Programiranje 1
6. predavanje

Saša Singer

PMF – Matematički odsjek, Zagreb

Sadržaj predavanja

- Osnovni elementi jezika C:
  - Skup znakova.
  - Identifikatori.
  - Ključne riječi.
  - Osnovni tipovi podataka.
- Konstante i varijable:
  - Konstante.
  - Varijable i deklaracija.
  - Inicijalizacija varijabli.
  - Enumeracije — pobrojani tipovi.

Sadržaj predavanja (nastavak)

- Operatori i izrazi (prvi dio):
  - Izraz, operandi i operatori.
  - Operator pridruživanja.
  - Aritmetički operatori.
  - Pretvaranje tipova u aritmetičkim izrazima.
  - Eksplicitno pretvaranje tipova — cast operator.
  - Redoslijed računanja izraza.
  - Prioritet i asocijativnost operatora.
- Operatori i izrazi (drugi dio) — sljedeće predavanje.

Osnovni elementi jezika C

Sadržaj

- Osnovni elementi jezika C:
  - Skup znakova.
  - Identifikatori.
  - Ključne riječi.
  - Osnovni tipovi podataka.

Skup znakova

Programski jezik C koristi sljedeći skup znakova:

- velika i mala slova engleske abecede `A-Z` i `a-z`,
- numeričke znakove — dekadske znamenke `0-9`,
- specijalne znakove:

```
+  -  *  /  =  %  &  #
!  ?  ^  "  '  ~  \  |
<  >  (  )  [  ]  {  }
:  ;  .  ,  _  (bjelina)
```

Pod bjelinom se podrazumijeva, osim same bjeline (blanka), horizontalni i vertikalni tabulator, te znakovi za prijelaz u novi red, na novu stranicu i vraćanje na početak reda.

Razmak ili bjelina = separator

Gramatički gledano, razmak ili "praznina" (engl. blank) služi

- odvajanju pojedinih riječi ili drugih cjelina u jeziku, tj. kao separator.

Potpuno isto vrijedi i na kraju reda teksta programa.

- Znak za kraj reda je "bjelina" — pa i separator.

Takvih separatora smije biti i više, a prevoditelj preskače (ignorira) "višak separatora".

Naravno, ovo vrijedi izvan stringova i sl.

- Tamo se svaki znak, pa i praznina, interpretira naprosto kao podatak.

Komentari

Program treba komentirati radi lakšeg razumijevanja njegovog funkcioniranja.

- Prevoditelj ignorira (preskače) komentare pri prevođenju!

Pravila za pisanje komentara:

- Komentar započinje parom znakova `/*`
- i završava prvim sljedećim parom `*/`.

Komentar može sadržavati više linija teksta.

Primjer.

```
/*        Ovo je
             komentar.   */
```

Komentari (nastavak)

Tipične greške u pisanju komentara:

- Ako je ispušten jedan graničnik (zadnji), može se dogoditi gubitak kôda.

```
/*    Ovo je prvi komentar.       Nezatvoren!!!
x = 72.0;
/*    Ovo je drugi komentar. */
```

Prevoditelj ignorira tekst do prvog para znakova `*/` .

Posljedica. Dodjeljivanje

```
x = 72.0;
```

je dio komentara.

Komentari (nastavak)

Nije dozvoljeno pisati komentar unutar komentara — greška.

Primjer.

```
/*
x = 72.0;    /*   Inicijalizacija  */
y = 31.0;
*/
```

Prvi komentar završava prvim sljedećim parom znakova `*/`, tj. na kraju druge linije.

Drugi komentar nema početak `/*` (onaj kojem bi kraj bio u četvrtoj liniji).

Komentari (nastavak)

Noviji standard C99 dovoljava još jedan "skraćeni" oblik komentara — tzv. C++ tip komentara.

- Takav komentar počinje parom znakova `//`
- i završava krajem linije.

Oprez: ovaj oblik komentara nije dozvoljen u starijim prevoditeljima.

Primjer.

```
x = 2.17;   // Inicijalizacija
```

Identifikatori

Identifikatori su imena koja pridružujemo različitim elementima programa — na primjer,

- varijablama, poljima i funkcijama.

Pravila za pisanje identifikatora:

- Sastoje se od slova i znamenki (alfanumeričkih znakova), s tim da prvi znak mora biti slovo.
- Velika i mala slova se razlikuju.
- Znak `_` (donja crta) smatra se slovom.

Duljina identifikatora je proizvoljna (katkad se ograničava na 255 znakova). Međutim, prevoditelj

- nije dužan razlikovati identifikatore koji su isti na prvih 6–63 znakova (ovisno o standardu i vrsti varijable).

Ključne riječi

Ključne riječi imaju posebno značenje u jeziku i

- ne smiju se koristiti kao identifikatori.

Programski jezik C, standardno, ima 32 ključne riječi:

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

Proširenja jezika C mogu imati i više ključnih riječi!

Primjeri identifikatora

Primjer. Ispravno napisani identifikatori:

```
x,       y13,    sum_1,    _temp,
names,   Pov1,   table,    TABLE
```

Primjer. Neispravno napisani identifikatori:

```
3dan,     /* prvi znak je broj */
"x",      /* nedozvoljeni znak " */
ac-dc     /* nedozvoljeni znak - */
print f   /* nedozvoljeni znak praznina,
             pa su to dvije rijeci */
extern    /* kljucna rijec */
```

Osnovni tipovi podataka

Imena za osnovne tipove podataka u C-u su ključne riječi.

- `int`: cjelobrojni podatak. Tipično zauzima 4 bajta.
- `char`: znakovni podatak. Sadržava jedan znak. Tipično zauzima 1 bajt.
- `float`: realni broj s pomičnom točkom (floating–point) u jednostrukoj preciznosti. Tipično zauzima 4 bajta. (IEEE single, ili binary32).
- `double`: realni broj s pomičnom točkom (floating–point) u dvostrukoj preciznosti. Tipično zauzima 8 bajtova. (IEEE double, ili binary64).
- pokazivač: podatak je adresa nekog drugog podatka u memoriji. Tipično zauzima 4 bajta ili 8 bajtova (Oprez!).

Kratki i dugi tip int

Cjelobrojni tip `int` može se modificirati

- pomoću kvalifikatora `short` i `long`.

Tako dobivamo nove cjelobrojne tipove.

Cjelobrojni tipovi za brojeve s predznakom:

- `short int`, ili, kraće, `short`: "kratki" cjelobrojni podatak. U memoriji zauzima manje ($\leq$) mjesta od `int`, pa mu je manji raspon prikazivih cijelih brojeva.
- `long int`, ili, kraće, `long`: "dugi" cjelobrojni podatak. U memoriji zauzima više ($\geq$) mjesta od `int`, pa mu je veći raspon prikazivih cijelih brojeva.

Neki prevoditelji dozvoljavaju i `long long`, tj. "vrlo dugi" cjelobrojni podatak.

Tipovi za brojeve bez predznaka

Cjelobrojne tipove za brojeve bez predznaka dobivamo

- upotrebom kvalifikatora `unsigned`.

Oni zauzimaju isti memorijski prostor kao osnovni tipovi podataka (`int`, `short`, `long`), a mogu reprezentirati samo nenegativne cijele brojeve.

- Pokrivaju približno dvostruko veći raspon pozitivnih cijelih brojeva od osnovnih tipova. (Ponoviti prikaz!)

Cjelobrojni tipovi za brojeve bez predznaka:

- `unsigned int`, ili, kraće, `unsigned`,
- `unsigned short int`, ili, kraće, `unsigned short`,
- `unsigned long int`, ili, kraće, `unsigned long`.

Još o cjelobrojnim tipovima

C propisuje samo minimalnu preciznost (tj. duljinu) pojedinih cjelobrojnih tipova:

- tip `int` mora imati najmanje 16 bitova, a
- tip `long` mora imati najmanje 32 bita.

Operator `sizeof` (piše se kao funkcija) daje broj bajtova rezerviranih za prikaz vrijednosti odgovarajućeg tipa. Vrijedi:

```
sizeof(short) <= sizeof(int) <= sizeof(long)
```

Datoteka zaglavlja `<limits.h>` sadrži simboličke konstante za minimalne i maksimalne dozvoljene vrijednosti pojedinih cjelobrojnih tipova.

Tipovi char i signed char

Osnovni tip `char` služi primarno za prikaz znakova. Znakovi se prikazuju svojim kôdom,

- kao "vrlo kratki" cijeli brojevi bez predznaka.

Standardno, ovaj tip zauzima 1 bajt, a prikazivi brojevi (odnosno, kôdovi) imaju raspon od 0 do 255.

Tip `char` može se modificirati

- pomoću kvalifikatora `signed`,

tako da tip `signed char` sadrži

- "vrlo kratke" cijele brojeve s predznakom.

Standardno, i ovaj tip zauzima 1 bajt, a prikazivi brojevi imaju raspon od -128 do 127.

Svrha tipa signed char

Tip `signed char` ima koristi samo kad treba

- "gusto" pakirati podatke u složenijim strukturama,

recimo, pri komunikaciji sa specijalnim vanjskim uređajima.

Za stvarno računanje — nema puno smisla,

- osim za "štednju" memorije,

zbog automatskog pretvaranja tipova (v. sljedeći put).

Realni tipovi

Kvalifikator `long` se može primijeniti i na realne tipove podataka.

- `long float` je isto što i `double`, a
- `long double` ima četverostruku preciznost (ako postoji).

Datoteka zaglavlja `<float.h>` sadrži simboličke konstante koje daju različite informacije o realnim tipovima podataka.

Osnovni tipovi podataka — sažetak

Osnovni tipovi podataka u jeziku C su:

- `char` — znakovni tip, `sizeof(char) = 1` (bajt),
- `int` — cjelobrojni tip, `sizeof(int) = 4`,
- `float` — realni tip, jednostruka preciznost, `sizeof(float) = 4`,
- `double` — realni tip, dvostruka preciznost, `sizeof(double) = 8`.

Kvalifikatori:

- `unsigned` — brojevi bez predznaka,
- `short` — "skraćuje" duljinu tipa,
- `long` — "produljuje" duljinu tipa.

Konstante i varijable

Sadržaj

- Konstante i varijable:
  - Konstante.
  - Varijable i deklaracija.
  - Inicijalizacija varijabli.
  - Enumeracije — pobrojani tipovi.

Operandi u izrazima

Osnovni operandi u jeziku C su:

- konstante,
- varijable.

Poslije dođu još i vrijednosti funkcija.

Operandi imaju sljedeće atribute ("osobni opis"):

- ime,
- vrijednost,
- tip.

Za varijablu — ime je identifikator.

Konstante nemaju posebno ime (tj. vrijednost konstante je, ujedno, i njezino ime).

Cjelobrojne konstante

Cjelobrojne konstante mogu biti zapisane u tri brojevna sustava:

- decimalnom (baza 10),
- oktalnom (baza 8)
- i heksadecimalnom (baza 16).

Baza zapisa prepoznaje se

- po početnim znakovima zapisa konstante.

Napomena. Binarni zapis konstanti nije predviđen (čisto zbog duljine zapisa — oktalni i heksadecimalni zapisi su kraći).

Cjelobrojne konstante (nastavak)

Decimalne (dekadske) konstante:

- sadrže znamenke `0-9`, dozvoljen predznak `-` ili `+`,
- ako konstanta ima više od jedne znamenke (bar dvije), prva znamenka nije `0` (služi kao oznaka za ostale baze).

Primjer.

```
0,  1,  234,  -456,  99999
```

Napomena. Decimalna točka nije dozvoljena, jer dobijemo realni broj. Dakle, `17` nije isto što i `17.` (s točkom).

- Prva konstanta je tipa `int`, a druga je tipa `double`.

One jesu iste kad se uspoređuju, ali nije isto kod dijeljenja!

Cjelobrojne konstante (nastavak)

Oktalne konstante:

- sadrže znamenke `0-7`, dozvoljen predznak `-` ili `+`,
- prva znamenka je uvijek `0`.

Primjer.

```
0,  01,  0235, -0651,  077777
```

Cjelobrojne konstante (nastavak)

Heksadecimalne konstante:

- sadrže znamenke `0-9`, dozvoljen predznak `-` ili `+`,
- mala slova `a-f` ili velika slova `A-F`,
- ova slova su heksadecimalne znamenke:
  `a = 10`, `b = 11`, `c = 12`, `d = 13`, `e = 14`, `f = 15`,
- uvijek počinju s `0x` ili `0X`.

Primjer.

```
0x0,  0x1,  -0x7FFF,  0X1FFF,
0xabcd,  0XABCD,  -0x23aa
```

Cjelobrojne konstante (nastavak)

Ako specijalno ne navedemo drugačije, onda konstanta ima tip `int` (svi prošli primjeri).

Konstante ostalih cjelobrojnih tipova definiraju se

- dodavanjem sufiksa (na kraju konstante).

Konstanta tipa:

- `long` — formira se tako da se na kraj cjelobrojne konstante doda slovo `L` (veliko ili malo),
- `unsigned` — formira se dodavanjem slova `U` (veliko ili malo), nema predznaka,
- `unsigned long` — formira se dodavanjem slova `U` i `L` (veliko ili malo, u bilo kojem poretku), nema predznaka.

Cjelobrojne konstante (nastavak)

Primjer.

```
500000U             /* unsigned (decimalna) */
123456789L          /* long (decimalna) */
123456789ul         /* unsigned long (decimalna) */
123456789LU         /* unsigned long (decimalna) */
01234561            /* long (oktalna) */
0X50000U            /* unsigned (heksadecimalna) */
123456789012ull     /* unsigned long long (dec.) */
```

Oznake "konverzije" za formatirano čitanje i pisanje (zasad):

- `%d`, `%ld`, `%lld` (decimalno),
- `%u`, `%lu`, `%llu` (unsigned).

Znakovne konstante (tipa char)

Znakovna konstanta (tipa `char`) je

- jedan znak napisan u jednostrukim navodnicima `'`.

Primjer.

```
'A',  'x',  '5',  '?',  ' '
```

Zadnji znak je razmak (praznina, blank ili space).

Svi objekti tipa `char`, pa tako i znakovne konstante, prikazuju se kao cjelobrojne vrijednosti (bez predznaka).

- Taj cijeli broj odgovara (jednak je) kôdu znaka u odgovarajućem načinu prikazivanja znakova (na pr. ASCII kôd).

Zadavanje znaka kôdom

Bilo koji znak može se zadati i svojim kôdom, u obliku:

- `\ooo`, gdje je `ooo` troznamenkasti oktalni broj, ili
- `\xoo`, gdje je `oo` dvoznamenkasti heksadecimalni broj.

To je bitno za one znakove kojih nema na tipkovnici.

Primjer.

```
\170    /* znak s kodom 170 oktalno
           = znak s kodom 120 decimalno
           = ASCII znak 'x' */
\x78    /* znak s kodom 78 heksadecimalno
           = znak s kodom 120 decimalno
           = ASCII znak 'x' */
```

Posebni znakovi

Posebni znakovi u C-u reprezentiraju se pomoću dva znaka:

```
\b     /* idi 1 mjesto unazad (backspace) */
\f     /* nova stranica (form feed) */
\n     /* novi red (new line) */
\r     /* povratak na pocetak linije
          (carriage return) */
\t     /* horizontalni tabulator */
\v     /* vertikalni tabulator */
\0     /* nul znak (null character) */
\?     /* upitnik - svrha?, jer radi i ? */
\"     /* navodnik */
\'     /* jednostruki navodnik */
\\     /* obrnuta kosa crta (backslash) */
```

Posebni znakovi (nastavak)

```
\a    /* ASCII znak BELL ili alert */
```

Ovaj znak postoji, ali nema nikakvog efekta kad se napiše na ekran. Služio je nekad davno, na teleprinterima.

Zapis svih specijalnih znakova počinje znakom \.

Takav oblik zapisa znaka

- kôdom ili posebnim imenom,

smijemo koristiti i u znakovnim konstantama.

Primjer.

```
'\n',  '\b',  '\\',  '\170',  '\x78'
```

Konstantni znakovni nizovi (stringovi)

Konstantni znakovni nizovi (konstantni stringovi) su nizovi znakova navedeni unutar dvostrukih navodnika ".

Primjer.

```
"Zagreb"
"01/07/2001"
"Linija 1\nLinija 2\nLinija3"
```

Bitno. Zadnji znak u nizu, iza svih navedenih znakova, je nul-znak `'\0'`.

Napomena. `'a'` nije isto što i `"a"`.

- `'a'` je tipa `char` i sadrži 1 znak: `a`,
- `"a"` je niz (ili polje) od 2 znaka: `a` i `\0`.

Stringovi (nastavak)

Kako pisati dugačke stringove koji ne stanu uredno u jedan red programa? Imamo dvije mogućnosti:

- koristimo znak \ na kraju linije, kao oznaku da će se string nastaviti na početku sljedećeg reda, ili
- rasječemo znakovni niz u nekoliko nizova, a oni će se nadovezati (spojiti, ili konkatenirati) u jedan.

Primjer. String `s1` jednak je stringu `s2`.

```
    char s1[] = "Vrlo dugacak \
niz znakova";

    char s2[] = "Vrlo dugacak "
                "niz znakova";
```

Realne konstante

Realna konstanta je broj zapisan u dekadskom sustavu koji:

- sadrži decimalnu točku,
- i/ili eksponent baze 10. Tada decimalna točka nije potrebna. Eksponent mora biti cijeli broj kojem prethodi slovo `e` (malo ili veliko).

Primjer. Zapis realnih konstanti s decimalnom točkom.

```
0.    1.    -0.2    5000.0
```

Primjer. Razni mogući zapisi realne konstante 3.0×10^5.

```
300000.   3e5   3E+5   3.0e+5   .3e6   30E4
```

Realne konstante (nastavak)

Ako specijalno ne navedemo drugačije, onda realna konstanta ima tip `double` (svi prošli primjeri).

Konstante ostalih realnih tipova definiraju se dodavanjem sufiksa (na kraju konstante).

Za konstantu tipa:

- `float` — na kraju treba dodati `f` ili `F`,
- `long double` — na kraju treba dodati `l` ili `L`.

Primjer.

```
3.f     0.337f     -3.e4f     3e-3f     1.345E-8F
-0.2e3l     5000.0L
```

Realne konstante (nastavak)

Oznake "konverzije" za formatirano čitanje (zasad):

- `%g` za `float`,
- `%lg` za `double`,
- `%Lg` za `long double`.

Oznake "konverzije" za formatirano pisanje (zasad):

- `%g` za `double` (`float` se uvijek pretvara u `double`),
- `%Lg` za `long double`.

Simboličke konstante

Simboličke konstante su imena koja pretprocesor zamjenjuje zadanim nizom znakova.

- Definiraju se najčešće na početku programa.
- Svrha: olakšavaju razumijevanje programa (čitljivost).

Sintaksa (pravilo pisanja):

```
#define ime tekst
```

gdje je:

- `ime` — ime simboličke konstante, a
- `tekst` — niz znakova koji će biti doslovno substituiran umjesto `ime`, na svakom mjestu nadalje u programu na kojem se javlja `ime` (osim u konstantnim stringovima).

Simboličke konstante (nastavak)

Primjer.

```
#define PI  3.141593
#define TRUE  1
#define FALSE 0
```

Napomena. Prvi primjer je formalno korektan, ali ga matematičari ne bi trebali koristiti (zabranjujem)! Razlog:

- mala točnost `PI` ne može dati točnije rezultate!

Koristite π u punoj točnosti realnog tipa s kojim računate!

- `PI = 3.14159265358979323846264433;` (može i dalje),
- `PI = 4.0 * atan(1.0);` (ovo je sasvim dovoljno).

Za `atan` (= arctan) treba uključiti zaglavlje `<math.h>`.

Simboličke konstante (nastavak)

Uputa. Simboličke konstante treba koristiti zaista

- samo za konstante, a ne za složenije izraze.

Razlog: zbog doslovne supstitucije zamjenskog teksta, složeniji izraz se svaki puta ponovno računa.

Osim toga, može doći do neželjenih efekata (grešaka) unutar drugih izraza. Zato zamjenski izraz uvijek treba zatvoriti u okrugle zgrade.

Primjer. Ako već mora, onda ovako:

```
#define PI (4.0 * atan(1.0))
```

Uočite vanjske zagrade oko izraza za `PI`. One služe korektnom uvrštavanju u složenije izraze, poput `1/PI`. Ako ih nema . . . ?

Simboličke konstante (nastavak)

U ovakvim slučajevima, puno bolje je koristiti

- inicijalizaciju varijable i kvalifikator `const`.

Detaljni opis malo kasnije.

Primjer. "Prava" deklaracija konstante za π:

```
const double pi = 4.0 * atan(1.0);
```

Varijable

Varijable su simbolička imena za lokacije u memoriji u koje možemo pohraniti neke vrijednosti. Imaju adresu i sadržaj.

Osnovni tipovi varijabli:

- numerički (cjelobrojni, realni, ...),
- znakovni,
- pokazivači — varijable koje sadrže adrese drugih varijabli.

Ime varijable je identifikator (duljina može biti ograničena):

- počinje slovom,
- razlikuju se mala i velika slova: `a` nije isto što i `A`,
- dozvoljeni znakovi u imenu su: slova, znamenke i znak `_`,
- ključna riječ ne smije biti ime varijable.

Ime varijable

Primjer. Ispravna imena varijabli:

```
x    y2    rez_mjerenja    Program_V03    __SYSTEM
```

Savjet. Izbjegavajte imena koja počinju znakom `_` (donja crta, engl. underscore), da ne dođe do kolizije sa sistemskim ili internim imenima!

Primjer. Neispravna imena varijabli:

```
"x"    ad-c    extern    3x
```

Deklaracija varijable

Deklaracija određuje ime i tip varijable. Ima oblik:

```
tip ime;
```

gdje je `tip` = tip varijable, a `ime` = njezino ime.

Primjer.

```
int a, b;
unsigned c;
char d;
```

Deklaracija varijable (nastavak)

Varijable istog tipa moguće je deklarirati u istoj deklaraciji tipa, a varijable (imena) se odvajaju zarezom.

Primjer.

```
short a, b, c;
```

Svaku od tih varijabli možemo deklarirati i u zasebnoj deklaraciji tipa:

```
short a;
short b;
short c;
```

Inicijalizacija varijabli

Varijable se mogu inicijalizirati u trenutku deklaracije.

Sintaksa (pravilo pisanja):

```
tip varijabla = izraz;
```

Izraz se računa odmah — "na licu mjesta", tj. sve varijable u njemu moraju imati definiranu vrijednost.
Znak `=` je operator pridruživanja vrijednosti.

Primjer.

```
int a = 7, b;
unsigned c = 2345, c1 = c + 1;
char d = '\t';
```

Varijable `a`, `c`, `c1` i `d` su inicijalizirane, a `b` nije.

Deklaracija i inicijalizacija varijable

Uočite da svaka varijabla ima dva bitna dijela (ili atributa):

- adresu i vrijednost (ili sadržaj).

Deklaracija rezervira prostor, tj. "dodjeljuje" adresu varijabli. Međutim, vrijednost varijable nije definirana, osim kad

- eksplicitno inicijaliziramo varijablu u deklaraciji.

```
int a = 7, b;
```

Varijabla `a` je deklarirana i inicijalizirana — njezina adresa `&a` ima vrijednost i sama varijabla `a` ima vrijednost.

Za razliku od toga, varijabla `b` je deklarirana, ali ne i inicijalizirana. Dakle, `&b` ima vrijednost, ali `b` nema.

Bez deklaracije — nema adrese, a kamo li vrijednosti!

Inicijalizacija varijabli (nastavak)

Varijable se mogu inicijalizirati i kvalifikatorom `const` (ključna riječ) na početku deklaracije.

- Prevoditelj tada neće dozvoliti izmjenu vrijednosti te varijable u programu. Drugim riječima, ta "varijabla", zaista, ima konstantnu vrijednost.

Zgodno za fizikalne konstante, faktore pri pretvorbi jedinica.

Primjer.

```
const double c = 299792.458;
const double e = 2.71828182845905;
```

Zbog točnosti, opet, ima i boljih rješenja. Recimo:

- `e = exp(1.0);`. Funkcija `exp` ($= e^x$) je u `<math.h>`.

Deklaracija polja

Polje je niz varijabli istog tipa, indeksiranih cjelobrojnim indeksom u rasponu od `0` do `n - 1`, gdje je `n` broj elemenata polja.

Deklaracija polja ima oblik:

```
tip ime[dimenzija];
```

gdje je:

- `tip` = tip podataka svakog elementa polja,
- `ime` je ime polja (zajedničko ime svih elemenata), a
- `dimenzija` je broj elemenata polja.

Pojedini elementi polja razlikuju se po indeksu, koji se piše unutar uglatih zagrada.

Deklaracija polja (nastavak)

Primjer.

```
float vektor[10];
```

Elementi polja su:

```
vektor[0], vektor[1], ..., vektor[9].
```

Svaki element polja je varijabla tipa `float`.

Inicijalizacija polja

Polja se mogu inicijalizirati navođenjem popisa vrijednosti elemenata polja unutar vitičastih zagrada.

Primjer.

```
double x[] = {1.2, 3.4, -6.1};
```

Dimenzija polja se računa na osnovu broja konstanti u popisu unutar zagrada.

Prethodna deklaracija rezervira prostor za polje `x` s 3 elementa tipa `double` i inicijalizira ga na vrijednosti:

```
x[0] = 1.2,    x[1] = 3.4,    x[2] = -6.1.
```

Inicijalizacija polja (nastavak)

Polje znakova može se inicijalizirati i konstantnim znakovnim nizom (stringom), a ne samo popisom znakova.

Primjer.

```
char tekst[] = "Init";
```

Ovo definira polje od 5 znakova (dodaje se i nul-znak \0 na kraju stringa).

Ekvivalentne inicijalizacije su:

```
char tekst[5] = "Init";
char tekst[] = {'I', 'n', 'i', 't', '\0'};
char tekst[5] = {'I', 'n', 'i', 't', '\0'};
```

Deklaracija pokazivača

Pokazivači su varijable koje sadrže adrese drugih varijabli (nekog zadanog tipa).

Deklaracija pokazivača sadrži `*` (operator dereferenciranja):

```
tip *ime;
```

gdje je:

- `ime` = ime pokazivača (varijable), a
- `*` označava da identifikator `ime` nije varijabla tipa `tip`, nego pokazivač na varijablu tipa `tip` (tj. sadrži adresu varijable tipa `tip`).

Za lakše čitanje: `tip *ime` — kad dereferenciramo `ime` (uzmemo sadržaj na toj adresi) dobijemo objekt tipa `tip`.

Deklaracija pokazivača (nastavak)

Deklaracije varijabli nekog tipa i pokazivača na isti tip mogu se pisati u jednom retku:

```
double u, *pu;
```

ili u više redaka:

```
double u;
double *pu;
```

Ekvivalentne deklaracije su i ovo:

```
double *pu, u;
double* pu, u;
```

tj. operator * djeluje na prvu sljedeću varijablu!

Inicijalizacija pokazivača

Varijablu tipa pokazivač smijemo inicijalizirati adresom neke druge, već deklarirane varijable (tako da ona ima adresu).

Primjer.

```
float u = 7.5f, *pu = &u;
```

Znak `&` je unarni operator adresiranja (uzimanja adrese). Pokazivač `pu` sadrži stvarnu adresu varijable `u`.

Do vrijednosti spremljene na adresi koju sadrži pokazivač dolazimo unarnim operatorom dereferenciranja — znak `*`.

```
*pu    /* Isto sto i sadrzaj od u, tj. 7.5f */
```

Enumeracije

U C-u postoji još jedna vrsta konstanti, tzv. enumeracijske konstante. Te konstante su

- simbolička imena za cjelobrojne konstante, a pišu se kao identifikatori.

Enumeracija (nabrajanje ili pobrojani tip) je tip koji

- počinje ključnom riječi `enum`, a zatim sadrži
- popis takvih imena za cjelobrojne konstante u vitičastim zagradama (slično kao kod inicijalizacije).

Princip enumeracije (ako ne navedemo drugačije):

- prvom identifikatoru pridružuje se konstanta `0`, drugom konstanta `1`, i tako redom ...

Enumeracijske konstante

Enumeracije su alternativa uvođenju simboličkih konstanti korištenjem pretprocesorske direktive `#define`.
Svrha je ista: čitljivost programa.

Primjer.

```
enum { FALSE, TRUE };
```

Nakon te deklaracije, vrijednosti identifikatora su:

```
FALSE = 0,     TRUE = 1.
```

To je (skoro) ekvivalentno pretprocesorskim naredbama:

```
#define FALSE 0
#define TRUE  1
```

Ime tipa enumeracije

Enumeraciji možemo dati ime, odmah iza riječi `enum`.

- To ime je naziv (ime) za tip podataka koji sadrži samo vrijednosti iz te enumeracije.

Katkad se takav tip zove još i pobrojani tip.

Primjer.

```
enum logical { FALSE, TRUE };
```

Nakon toga možemo deklarirati varijable tipa te enumeracije.

Primjer.

```
enum logical x, y;
```

Varijable tipa enumeracije

Te varijable mogu poprimiti samo one vrijednosti koje su navedene u enumeraciji.

Možemo ih koristiti na sljedeći način:

```
x = FALSE;
...
if (x == TRUE) y = FALSE;
```

Deklaracija tipa enumeracije

Enumeracija se, općenito, deklarira naredbom oblika:

```
enum ime {clan_1, clan_2, ..., clan_n};
```

gdje je:

- `ime` = ime enumeracije, tj. pripadnog `enum` tipa,
- a `clan_1`, `clan_2`, ..., `clan_n` predstavljaju identifikatore koji su imena vrijednosti u tom tipu.

Te vrijednosti mogu se pridružiti varijabli tipa `enum ime`.

Napomena.

- Identifikatori koji su imena za vrijednosti u tipu moraju biti međusobno različiti.

Vrijednosti u tipu enumeracije

Ako ne zadamo (inicijaliziramo) vrijednosti za identifikatore, onda se tim identifikatorima

- automatski pridružuju cjelobrojne vrijednosti.

Početna je 0, a svaka sljedeća je za 1 veća od prethodne.

```
clan_1 = 0
clan_2 = 1
...
clan_n = n - 1
```

Varijable tipa enumeracije deklariraju se naredbom:

```
enum ime var_1, var_2, ..., var_m;
```

Tip i varijable, inicijalizacija vrijednosti u tipu

Deklaracije tipa enumeracije i varijabli tog tipa mogu se spojiti (ali nije naročito čitljivo):

```
enum ime {clan_1, clan_2, ..., clan_n}
                        var_1, var_2, ..., var_m;
```

Vrijednosti koje se dodijeljuju pojedinim identifikatorima

- mogu se modificirati — eksplicitnom inicijalizacijom.

Primjer.

```
enum esc_ch { BACKSPACE = '\b', TAB = '\t',
                NEWLINE = '\n', RETURN = '\r' };
```

Ovdje koristimo da su znakovi, zapravo, cijeli brojevi!

Inicijalizacija vrijednosti u tipu enumeracije

Primjer.

```
enum boje {plavo = -1, zuto, crveno,
           zeleno = 0, ljubicasto, bijelo};
```

Time dobivamo:

```
plavo = -1
zuto = 0
crveno = 1
zeleno = 0
ljubicasto = 1
bijelo = 2
```

Uočiti: identifikatori moraju biti različiti, ali vrijednosti ne.

Primjer enumeracije

Primjer. Uvođenje imena za mjesece u godini (umjesto brojeva) bitno poboljšava čitljivost.

```
enum month { JAN = 1, FEB, MAR, APR, MAY, JUN,
             JUL, AUG, SEP, OCT, NOV, DEC };
```

Slično može i za dane u tjednu (probajte sami).

Kod deklaracije varijabli tipa enumeracije, dosad smo stalno za tip pisali dvije riječi: `enum ime`. To se može skratiti tako

- da ovom tipu damo novo ime u `typedef` deklaraciji.

Imenovanje tipova — typedef

Opći oblik `typedef` deklaracije je:

```
typedef stari_tip novi_tip;
```

Primjer.

```
typedef float real;     /* za Pascalce */
real a, b;
enum logical { FALSE, TRUE };
typedef enum logical boolean;
boolean x, y, flag;
```

Identifikator `boolean` je sinonim (novo ime) za `enum logical`.

Može i kraće: `typedef enum { FALSE, TRUE } boolean;`

Operatori i izrazi

Sadržaj

- Operatori i izrazi (prvi dio):
  - Izraz, operandi i operatori.
  - Operator pridruživanja.
  - Aritmetički operatori.
  - Pretvaranje tipova u aritmetičkim izrazima.
  - Eksplicitno pretvaranje tipova — cast operator.
  - Redoslijed računanja izraza.
  - Prioritet i asocijativnost operatora.

Izraz

Većina stvarnog posla kod izvršavanja programa svodi se na

- računanje vrijednosti raznih izraza u programu.

Tipična mjesta na kojima se pojavljuju izrazi su:

- desna strana naredbe pridruživanja, argument funkcije,
- uvjeti u uvjetnim naredbama i petljama,
- granice u petljama.

Kako se pišu i izračunavaju izrazi?

- Pravila pisanja strukture "izraza" su najkompliciraniji dio gramatike jezika C.
- Kog zanima, može pogledati zadnji dio knjige KR2.
- Dajemo "samo" relativno detaljni opis tih pravila.

Operandi i operatori

Svaki izraz ima tip i vrijednost, a formira se od

- operanada i operatora.

Operand je objekt (vrijednost) nekog tipa. Jezički gledano, to može biti

- konstanta, varijabla, vrijednost funkcije, podizraz, itd.

Operatori djeluju na operande (određenih tipova) i daju neku vrijednost (nekog tipa) kao rezultat.

C je jezik bogat operatorima, jer podržava sve operacije koje se mogu izvesti na modernim računalima.

U ovom pregledu, preskačemo operatore vezane uz strukture i pokazivače. Njih obrađujemo kasnije.

Operator pridruživanja

Osnovni operator pridruživanja je `=`. Prioritet mu je

- niži od većine ostalih operatora (`,` je izuzetak).

To je zato da naredba pridruživanja, oblika

```
varijabla = izraz;
```

prvo izračuna izraz na desnoj strani, a onda tu vrijednost pridruži varijabli na lijevoj strani operatora.

Primjer.

```
x = 3.17;
y = x + 5.342;
a = a + 1;
c = 'm';
```

Aritmetički operatori

U programskom jeziku C postoji 5 aritmetičkih operatora:

operator	značenje
`+`	zbrajanje, unarni plus
`-`	oduzimanje, unarni minus
`*`	množenje
`/`	dijeljenje
`%`	ostatak (modulo)

Operatori `-` i `+` imaju dva različita značenja, ovisno o tome

- kako se operator napiše — obzirom na operand(e).

Aritmetički operatori (nastavak)

Operator promjene predznaka `-` je unarni operator i

- piše se ispred operanda

(tzv. prefiks notacija).

```
-operand
```

Slično vrijedi i za unarni `+` (ne mijenja predznak).

Ostali aritmetički operatori su binarni i

- pišu se između dva operanda

(tzv. infiks notacija).

```
operand_1 operacija operand_2
```

Aritmetički operatori (nastavak)

Aritmetički operatori djeluju na numeričke operande raznih tipova. Operandi mogu biti:

- nekog cjelobrojnog tipa,
- nekog realnog tipa,
- znakovnog tipa (`char` se prikazuje kao cijeli broj).

Problem. Kad operandi nisu istog tipa, kojeg tipa je rezultat?

- Tada dolazi do konverzije ili pretvaranja tipova po određenim pravilima (v. malo kasnije).

Za početak, pogledajmo kako radi cjelobrojno dijeljenje!

Cjelobrojno dijeljenje

Operacija dijeljenja `/`, u slučaju kad su

- oba operanda cjelobrojna,

daje cjelobrojan rezultat (operacija div od ranije).

Po C99 standardu, rezultat se uvijek dobiva zaokruživanjem kvocijenta prema nuli. Dakle, vrijedi:

```
3/2 = 1     -3/2 = -1
```

Po C90 standardu, to vrijedi za pozitivne operande. Inače, ako je bar jedan operand negativan, rezultat ovisi o implementaciji.

Napomena. Ako je bar jedan operand realan broj, dijeljenje je uobičajeno dijeljenje realnih brojeva, pa je `3.0/2 = 1.5`.

Cjelobrojni ostatak (modulo)

Operator `%` (modulo) djeluje samo na cjelobrojnim operandima i kao rezultat daje

- cjelobrojni ostatak pri cjelobrojnom dijeljenju operanada.

(operacija mod od ranije).

Primjer. Za `x = 10` i `y = 3` dobivamo

```
x / y = 3        x % y = 1
```

Ostatak se računa tako da uvijek vrijedi (osim za `y == 0`):

```
(x / y) * y + x % y == x
```

Dakle, ostatak ima predznak prvog operanda.

Veza cjelobrojnog i običnog dijeljenja

Ponavljanje: $a = q \cdot b + r$

- kvocijent q se uvijek "zaokružuje" prema nuli,

$$\texttt{a / b} = a \text{ div } b = q = \text{sign}\left(\frac{a}{b}\right) \cdot \left\lfloor \left| \frac{a}{b} \right| \right\rfloor,$$

- ostatak r ima isti predznak kao i a.

$$\texttt{a \% b} = a \bmod b = r = \text{sign}(a) \cdot (|a| \bmod |b|).$$

Za ostatak r ovdje vrijedi:

- ako je $a \geq 0$, onda je $r \in \mathbb{Z}_b$, tj. $0 \leq r < |b|$,
- ako je $a < 0$, onda je $r \in -\mathbb{Z}_b$, tj. $-|b| < r \leq 0$.

Onda je $(a \text{ div } b) * b + a \bmod b = a$, za svaki $b \neq 0$.

Konverzije (pretvaranja) tipova

U C programu, svaka vrijednost ima svoj tip. Na određenim mjestima u programu pojavljuje se potreba za pretvaranjem vrijednosti iz jednog tipa u neki drugi tip.

Konverzije ili pretvaranja tipova događaju se automatski (po C standardu) na sljedećim mjestima.

- U aritmetičkim izrazima, kad neka operacija djeluje na operande različitog tipa.
- U operaciji pridruživanja, ako tip lijeve strane nije isti kao tip desne strane.
- Pri prijenosu argumenata u funkciju, ako su stvarni i formalni argument različitog tipa.
- Pri prijenosu vrijednosti iz funkcije na mjesto poziva, ako je vraćena vrijednost različitog tipa od deklariranog.

Konverzije u aritmetičkim izrazima

Kad aritmetički operator ima dva operanda različitog tipa, onda dolazi do

- konverzije ili pretvaranja tipova.

U pravilu:

- prije operacije, operand "nižeg" ili "užeg" tipa se promovira, odnosno, pretvara u "viši" ili "širi" tip,
- zatim se izvršava operacija — sad na operandima istog tipa,
- rezultat operacije ima taj isti (zajednički) tip.

Ovo pretvaranje se radi onim redom kojim se izvršavaju operacije u izrazu, tj. po prioritetu operacija (v. kasnije).

Konverzije u aritmetičkim izrazima (nastavak)

Pravila konverzije tipova u aritmetičkim izrazima:

Operandi tipa `short` i `char` (s predznakom ili bez njega) automatski se konvertiraju u `int` ili `unsigned int`, i to prije svake aritmetičke operacije.

- Ako je `short` kraći od `int`, onda konverzija ide u `int`. (Ovo je najčešći slučaj u modernim realizacijama C-a.)
- Ako je `short` isto što i `int`, onda je `unsigned short` širi od `int`, pa konverzija ide u `unsigned int`.

U svakoj operaciji koja uključuje operande različitih tipova, prije izvršenja operacije, vrši se konverzija operanada u širi tip (od ta dva).

Konverzije u aritmetičkim izrazima (nastavak)

Tipovi su, prema širini, poredani na sljedeći način, od najšireg prema najužem:

- `long double`,
- `double`,
- `float`,
- `unsigned long long`,
- `long long`,
- `unsigned long`,
- `long`,
- `unsigned int`,
- `int`.

Konverzije u aritmetičkim izrazima (nastavak)

Jedina iznimka je kad su `long` i `int` isti (najčešće je tako).

- Tada je `unsigned int` isto što i `unsigned long`, pa je širi od `long`,
- tj. možemo uzeti da tablica vrijedi za "jači" tip `long`.

Uži tipovi od `int` se ovdje ne pojavljuju, jer se oni automatski konvertiraju u `int` ili `unsigned int` (prema prvom pravilu).

Napomena.

- Kod ovih konverzija nema gubitka informacije, osim, eventualno, kod pretvaranja cjelobrojnog tipa u realni.

(Na primjer, "dugački" cjelobrojni u "kratki" realni tip.)

Konverzije kod pridruživanja

U operaciji pridruživanja dolazi do konverzije, ako tip lijeve strane nije isti kao tip desne strane. Tada se:

- operand na desnoj strani konvertira u tip operanda na lijevoj strani.

Pri tome može doći do gubitka informacije,

- ako se širi tip konvertira u uži.

Najčešći primjer je pretvaranje realnog u cjelobrojni tip.

- To se radi "odbacivanjem", tj. zaokruživanjem prema nuli.

Na primjer, ako je `x` varijabla tipa `float` i `n` varijabla tipa `int`, prilikom pridruživanja `n = x`, doći će do odsjecanja decimala u broju `x`.

Konverzije tipova — primjer

Primjer.

```
double x = 2.0;
float y = 3.0f;
int z;
...
z = x + y;
```

Ovdje imamo dvije konverzije tipova:

- prvo se `y`, prije zbrajanja, pretvara iz `float` u `double`,
- rezultat zbrajanja je tipa `double`,
- a zatim se u naredbi pridruživanja, taj rezultat pretvara u tip `int` varijable `z` na lijevoj strani naredbe (zaokruživanjem prema nuli).

Konverzije kod prijenosa argumenata

Do konverzije pri prijenosu argumenata u funkciju dolazi ako su stvarni i formalni argument različitog tipa.

Naime, stvarni argument je, općenito, izraz. Kad se izračuna njegova vrijednost, taj rezultat ima neki tip.

- Ako taj tip nije isti kao i tip pripadnog formalnog argumenta, dolazi do konverzije.

Pravilo ovisi o tome ima li funkcija prototip (zaglavlje) ili ne.

- Ako funkcija nema prototipa, onda se svaki argument tipa `char` i `short` konvertira u `int`, a `float` u `double`.
- Ako funkcija ima prototip, onda se svi stvarni argumenti pri pozivu konvertiraju (ako je to potrebno) u tipove deklarirane u prototipu.

Konverzije kod prijenosa argumenata — primjer

Primjer.

```
void f(float);
...
f(2.4);
```

Ovdje imamo konverziju tipova.

- Stvarni argument funkcije je konstanta `2.4` (`double`).
- U prototipu funkcije, pripadni formalni argument je tipa `float`.
- Dolazi do konverzije iz tipa `double` u tip `float` i funkcija radi s `2.4f`.

Usput dolazi i do gubitka točnosti!

Eksplicitne konverzije — cast operator

Vrijednost nekog izraza može se eksplicitno pretvoriti u željeni tip, tako da

- ispred izraza, u zagradama navedemo ime tog tipa.

Sintaksa:

```
(tip_podataka) izraz
```

Ovdje je:

- `(tip_podataka)` tzv. cast operator, ili operator ekplicitne konverzije tipa (unarni).

Prvo se računa vrijednost izraza, a onda se radi konverzija.

Ako `izraz` sadrži druge operatore, treba ga zatvoriti u zagrade — `(izraz)`, zbog visokog prioriteta "cast" operatora.

Eksplicitne konverzije — primjeri

Primjer.

```
double x;
float y;
...
x = (double) y;
```

Vrijednost varijable y eksplicitno se pretvara u tip `double` (naravno, točnost te vrijednosti se nije povećala sama od sebe).

Napomena. Eksplicitna konverzija ovdje nije potrebna. Ista konverzija se radi po standardnim pravilima.

Usput, u C-u je pogrešno napisati: `x = double(y);` (dozvoljeno u C++).

Eksplicitne konverzije — primjeri (nastavak)

Primjer.

```
double x;
int i, j;
...
j = ((int) (i + x)) % 2;
```

Ovdje je eksplicitna konverzija nužna, jer:

- operator `%` djeluje na cjelobrojne operande, a
- vrijednost izraza `i + x` ima tip `double`.

Uočiti zagrade oko izraza koji se konvertira. One su nužne, zbog visokog prioriteta "cast" operatora (v. malo kasnije).

"Vanjske" zagrade ne trebaju, osim za čitljivost.

Eksplicitne konverzije — primjeri (nastavak)

Prototip funkcije `sqrt` za drugi korijen iz matematičke biblioteke (zaglavlje `<math.h>`) je:

```
double sqrt(double);
```

Ako je `n` cjelobrojna varijabla tipa `int`, možemo pisati

```
x = sqrt((double) n);
```

za računanje $x = \sqrt{n}$. Međutim, ovdje eksplicitna konverzija nije nužna, zbog pravila o konverziji kod prijenosa argumenata u funkciju. Iz istog razloga, korektno radi i

```
x = sqrt(2);
```

Redoslijed računanja izraza

Redoslijed računanja operacija u nekom izrazu određen je prioritetom pojedinih operatora.

- Svi operatori grupirani su hijerarhijski u grupe, prema svom prioritetu.
- Operatori višeg prioriteta izvršavaju se prije onih s nižim prioritetom.
- Kad imamo više operatora istog prioriteta, redoslijed izvršavanja određen je
  - smjerom asocijativnosti te grupe operatora.
- Obične zagrade ( ) služe za promjenu redoslijeda izvršavanja, tako da se uvijek
  - prvo računa podizraz u zagradama.

Prioritet operacija

Za aritmetičke operatore vrijede standardna pravila prioriteta:

- multiplikativni operatori (`*`, `/`, `%`) imaju viši prioritet od aditivnih (`+`, `-`),
- unarni operator `-` promjene predznaka ima viši prioritet od svih binarnih.

Operator pridruživanja `=` ima niži prioritet od većine ostalih operatora, zato da naredba pridruživanja

```
varijabla = izraz;
```

(ujedno i izraz) radi onako kako očekujemo:

- prvo izračunaj izraz na desnoj strani,
- a onda pridruži tu vrijednost objektu na lijevoj strani.

Prioritet operacija — primjeri

Primjer.

```
x = 2 + 4 / 2;
```

daje rezultat `x = 4`, dok

```
x = (2 + 4) / 2;
```

daje rezultat `x = 3`.

Dakle, zagrade imaju "najviši" prioritet (kao u matematici).

Asocijativnost operatora

Već smo rekli da zagrade koristimo zato da bismo

- eksplicitno grupirali operande oko operatora.

Ako nema zagrada, operandi se grupiraju oko operatora koji (tog trena) ima najviši prioritet.

Kad više operatora ima isti (trenutno najviši) prioritet,

- redoslijed izvršavanja određen je pravilom asocijativnosti te grupe operatora.

Asocijativnost može biti:

- slijeva nadesno, oznaka L $\rightarrow$ D (uobičajeno, kako čitamo),
- zdesna nalijevo, oznaka D $\rightarrow$ L (obratno od čitanja).

Asocijativnost operatora — primjer

Ako imamo dva operatora istog prioriteta, čija je asocijativnost slijeva nadesno (L → D), onda se

- operandi prvo grupiraju oko lijevog operatora.

Primjer. Aditivni operatori imaju uobičajenu asocijativnost L → D, pa je

```
a - b + c
```

ekvivalentno s

```
(a - b) + c
```

a ne s `a - (b + c)`!

Baš zato je asocijativnost aritmetičkih operatora L → D.

Trenutna tablica prioriteta operatora

Kategorija	Operatori	Asocijativnost
unarni	`+ - * & (type)`	D → L
aritm. mult.	`* / %`	L → D
aritm. adit.	`+ -`	L → D
pridruživanje	`=`	D → L

Objašnjenje za asocijativnost operatora pridruživanja `=` (D → L) ide malo kasnije.

Asocijativnost svih unarnih operatora je D → L, zato što se obično pišu ispred operanda (prefiks notacija). Izuzetak su `++` i `--`, koji se mogu pisati i iza operanda (postfiks notacija).

Razmislite zašto to mora biti tako, tj. ispred povlači D → L.

Nema pravila za redoslijed operanada!

Napomena. Kod binarnih operatora nema pravila kojim redom se računaju operandi — osim za logičke operatore i, ili.

Sasvim općenito, u primjeru

```
izraz_1 operacija izraz_2
```

nema pravila po kojem bi se

- `izraz_1` trebao izračunati prije `izraz_2`.

Dozvoljeno je i obratno. Zato, oprez!

To, uglavnom, nije neki problem za "obične" izraze, ali može "biti svega" u složenijim izrazima. Na primjer,

- kod poziva funkcija koje mijenjaju okolne (globalne) varijable — recimo, `f(x) + g(y)`.