 \cdot 10^{-1} + r_2 \cdot 10^{-2} + \ldots + r_m \cdot 10^{-m})$$

$$c_i \in \{0, 1, 2, \ldots, 9\}, i = 0, \ldots, n$$

$$r_j \in \{0, 1, 2, \ldots, 9\}, j = 1, \ldots, m$$

$$13.75_{10} = 1 \cdot 10^1 + 3 \cdot 10^0 + 7 \cdot 10^{-1} + 5 \cdot 10^{-2}$$

- slično, *binarni* razlomak **q** može se zapisati u obliku

$$\mathbf{q} = \pm (c_n \cdot 2^n + c_{n-1} \cdot 2^{n-1} + \ldots c_1 \cdot 2^1 + c_0 \cdot 2^0 + r_1 \cdot 2^{-1} + r_2 \cdot 2^{-2} + \ldots + r_m \cdot 2^{-m})$$

$$c_i \in \{0, 1\}, i = 0, \ldots, n$$

$$r_j \in \{0, 1\}, j = 1, \ldots, m$$

$$1101.11_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2}$$

Pretvaranje decimalnog u binarni razlomak

- cijeli dio (dio ispred decimalne točke) pretvara se u cijeli dio (dio ispred binarne točke) binarnog razlomka uzastopnim dijeljenjem brojem 2
- razlomljeni dio (dio iza decimalne točke) pretvara se u razlomljeni dio binarnog razlomka uzastopnim množenjem brojem 2. Prekida se kad se za razlomljeni dio dobije točno nula
  - $9.625_{10} = ?_2$

    $9_{10} = 1001_2$

    $0.625 \cdot 2 = \mathbf{1}.250$

    $0.250 \cdot 2 = \mathbf{0}.5$

    $0.5 \cdot 2 = \mathbf{1}.0$ ← kraj

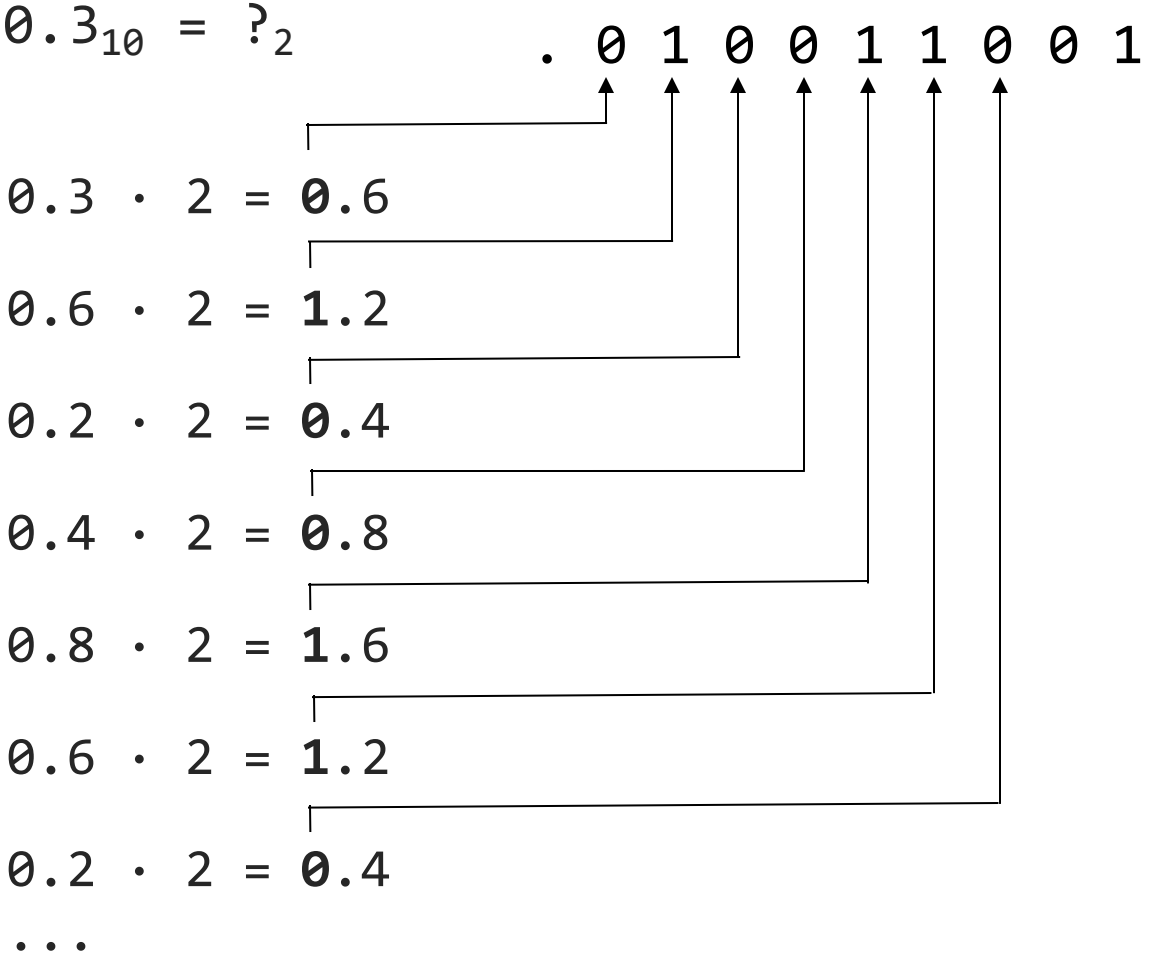
    . 1 0 1
  - $9.625_{10} = 1001.101_2$

Pretvaranje decimalnog u binarni razlomak

- ako nazivnik korespondentnog racionalnog broja sadrži faktor koji nije potencija broja 2, tada se decimalni razlomak neće moći prikazati kao binarni broj s konačnim brojem znamenaka. Npr.
  - konačnim brojem binarnih znamenaka mogu se prikazati brojevi
    - 1/2, 1/4, 1/8, 1/16, ..., 3/2, 3/4, 3/8, ..., 5/2, 5/4, 5/8, ...
  - konačnim brojem binarnih znamenaka ne mogu se prikazati brojevi
    - 1/3, 1/5, 1/6, 1/7, 1/9, 1/10, ..., 2/3, 2/5, 2/7, ...
- također očito: ako realni broj nije decimalni razlomak (ne može se prikazati s konačnim brojem znamenaka iza decimalne točke) tada se neće moći prikazati niti kao binarni broj s konačnim brojem znamenaka

Pretvaranje decimalnog u binarni razlomak

- $3/10 = 0.3_{10} = ?_2$



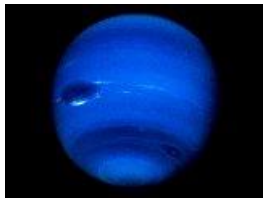
dok se ne dosegne zadovoljavajuća ili moguća preciznost

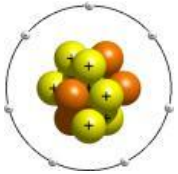
Množenje binarnog broja s potencijama broja 2

- binarni broj se množi s potencijama baze 2 tako da se binarna točka pomakne odgovarajući broj mjesta desno ili lijevo, ovisno o tome je li predznak potencije pozitivan ili negativan
- Primjer:
  - $111.000 \cdot 2^{2} = 11100.0$
  - $0001.101 \cdot 2^{-3} = 0.001101$

Prikaz vrlo velikih i vrlo malih brojeva

- Kako inženjeri i znanstvenici prikazuju vrlo velike i vrlo male brojeve?
  - Kolika je prosječna udaljenost Neptuna i Sunca?
    - 450393**0000000** m
  - Koliko iznosi masa elektrona?
    - 0.**000000000000000000000000000000**910938188 kg
  - previše znamenaka (nula) troši se samo za prikaz magnitude broja





- zato se koristi znanstvena notacija: decimalni broj s jednom znamenkom ispred decimalne točke (zareza), pomnožen odgovarajućom potencijom broja 10 (jer je baza brojanja = 10)
  - $4.50393 \cdot 10^{12}$
  - $9.10938188 \cdot 10^{-31}$

mantisa

eksponent

Prikaz vrlo velikih i vrlo malih binarnih brojeva

- Slično, binarni razlomak se može prikazati kao binarni broj s jednom binarnom znamenkom ispred binarne točke, pomnožen odgovarajućom potencijom broja 2 (jer je baza brojanja = 2)
- Za broj u takvom obliku kaže se da je normaliziran. Npr.

$$101.11 = 1.0111 \cdot 2^{2}$$

$$0.0000000000010011 = 1.0011 \cdot 2^{-14}$$

binarna mantisa

binarni eksponent

- Normalizacija broja omogućava prikaz vrlo velikih i vrlo malih binarnih brojeva, uvijek u istoj formi, bez korištenja velikog broja nula

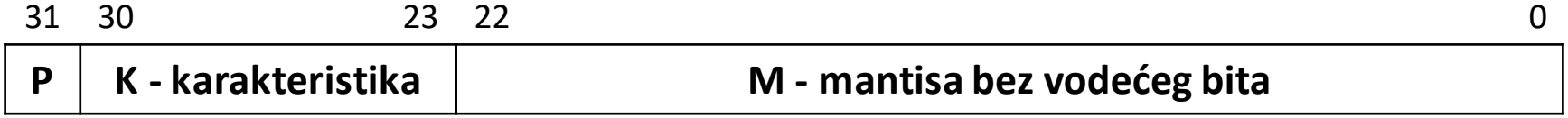
Prikaz realnih brojeva u računalu

- kako se podatak tipa float pohranjuje u registru računala?
- realni brojevi se u računalu pohranjuju u normaliziranom binarnom obliku
  - naravno, samo realni brojevi koji se mogu prikazati kao binarni razlomci s prihvatljivim brojem binarnih znamenaka
- Kako točno normalizirani binarni broj pohraniti u registar računala?
  - Standardom IEEE 754 propisan je način pohrane realnih brojeva
    - IEEE 754 standardna preciznost - *single precision*
    - IEEE 754 dvostruka preciznost - *double precision*

IEEE - Institute of Electrical and Electronics Engineers

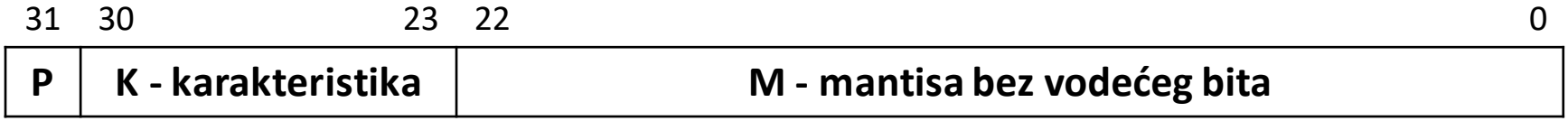
Realni brojevi standardne preciznosti

- IEEE 754 - *single precision*
  - najčešće se koristi za prikaz vrijednosti tipa float
  - binarni razlomak, u normaliziranom obliku, pohranjuje se u ukupno 32 bita (4 bajta) u sljedećem obliku:



  - 1 bit za pohranu predznaka broja
    - sam sadržaj bita određuje predznak broja
  - 8 bitova za pohranu karakteristike
    - pozitivni ili negativni binarni eksponent (BE) preslikan u uvijek pozitivnu karakteristiku (K)
  - 23 bita za pohranu mantise *bez vodećeg bita*
    - pohrana prvog bita (koji je uvijek jedinica) je nepotrebna

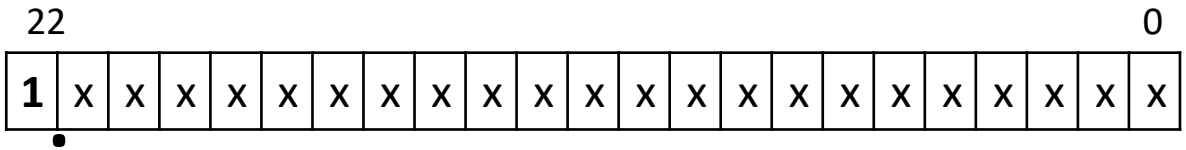
Realni brojevi standardne preciznosti



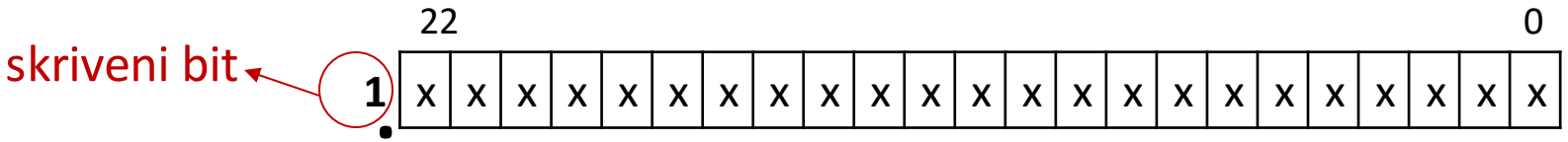
- predznak P: na mjestu najznačajnijeg bita upisuje se
  - 1 ako se radi o negativnom broju
  - 0 ako se radi o pozitivnom broju
- karakteristika K:
  - raspon binarnog eksponenta BE $\in$ [-126, 127]
    - radi izbjegavanja dvojnog komplementa za prikaz negativnih eksponenata, umjesto BE pohranjuje se karakteristika K
    - raspon karakteristike: K $\in$ [0, 255], K = BE + 127 $\Rightarrow$ BE $\in$ [-126, 127]
    - K = 0 i K = 255 se koriste za posebne slučajeve (objašnjeno kasnije)
- mantisa M:
  - ne pohranjuje se vodeći bit mantise - jedinica ispred binarne točke

Prvi bit mantise je implicitno određen

- ako se u 23 bita mora pohraniti *cijela* mantisa
  - iza binarne točke mogu se pohraniti najviše **22** binarne znamenke



  - uočiti: jedina znamenka koja se u normaliziranom broju može pojaviti ispred binarne točke je 1 (osim za prikaz broja nula). Jedan bit se troši nepotrebno jer je njegova vrijednost implicitno poznata
- stoga, vodeći bit (*leading bit*, *hidden bit*) se neće pohranjivati



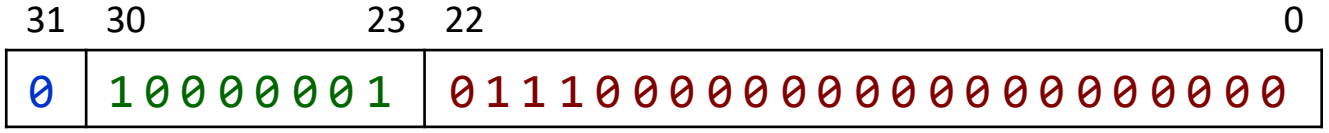
  - iza binarne točke mogu se pohraniti najviše **23** binarne znamenke
  - povećava se preciznost prikaza broja

Primjer

- Odrediti sadržaj registra u kojeg je pohranjen broj 5.75 prema standardu IEEE 754, standardnoj preciznosti. Sadržaj registra izraziti u binarnom i heksadekadskom brojevnom sustavu
  1. Odrediti bit za predznak: broj je pozitivan, stoga je P = 0
  2. Na temelju decimalnog razlomka izračunati binarni razlomak i normalizirati broj

     $5.75_{10} = 101.11_2 = 1.0111_2 \cdot 2^2$
  3. Izračunati karakteristiku i izraziti je u binarnom obliku, u 8 bitova

     $K = 2_{10} + 127_{10} = 129_{10} = 1000\ 0001_2$
  4. U registar prepisati bit za predznak, karakteristiku i mantisu <u>bez vodećeg bita</u>



$40\ B8\ 00\ 00_{16}$

Primjer

- 2.0

  $P = 0$

  $10_2 \cdot 2^0 = 1.0_2 \cdot 2^1$

  $BE = 1,\ K = 1 + 127 = 128_{10} = 10000000_2$

  $M = 1.000\ 0000\ \ldots\ 0000_2$

  $0100\ 0000\ 0000\ 0000\ \ldots\ 0000_2 = 4000\ 0000_{16}$

- - 2.0

  $P = 1$

  sve ostalo je jednako kao za 2.0

  $1100\ 0000\ 0000\ 0000\ \ldots\ 0000_2 = C000\ 0000_{16}$

Primjer

- 4.0

  $P = 0$

  $100_2 \cdot 2^0 = 1.0_2 \cdot 2^2$

  $BE = 2,\ K = 2 + 127 = 129_{10} = 10000001_2$

  $M = 1.000\ 0000\ \ldots\ 0000_2$

  $0100\ 0000\ 1000\ 0000\ \ldots\ 0000_2 = 4080\ 0000_{16}$

- 6.0

  $P = 0$

  $110_2 \cdot 2^0 = 1.10_2 \cdot 2^2$

  $BE = 2,\ K = 2 + 127 = 129_{10} = 10000001_2$

  $M = 1.100\ 0000\ \ldots\ 0000_2$

  $0100\ 0000\ 1100\ 0000\ \ldots\ 0000_2 = 40C0\ 0000_{16}$

Primjer

- 1.0

  $P = 0$

  $1_2 \cdot 2^0 = 1.0_2 \cdot 2^0$

  $BE = 0,\ K = 0 + 127 = 127_{10} = 01111111_2$

  $M = 1.000\ 0000\ ...\ 0000_2$

  $0011\ 1111\ 1000\ 0000\ ...\ 0000_2 = 3F80\ 0000_{16}$

- 0.75

  $P = 0$

  $0.11_2 \cdot 2^0 = 1.1_2 \cdot 2^{-1}$

  $BE = -1,\ K = -1 + 127 = 126_{10} = 01111110_2$

  $M = 1.100\ 0000\ ...\ 0000_2$

  $0011\ 1111\ 0100\ 0000\ ...\ 0000_2 = 3F40\ 0000_{16}$

Kalkulator za vježbanje

- Internet stranica na kojoj se nalazi dobar kalkulator za uvježbavanje zadataka s prikazivanjem realnih brojeva

  http://babbage.cs.qc.cuny.edu/IEEE-754/

Posebni slučajevi: prikaz realnog broja 0

- vodeća znamenka (bit) normaliziranog broja u binarnom obliku je uvijek 1, osim u slučaju realnog broja 0. Zato se u IEEE 754 standardu koristi posebno pravilo:
  - kada je K=0 i svi bitovi mantise su postavljeni na 0, radi se o prikazu realnog broja 0 (tada se ne smatra da je vodeći bit implicitno 1)
  - s obzirom da se pri tome bit za predznak može postaviti na 0 ili 1, moguće je prikazati "pozitivnu" i "negativnu" nulu, tj. +0 i -0
    - $00\ 00\ 00\ 00\ 00_{16} \rightarrow +0$
    - $80\ 00\ 00\ 00\ 00_{16} \rightarrow -0$
  - u relacijskim izrazima se te dvije vrijednosti smatraju jednakim

```
float x = 0.f, y = -0.f;
printf("x=%f, y=%f\n", x, y);
if (x == y) printf("x je jednak y");
```

```
x=0.000000, y=-0.000000↵
x je jednak y
```

Posebni slučajevi: denormalizirani broj

- kada je K = 0 i postoje bitovi mantise koji nisu 0, radi se o "denormaliziranom" broju
  - vodeći bit implicitno ima vrijednost 0
  - vrijednost binarnog eksponenta (BE) fiksirana je na -126 (ne koristi se izraz BE = K - 127)
  - omogućuje prikaz još manjih brojeva, ali uz smanjenu preciznost
    - stoga, izbjegavati – koristiti precizniji tip

0000 0000 0110 0000 0000 0000 0000 0000

$\rightarrow 0.11 \cdot 2^{-126}$

0000 0000 0000 0000 0000 0000 0000 0110

$\rightarrow 0.000\ 0000\ 0000\ 0000\ 0000\ 0110 \cdot 2^{-126}$

$\rightarrow 0.11 \cdot 2^{-146}$

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

- kada je K = 255 i svi bitovi mantise su postavljeni na 0, radi se o prikazu $+\infty$ ili $-\infty$, ovisno o bitu za predznak

  `0111 1111 1000 0000 0000 0000 0000 0000` $\rightarrow +\infty$

  `1111 1111 1000 0000 0000 0000 0000 0000` $\rightarrow -\infty$

  - vrijednosti se mogu dobiti npr.
    - dijeljenjem s nulom u realnoj domeni
    - prekoračenjem najvećeg broja koji se može prikazati

```
float x = 1.f / 0.f;
float y = -1.f / 0.f;
float z = 3.e38f * 2.f;

printf("%f %f %f", x, y, z);
```

```
inf -inf inf
```

Posebni slučajevi: prikaz "ne-broja" (NaN)

- Ako je K=255 i postoje bitovi mantise koji nisu 0, radi se o nedefiniranoj vrijednosti ili vrijednosti koju nije moguće prikazati (*NaN - Not a Number*), tj. ne radi se o legalnom prikazu broja
  - bit za predznak nema značenje
  - *NaN* je posljedica obavljanja operacije čiji je rezultat nedefiniran ili se prilikom obavljanja operacije dogodila pogreška, npr.

```
float x = 1.f / 0.f + -1.f / 0.f;
float y = 0.f / 0.f;
float z = -1.f / 0.f * 0.f;
float w = x * 0.f;
printf("%f %f %f %f\n", x, y, z, w);
```

```
nan nan nan nan
```

x → 1111 1111 1100 0000 0000 0000 0000 0000