

Programiranje (C)

14. predavanje

Saša Singer

`singer@math.hr`

`web.math.hr/~singer`

`www.math.hr/~singer`

PMF – Matematički odjel, Zagreb

Sadržaj predavanja

- Operacije nad bitovima:
 - Bitovni operatori i prioriteti.
 - Polja bitova.
- Pretprosesor:
 - Naredba `#include`.
 - Naredba `#define`.
 - Parametrizirana `#define` naredba.
 - Uvjetno uključivanje.
- Pregled standardne C biblioteke.
 - Matematičke funkcije `<math.h>`.

Operacije nad bitovima

Operatori nad bitovima

Operatori nad bitovima mogu se primijeniti na cjelobrojne tipove podataka **char**, **short**, **int** i **long** i djeluju na **bitove** unutar varijable:

Operator	Značenje
&	logičko I bit po bit
	logičko ILI bit po bit
^	ekskluzivno logičko ILI bit po bit
<<	lijevi pomak
>>	desni pomak
~	1-komplement

Svi operatori osim zadnjeg su **binarni**, a **~** je **unarni**.

Logički operatori nad bitovima

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

Definicije operacija dane su u sljedećoj tablici:

b1	b2	b1 $\&$ b2	b1 \wedge b2	b1 \mid b2
1	1	1	0	1
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Prioritet je redom: $\&$, \wedge pa \mid , ispod relacijskih jednakosti, iznad logičkog $\&\&$. Asocijativnost je $L \rightarrow D$.

Oprez! C nema binarnih konstanti!

Logički operatori nad bitovima (nastavak)

Primjer:

Logičko I:

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

Logičko ILI:

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

Logički operatori nad bitovima i 1-komplement

Ekskluzivno logičko ILI:

$$\begin{array}{rcl} a & = & 0x0003 \ /* = 0000\ 0000\ 0000\ 0011 */ \\ b & = & 0x0009 \ /* = 0000\ 0000\ 0000\ 1001 */ \\ \hline a \ ^ \ b & = & 0x000a \ /* = 0000\ 0000\ 0000\ 1010 */ \end{array}$$

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

Primjer:

1-komplement:

$$\begin{array}{rcl} a & = & 0x0c03 \ /* = 0000\ 1100\ 0000\ 0011 */ \\ \sim a & = & 0xf3fc \ /* = 1111\ 0011\ 1111\ 1100 */ \end{array}$$

Operatori pomaka

Operatori pomaka `<< i >>` pomiču binarni zapis broja nalijevo ili nadesno.

- Operatori pomaka **nalijevo** `<< i nadesno >>` uzimaju dva operanda:
 - prvi operand mora biti **cjelobrojni tip** nad kojim se operacija vrši,
 - a drugi operand je **broj bitova** za koji treba izvršiti pomak (tipa `unsigned int`).
- Drugi operand **ne smije** premašiti broj bitova u prvom operandu.

Prioritet operatora `<< i >>` je isti, **ispod** aritmetičkih aditivnih, **iznad** relacijskih. Asocijativnost je $L \rightarrow D$.

Operatori pomaka (nastavak)

- `<<` pomicе bitove **ulijevo** i to **ne ciklički**, tj. najznačajniji se bitovi **gube**, a zdesna se dodaju **nule**.
- `>>` pomicе bitove **udesno** i to **ne ciklički**, tj. najmanje značajni bitovi se **gube**.
 - Ako se pomak vrši na varijabli tipa **unsigned**, slijeva se dodaju **nule**.
 - Ako je varijabla cjelobrojnog tipa **s predznakom**, rezultat **ovisi** o implementaciji. Većina prevoditelja na lijevoj strani uvodi **bit predznaka**, dok drugi popunjavaju prazna mesta **nulama**.

Operatori pomaka (nastavak)

Primjer. $b = a \ll 6$ radi sljedeće:

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

Sve se bitovi pomiču 6 mesta ulijevo.

Primjer. $b = a \gg 6$ radi sljedeće:

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

Ako je **a** cjelobrojnog tipa **s predznakom** rezultat **ovisi** o implementaciji — na lijevoj strani uvodi se bit predznaka ili nule.

Jednostavni primjer

Primjer. Ako je u programu definirano:

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

onda bitovni operatori daju sljedeći rezultat:

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

Operatori pridruživanja

Logički **binarni** operatori uvedeni u ovom poglavlju formiraju složene **operatore pridruživanja**:

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

Maskiranje

Logički operatori najčešće služe **maskiranju** pojedinih bitova u operandu.

- Logičko I kao služi postavljanju određenih bitova **na 0**.
- Logičko ILI služi postavljanju određenih bitova **na 1**.
- Ekskluzivno ILI možemo koristiti za postavljanje određenih bitova **na 1** ako su bili **0** i obratno.

Primjer. Postavimo deseti najmanje značajan bit na nulu.

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

Maskiranje (nastavak)

Varijablu **mask** je najlakše napraviti tako da se:

- prvo postavi na **1** (najmanje značajni bit je **1**),
 - izvrši pomak **9** mesta **ulijevo**, čime je dobivena jedinica na **10.** mjestu, a sve ostalo su nule,
 - varijabla **mask** se 1-komplementira.
-

```
int a = 25;  
unsigned mask;  
  
mask = ~(1 << 9);  
a = a & mask; /* a &= mask */
```

Maskiranje (nastavak)

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

Prvo definiramo varijablu **mask** koja ima šest najmanje značajnih bitova jednakih **0**, a ostale **1**. Logički ILI s **mask** izdvaja najmanje značajne bitove u **a** i postavlja vodeće bitove u **b** na **1**.

```
mask = 0xffc0 /* = 1111 1111 1100 0000 */
b = a | mask;
```

Ova operacija **ovisi** o duljini tipa za **int**, no to se može izbjeći korištenjem 1-komplementa.

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

Složeniji primjer

Primjer. Program koji ispisuje binarni zapis cijelog broja tipa **int**.

```
#include <stdio.h>
int main(void) {
    int a, b, i, nbits;
    unsigned mask;

    nbits = 8 * sizeof(int);
    /* duljina tipa int */
    mask = 0x1 << (nbits - 1);
    /* 1 na najznačajnijem mjestu */
    printf("\nUnesite cijeli broj: ");
    scanf("%d", &a);
```

Složeniji primjer (nastavak)

```
for (i = 1; i <= nbits; ++i) {
    b = (a & mask) ? 1 : 0;
    printf("%d", b);
    if (i % 4 == 0) printf(" ");
    mask >>= 1;
}
printf("\n");
return 0;
}
```

Tablica prioriteta operatora (potpuna)

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

Tablica prioriteta operatora (potpuna)

Kategorija	Operatori	Asoc.
logičko I	<code>&&</code>	$L \rightarrow D$
logičko ILI	<code> </code>	$L \rightarrow D$
uvjetni	<code>? :</code>	$D \rightarrow L$
pridruživanje	<code>= += -= *= /= %=</code>	$D \rightarrow L$
pridruživanje (nast.)	<code>&= ^= = <<= >>=</code>	$D \rightarrow L$
op. zarez	<code>,</code>	$L \rightarrow D$

Polja bitova

Polja bitova omogućuju rad s pojedinim bitovima unutar jedne riječi u računalu. Polje bitova je skup susjednih bitova u sklopu jedne memorijeske jedinice (rijeci).

Upotreba:

- spemanje 1-bitnih zastavica (engl. flag) u jednu riječ, na primjer, koriste se u aplikacijama kao što je to tablica simbola za kompjeler,
- komunikacija s vanjskim uređajima – treba postaviti ili očitati samo dijelove riječi.

Deklaracija polja bitova

Deklaracija polja bitova je slična deklaraciji strukture:

```
struct ime {  
    clan 1 : broj_bitova 1;  
    ...  
    clan n : broj_bitova n;  
};
```

Svaki **clan k** polja bitova predstavlja polje bitova unutar riječi u računalu duljine **broj_bitova k**.

Polja bitova

Primjer:

```
struct primjer {  
    unsigned a : 1;  
    unsigned b : 3;  
    unsigned c : 2;  
    unsigned d : 1;  
};  
struct primjer v;  
...  
if (v.a == 1) ...  
v.c = STATIC;
```

Polja bitova (nastavak)

- Prva deklaracija definira **strukturu** razbijenu u četiri polja bitova: **a**, **b**, **c** i **d**.
- Ta polja redom imaju duljinu **1, 3, 2** i **1** bit. Prema tome zauzimaju **7** bitova.
- Poredak tih bitova unutar jedne riječi u računalu **ovisi o implementaciji**.
- Pojedine članove polja bitova dohvaćamo na isti način kako se dohvaćaju strukture, dakle **v.a**, **v.b** itd.
- Ako broj bitova deklariran u polju bitova **nadmašuje** duljinu **jedne** riječi u računalu, za pamćenje polja bit će upotrebljeno **više** riječi.

Polja bitova (nastavak)

Primjer. Program koji upotrebljava polje bitova:

```
#include <stdio.h>
int main(void) {
    static struct{
        unsigned a : 5;
        unsigned b : 5;
        unsigned c : 5;
        unsigned d : 5;
    } v = {1, 2, 3, 4};
    printf("v.a=%d, v.b=%d, v.c=%d, v.d=%d\n",
           v.a, v.b, v.c, v.d);
    printf("v treba %d okteta\n", sizeof(v));
    return 0; }
```

Polja bitova (nastavak)

Poredak polja unutar riječi može se kontrolirati korištenjem neimenovanih članova unutar polja kao u sljedećem primjeru. Primjer.

```
struct {  
    unsigned a : 5;  
    unsigned b : 5;  
    unsigned : 5;  
    unsigned c : 5;  
};  
struct primjer v;
```

Polja bitova (nastavak)

Primjer. Neimenovani član širine 0 bitova tjera prevoditelj da sljedeće polje smjesti u sljedeću računalnu riječ.

```
#include <stdio.h>

int main(void) {
    static struct{
        unsigned a : 5;
        unsigned b : 5;
        unsigned   : 0;
        unsigned c : 5;
    } v = {1, 2, 3};
```

Polja bitova (nastavak)

```
    printf("v.a=%d, v.b=%d, v.c=%d\n",
           v.a, v.b, v.c);
    printf("v treba %d okteta\n", sizeof(v));
    return 0;
}
```

Pretprocesor

Općenito o preprocessoru

- Prije prevodenja izvornog koda u objektni ili izvršni izvršavaju se **preprocessorske naredbe**.
- Svaka linija izvornog koda koja započinje znakom **#** predstavlja jednu preprocessorskiju naredbu.
- Preprocessorska naredba završava krajem linije, a **ne znakom ;**.

Opći oblik preprocessorskje naredbe:

#naredba parametri

i one **nisu sastavni dio jezika C te ne podliježu sintaksi jezika**.

Neke od preprocessorskih naredbi su: **#include, #define, #undef, #if, #ifdef, #ifndef, #elif, #else**.

Naredba #include

Naredba `#include` može se pojaviti u **dva oblika**:

```
#include "ime_datoteke"
```

ili

```
#include <ime_datoteke>
```

U oba slučaja pretprocesor briše liniju s `#include` naredbom i uključuje **sadržaj datoteke** `ime_datoteke` u izvorni kôd, na mjestu `#include` naredbe.

Naredba #include (*nastavak*)

- Ako je **ime_datoteke** navedeno **unutar navodnika**, onda preprocessor datoteku traži u direktoriju u kojem se nalazi **izvorni program**.
- Ako je ime datoteke navedeno između znakova **< >**, to signalizira da se radi o **sistemskoj datoteci** (kao npr. **stdio.h**), pa će preprocessor datoteku tražiti na mjestu određenom operacijskim sustavom.

Naredba #define

Deklaracija:

```
#define ime tekst_zamjene
```

Preprocessor će od mjesta na kome se **#define** naredba nalazi **do kraja datoteke** svako pojavljivanje imena **ime** zamijeniti s tekstrom **tekst_zamjene**. Do zamjene **neće doći** unutar znakovnih nizova, tj. unutar dvostrukih navodnika.

Parametrizirane makro naredbe

U parametriziranoj makro naredbi simboličko ime i tekst koji zamjenjuje simboličko ime sadrže **argumente** koji se prilikom poziva makro naredbe zamjenjuju stvarnim argumentima.

Sintaksa:

```
#define ime(argumenti) tekst_zamjene
```

- Argumenti makro naredbe pišu se u zagradama.
- Makro naredba je **efikasnija** od funkcije, jer u njoj nema prenošenja argumenata.

Parametrizirane makro naredbe (nastavak)

Primjer:

```
#define max(A, B) ((A) > (B) ? (A) : (B))
```

gdje su **A** i **B** argumenti.

Ako se u kodu pojavi naredba:

```
x = max(a1, a2);
```

preprocesor će je zamijeniti s:

```
x = ((a1) > (a2) ? (a1) : (a2));
```

Formalni argumenti (parametri) **A** i **B** zamijenjeni su stvarnim argumentima **a1** i **a2**.

Parametrizirane makro naredbe (nastavak)

Ako na drugom mjestu imamo naredbu:

```
x = max(a1 + a2, a1 - a2);
```

ona će biti zamijenjena s

```
x = ((a1 + a2) > (a1 - a2)
      ? (a1 + a2) : (a1 - a2));
```

Primjer parametrizirane makro naredbe

Primjer. Dio izvornog kôda:

```
#include <stdio.h>

#define SQ1(x) x*x
#define SQ2(x) (x)*(x)
#define SQ3(x) ((x)*(x))

int main(void) {
    printf("%d\n", SQ1(1+1));
    printf("%d\n", 4/SQ2(2));
    printf("%d\n", 4/SQ3(2));
    return 0;
}
```

Primjer parametrizirane makro naredbe (nast.)

ispisuje:

3
4
1

Objašnjenje:

- SQ1: `#define SQ1(x) x*x`
 $x = 1 + 1$ i doslovnom supstitucijom u `SQ1` dobivamo
 $1 + 1 * 1 + 1 = \text{(prioritet!)} = 1 + 1 + 1 = 3$
- SQ2: `#define SQ2(x) (x)*(x)`
 $x = 2$ i dobivamo
 $4 / (2) * (2) = 4 / 2 * 2 = 2 * 2 = 4$

Primjer parametrizirane makro naredbe (nast.)

- SQ3: `#define SQ3(x) ((x)*(x))`

`x = 2` i dobivamo

$$4 / ((2) * (2)) = 4 / (2 * 2) = 4 / 4 = 1$$

Primijetite da se u parametriziranim makro naredbama koristi **gomila** zagrada da bi se osigurao **korektan prioritet operacija!**

Tek zadnji **SQ3** korektno daje **kvadrat** argumenta u svim slučajevima (a to se htjelo!).

Razlika makro naredbe i funkcije

Sličnost makro naredbe i funkcije može zavarati. Ako bismo makro naredbu `max` pozvali na sljedeći način:

```
max(i++, j++);
```

varijable `i` i `j` ne bi bile inkrementirane samo jednom (kao pri funkcijском pozivu) već bi veća varijabla bila inkrementirana dva puta.

- Kod makro naredbe **nema kontrole** tipa argumenata.
- Neke su “funkcije” deklarirane u `<stdio.h>` zapravo makro naredbe, na primjer `getchar` i `putchar`. Isto tako, funkcije u `<ctype.h>` uglavnom su izvedene kao makro naredbe.

Naredba #define i više linija teksta

- U **#define** naredbi tekst zamjene je od imena koje definiramo **do kraja linije**.
- Ako želimo da ime bude zamijenjeno s **više linija teksta** moramo koristiti kosu crtlu (****) na kraju svakog reda osim posljednjeg.

Primjer: Makro za inicijalizaciju polja možemo definirati ovako:

```
#define INIT(polje, dim) for(int i=0; \
                           i < (dim); ++i) \
                           (polje)[i] = 0.0;
```

Naredba #undef

Definicija nekog imena može se poništiti korištenjem **#undef** naredbe.

Primjer:

```
#include <math.h>
    /* math.h definira M_PI kao 3.14... */
#undef M_PI
#define M_PI atan(1.0)
...
```

Uvjetno uključivanje

Korištenjem preprocesorskih naredbi **#if**, **#else**, **#elif** možemo **uvjetno uključivati** ili **isključivati** pojedine dijelove programa.

Naredba **#if** ima sljedeći oblik:

```
#if uvjet
    blok naredbi
#endif
```

Ako je **uvjet** ispunjen blok naredbi između **#if uvjet** i **#endif** bit će uključen u izvorni kôd, a ako **uvjet** nije ispunjen, blok neće biti uključen.

Uvjetno uključivanje (nastavak)

- Uvjet koji se pojavljuje u `#if` naredbi je **konstantan cjelobrojni izraz**. Nula se interpretira kao laž, a svaka vrijednost različita od nule kao istina.
- Najčešća svrha uključivanja/isključivanja je uključiti neku varijablu ako nije bila definirana.

Tu nam pomaže izraz

`defined(ime)`

koji daje **1** ako je **ime** definirano, a **0** ako nije.

Uvjetno uključivanje (nastavak)

Primjer:

```
#if !defined(__datoteka.h__)
#define __datoteka.h__

/* ovdje dolazi datoteka.h */

#endif
```

To je standardna tehnika kojom se izbjegava višestruko uključivanje **.h** datoteka.

Naredbe #ifdef i #ifndef

Konstrukcije **#if defined** i **#if !defined** se često pojavljuju, pa postoje pokrate: **#ifdef** i **#ifndef**.

Primjer: Prethodnu konstrukciju mogli smo napisati u obliku:

```
#ifndef __datoteka.h__  
#define __datoteka.h__  
  
/* ovdje dolazi datoteka.h */  
  
#endif
```

Zagrade oko varijabli nisu obavezne.

Naredbe `#else` i `#elif`

Složene `if` naredbe grade se pomoću: `#else` i `#elif`.

- Značenje `#else` je isto kao značenje `else` u C-u.
- Značenje `#elif` je isto kao značenje `else if`.

Primjer. Testira se koji je operativni sustav u pitanju, tj. ime `SYSTEM` da bi se uključilo ispravno zaglavlje.

Naredbe #else i #elif (*nastavak*)

```
#if SYSTEM == SYSV
    #define DATOTEKA "sysv.h"
#elif SYSTEM == BSD
    #define DATOTEKA "bsd.h"
#elif SYSTEM == MSDOS
    #define DATOTEKA "msdos.h"
#else
    #define DATOTEKA "default.h"
#endif
```

Naredbe #else i #elif (*nastavak*)

Primjer. U razvoju programa korisno je ispisivati međurezultate, kako bi se lakše kontrolirala korektnost izvršavanja programa. U završenoj verziji programa, sav suvišan ispis treba eliminirati.

```
...
scanf("%d", &x);
#ifndef DEBUG
    printf("Debug:: x=%d\n", x);
    /* testiranje */
#endif
```

Prevoditelji pod Unix-om obično **-Dsimbol** opciju koja dozvoljava da se **simbol** definira na komandnoj liniji.

Naredbe #else i #elif (*nastavak*)

Primjer. Pretpostavimo da je program koji sadrži prikazani dio kôda smješten u `prog.c`. Tada će prevođenje naredbom

```
cc -o prog prog.c
```

proizvesti program u koji ispis varijable `x` nije uključen.
Prevođenje naredbom

```
cc -DDEBUG -o prog prog.c
```

dat će izvršni kôd koji uključuje `printf` naredbu, jer je varijabla `DEBUG` definirana.

Naredba assert

Mnoge funkcije očekuju argumente koji zadovoljavaju određene uvjete. Recimo, funkcija može očekivati da u argument tipa **double** dobiva samo pozitivne vrijednosti.

Takvih provjera može biti puno, a namjera ih je isključiti u konačnoj verziji kôda. Za to se koristi makro naredba **assert**.

- Makro naredba **assert** definirana je u **<assert.h>**.
- Makro naredba **assert** koristi se kao funkcija:

```
void assert(int izraz)
```

Naredba assert (*nastavak*)

Ako je **izraz** jednak nuli u trenutku izvršavanja

```
assert(izraz);
```

assert će ispisati poruku:

```
Assertion failed: izraz, file ime_datoteke,  
line br_linije
```

Nakon toga **assert** zaustavlja izvršavanje programa.

Naredba assert (*nastavak*)

Primjer. Funkcija očekuje pozitivan argument.

```
#include <stdio.h>
#include <assert.h>
int f(int x) {
    assert(x > 0);
    return x; }
int main(void) {
    int x = -1;
    printf("x=%d\n", f(x));
    return 0; }
```

Poruka koju program ispisuje:

Assertion failed: x > 0, file C:\a1.cpp, line 6

Isključivanje assert naredbe

Želimo li isključiti **assert** naredbe iz programa dovoljno je prije uključivanja datoteke zaglavlja **<assert.h>** definirati **NDEBUG** ovako:

```
#include <stdio.h>
#define NDEBUG
#include <assert.h>
...
```

Standardna biblioteka

Općenito o zaglavljima

Funkcije, tipovi i makro naredbe standardne biblioteke deklarirani su u standardnim zaglavljima:

```
<assert.h>  <float.h>   <math.h>     <stdarg.h>  
<stdlib.h>   <ctype.h>    <limits.h>   <setjmp.h>  
<stddef.h>   <string.h>   <errno.h>    <locale.h>  
<signal.h>   <stdio.h>    <time.h>
```

Matematičke funkcije u <math.h>

Konvencija: `x` i `y` tipa `double`, a `n` tipa `int`. Sve funkcije kao rezultat vraćaju `double`.

Funkcija	Značenje
<code>sin(x)</code>	$\sin x$
<code>cos(x)</code>	$\cos x$
<code>tan(x)</code>	$\operatorname{tg} x$
<code>asin(x)</code>	$\arcsin x \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right], \quad x \in [-1, 1]$
<code>acos(x)</code>	$\arccos x \in [0, \pi], \quad x \in [-1, 1]$
<code>atan(x)</code>	$\operatorname{arctg} x \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$
<code>atan2(y, x)</code>	(x, y) koordinate točke u ravnini – vraća $\operatorname{arctg} \frac{y}{x} \in [-\pi, \pi]$ jer zna o kojem se kvadrantu radi (v. primjer).

Matematičke funkcije u $\langle \text{math.h} \rangle$ (nastavak)

Funkcija	Značenje
<code>sinh(x)</code>	$\text{sh } x$
<code>cosh(x)</code>	$\text{ch } x$
<code>tanh(x)</code>	$\text{th } x$
<code>exp(x)</code>	e^x
<code>log(x)</code>	$\ln x, \quad x > 0$
<code>log10(x)</code>	$\log_{10} x, \quad x > 0$
<code>pow(x,y)</code>	x^y – greška ako $x = 0$ i $y \leq 0$ ili $x < 0$ i y nije cijeli broj
<code>sqrt(x)</code>	$\sqrt{x}, \quad x \geq 0$

Matematičke funkcije u <math.h> (nastavak)

Funkcija	Značenje
<code>ceil(x)</code>	$\lceil x \rceil$, u double formatu najmanji cijeli broj $\geq x$
<code>floor(x)</code>	$\lfloor x \rfloor$, u double formatu najveći cijeli broj $\leq x$
<code>fabs(x)</code>	$ x $
<code>ldexp(x, n)</code>	$x \cdot 2^n$
<code>frexp(x, int *exp)</code>	ako $x = y \cdot 2^n$, $y \in [\frac{1}{2}, 1)$, vraća y , a potenciju n spremu u <code>*exp</code> . Ako $x = 0$, $y = n = 0$.

Matematičke funkcije u <math.h> (nastavak)

Funkcija	Značenje
<code>modf(x, double *ip)</code>	rastavlja x na cijelobrojni i razlomljeni dio, oba istog predznaka kao x . Razlomljeni dio vrati, a cijelobrojni dio spremi u <code>*ip</code> .
<code>fmod(x, y)</code>	realni (floating-point) ostatak dijeljenja x/y istog znaka kao x . Ako $y = 0$ rezultat ovisi o implementaciji.

atan *vs.* atan2

Primjer: Program

```
#include <stdio.h>
#include <math.h>
main(void){
    double x, y;
    x = y = -1.0;
    printf("%f\n", atan(y/x));
    printf("%f\n", atan2(y, x));
    return 0; }
```

ispisuje

0.785398

-2.356194
