

Programiranje 2

13. predavanje

Saša Singer

`singer@math.hr`

`web.math.hr/~singer`

PMF – Matematički odjel, Zagreb

Sadržaj predavanja

- Polja bitova.
- Pretprocesor:
 - Naredba `#include`.
 - Naredba `#define`.
 - Parametrizirana `#define` naredba.
 - Uvjetno uključivanje.
- Pregled standardne C biblioteke.
 - Standardne datoteke zaglavlja `*.h`.
 - Matematičke funkcije `<math.h>`.

Informacije

Termin drugog kolokvija je:

- četvrtak, 3. 7., u 12 sati.

Kako sad stvari stoje, popravni kolokvij je u “jesen”:

- srijeda, 10. 9., u 12 sati.

Na službenom webu, pod materijalima, od nedavno postoji i

- službeni podsjetnik = “šalabahter”,
- tj., kratki opis svih bitnih elemenata C-a.

Ali, nemojte ga nositi na kolokvij — to je zabranjeno!

- Dobit ćete ga na kolokviju — to se “dijeli” kao poklon.

Razlog: Nećemo “obrađene” šalce — s “osobnim dodacima”.

Polja bitova

Polja bitova

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

Upotreba:

- spremanje 1-bitnih **zastavica** (engl. flag) u jednu riječ, na primjer, koriste se u aplikacijama kao što je to tablica simbola za kompajler,
- 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 upotrebjeno **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 pretprocesoru

- Prije prevođenja izvornog koda u objektni ili izvršni izvršavaju se **pretprocesorske naredbe**.
- Svaka linija izvornog koda koja započinje znakom **#** predstavlja jednu pretprocesorsku naredbu.
- Pretprocesorska naredba završava krajem linije, a **ne znakom ;**.

Opći oblik pretprocesorske naredbe:

```
#naredba parametri
```

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

Neke od preprocesorskih 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 pretprocesor datoteku traži u direktoriju u kojem se nalazi **izvorni program**.
- Ako je ime datoteke navedeno između znakova `< >`, to signalizira da se radi o **sistemskej datoteci** (kao npr. `stdio.h`), pa će pretprocesor datoteku tražiti na mjestu određenom operacijskim sustavom.

Naredba #define

Deklaracija:

```
#define ime tekst_zamjene
```

Pretprocesor će od mjesta na kome se `#define` naredba nalazi **do kraja datoteke** svako pojavljivanje imena `ime` zamijeniti s tekстом `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 = (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 funkcijskom 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 crtu (`\`) na kraju svakog reda osim posljednjeg.

Primjer. Makro za inicijalizaciju polja možemo definirati ovako:

```
#define INIT(polje, dim) for(int i=0;\n        i < (dim); ++i) \n        (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 (4.0*atan(1.0))
...
```

Uvjetno uključivanje

Korištenjem pretprocesorskih 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 **datoteku zaglavlja**, ako neka varijabla **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__)\n#define __datoteka.h__\n\n/* ovdje dolazi datoteka.h */\n\n#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);
#ifdef DEBUG
    printf("Debug:: x=%d\n", x);
    /* testiranje */
#endif
```

Prevoditelji pod Unix-om obično imaju `-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 ispisiuje:

```
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 prepoznaje kvadrant

Matematičke funkcije u `<math.h>` (nastavak)

Funkcija	Značenje
<code>sinh(x)</code>	$\operatorname{sh} x$
<code>cosh(x)</code>	$\operatorname{ch} x$
<code>tanh(x)</code>	$\operatorname{th} x$
<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 sprema u <code>*exp</code> . Ako $x = 0$, $y = n = 0$.

Matematičke funkcije u `<math.h>` (nastavak)

Funkcija

Značenje

`modf(x, double *ip)`

rastavlja x na cjelobrojni i razlomljeni dio, oba istog predznaka kao x . Razlomljeni dio vrati, a cjelobrojni dio spremi u `*ip`.

`fmod(x, y)`

realni (floating-point) ostatak dijeljenja x/y istog znaka kao x . Ako $y = 0$ rezultat ovisi o implementaciji.

Razlika atan i atan2

Razlika između funkcija

```
double atan(double x);  
double atan2(double y, double x);
```

atan(x) vraća:

• $\arctg x \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right],$

atan2(y, x) interpretira argumente kao **koordinate** točke (x, y) u ravnini i vraća:

• $\arctg \frac{y}{x} \in [-\pi, \pi],$ jer prepoznaje **kvadrant** u kojem je točka.

Razlika atan i atan2 (nastavak)

Primjer. Program

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

ispisuje:

```
0.785398
-2.356194
```

Točni rezultati su $\pi/4$, $-3\pi/4$.

Funkcije floor i ceil

Uočite da funkcije za “najmanje” i “najveće” cijelo:

```
double floor(double x);  
double ceil(double x);
```

vraćaju rezultat tipa `double`, a **ne** `int`.

Primjer. Rezultat ispisa sljedećeg dijela kôda

```
printf("%g\n", floor(5.2));  
printf("%g\n", floor(-5.2));  
printf("%g\n", ceil(5.2));  
printf("%g\n", ceil(-5.2));
```

je: `5`, `-6`, `6`, `-5` (zbog `%g` formata).

Funkcija fmod

Funkcija

```
double fmod(double x, double y);
```

vraća

- “realni” **ostatak** pri dijeljenju x/y ,
- s tim da ostatak ima **isti** predznak kao x .

Princip je isti kao kod cjelobrojnog dijeljenja.

Funkcija fmod (nastavak)

Primjer. Sljedeći odsječak kôda

```
printf("%g\n", fmod(5.2, 2.6));  
printf("%g\n", fmod(5.57, 2.51));  
printf("%g\n", fmod(5.57, -2.51));  
printf("%g\n", fmod(-5.57, -2.51));  
printf("%g\n", fmod(-5.57, -2.51));
```

ispisuje:

0, 0.55, 0.55, -0.55, -0.55

(zbog $5.57 = 2 \cdot 2.51 + 0.55$).

Funkcija frexp

Funkcija

```
double frexp(double x, int *exp);
```

rastavlja broj x na binarnu “mantisu” i binarni eksponent.

Primjer. Sljedeći odsječak kôda

```
double x = 8.0;
int exp;
printf("%f\n", frexp(x, &exp));
printf("%d\n", exp);
```

ispisuje

```
0.5, 4
```

Funkcije exp, log, log10 i pow

```
double exp(double x);  
double log(double x);  
double log10(double x);  
double pow(double x, double y);
```

Primjer.

```
printf("%g\n", log(exp(22)));  
printf("%g\n", log10(pow(10.0, 22.0)));
```

Neke funkcije iz <stdlib>

Datoteka zaglavlja <stdlib.h> ima nekoliko vrlo korisnih funkcija. Na primjer:

- `qsort` — QuickSort algoritam za općenito sortiranje niza podataka,
- `bsearch` — Binarno traženje zadanog podatka u (sortiranom) nizu.

U ovim funkcijama možemo sami zadati

- funkciju za uspoređivanje podataka u nizu.

Funkcija qsort

Funkcija za **sortiranje** niza **QuickSort** algoritmom:

```
void qsort(void *base, size_t n, size_t size,  
           int (*comp) (const void *, const void *));
```

Primjer.

```
int main(){  
    char rjecnik[3][20] = {"po", "ut", "sri"};  
  
    qsort(rjecnik, 3, 20, strcmp);  
    puts(rjecnik[2]);  
    return 0;  
}
```

Funkcija bsearch

Funkcija za **binarno traženje** zadanog podatka u sortiranom nizu:

```
void *bsearch(const void *key, const void *base,
             size_t n, size_t size,
             int (*comp) (const void *, const void *));
```

Vraća **pokazivač** na **nađeni** podatak (ako ga ima), ili **NULL**.

Primjer.

```
printf("%s\n",
       bsearch("ut", rjecnik, 3, 20, strcmp));
```

Definiranje kriterija sortiranja

Primjer.

```
int main()
{
    int i, polje[4] = {1, 3, -4, 3};

    qsort(polje, 4, sizeof(int), usporedi);

    for (i = 0; i < 4; ++i)
        printf("%d\n", polje[i]);

    return 0;
}
```

Funkcija usporedi

Funkcija za uspoređivanje cijelih brojeva:

```
int usporedi(const int *a, const int *b)
{
    if (*a == *b)
        return (0);
    if (*a > *b)
        return (1);
    else
        return (-1);
}
```

Funkcije rand i srand

Funkcije za generiranje “slučajnih” cijelih brojeva:

```
int rand(void);  
void srand(unsigned int seed);
```

Funkcija `rand()` vraća

- tzv. “pseudo-slučajni” cijeli broj u rasponu od 0 do `RAND_MAX`, s tim da `RAND_MAX` mora biti barem $2^{15} - 1 = 32767$ (tj. 16-bitni `int`).

Funkcija `srand(seed)`

- postavlja tzv. “sjeme” za generator “pseudo-slučajnih” brojeva na zadanu vrijednost `seed`.

Standardno “sjeme” je 1, ako ga ne postavimo sami!

Funkcije rand i srand (nastavak)

Primjer.

```
int seed;

scanf("%d", &seed);
srand(seed);

printf("%d\n", rand());
printf("%d\n", rand());
printf("%d\n", RAND_MAX);    /* 32767 */
```
