Programiranje 1

Predavanje 06 - Osnovni elementi C-a, konstante, varijable, enumeracije. Operatori i izrazi — prvi dio

Matej Mihelčić

Prirodoslovno-matematički fakultet
Matematički odsjek

31. listopada 2023.

Skup znakova u C-u

Dopušteni znakovi u programskom jeziku C su:

- velika i mala slova engleske abecede A-Z i a-z,
- numeričke znamenke - dekadske $0 - 9$,
- specijalni znakovi:

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

Bjelina podrazumijeva: bjelinu (blank), horizontalni i vertikalni tabulator, znakove za prijelaz u novi red, na novu stranicu i vraćanje na početak reda.

Razmak ili bjelina

Razmak ili bjelina (eng. blank) služi odvajanju pojedinih riječi ili drugih cjelina u jeziku (kao separator). Isto vrijedi i na kraju teksta programa.

U programu možemo imati i više uzastopnih bjelina - prevodioc **preskače** (ignorira) višak separatora.

Preskakanje bjelina se **ne provodi** ukoliko se one nalaze **unutar stringa**, tamo se svaka bjelina interpretira kao znak (**podatak**).

Komentari

Komentari se koriste za dokumentiranje koda (objašnjavanje funkcionalnosti, ulaznih parametara, izlaza funkcija, generalne namjene programa, specificiranje programera, institucije i slično).

Prevodioc programskog jezika `C` preskače komentare pri prevođenju. Standardni komentar u `C`-u (sadržan u svim standardiziranim verzijama jezika):

- započinje parom znakova /*
- završava prvim sljedećim parom */

Može sadržavati više linija teksta zato se zove i **blokovski komentar**.

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

Komentari

Tipična greška koja nastaje korištenjem blokovskih komentara je **ispuštanje graničnika**. U takvim slučajevima je moguć gubitak koda.

```
1 /* Ovo je prvi komentar. Nezatvoren!!!
2 x = 12.0;
3 /* Ovo je drugi komentar. */
```

U primjeru iznad se dio koda `x = 12.0;` smatra dijelom komentara (zbog toga prevodioc **preskače** navedenu naredbu).

Nije dozvoljeno pisati komentar unutar komentara (dolazi do pogreške pri prevođenju).

```
1 /*
2 x = 12.0; /* Inicijalizacija */
3 y = 21.0;
4 */
```

Komentari

Prvi komentar završava prvim sljedećim parom znakova */ (kraj druge linije). Zbog toga se par znakova /* u drugoj liniji smatra dijelom prvog komentara. Dolazi do greške zato što za par znakova */ u četvrtoj liniji ne postoji odgovarajući par znakova /* koji bi predstavljao početak komentara (*/ sam po sebi nije valjana konstrukcija u jeziku C).

Standard C99 dozvoljava skraćeni (**linijski**) oblik komentara (još se zove i C++ tip komentara).

- Počinje parom znakova //
- Završava krajem linije.

Nije dozvoljen u starijim verzijama standarda (prevoditelji koji implementiraju te standarde će javiti grešku).

```
1  x = 3.14; //Inicijalizacija
```

Identifikatori

Identifikatori su **imena** koja pridružujemo različitim elementima programa (npr. varijablama, poljima, funkcijama).

Pravila za pisanje identifikatora:

- sastoje se od **slova** i **znamenki** (alfanumeričkih znakova), **prvi** znak mora biti slovo.
- Velika i mala slova se **razlikuju**.
- Znak _ (donja crta) se smatra **slovom**.

Duljina identifikatora je proizvoljna (katkad ograničena na 255 znakova). 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**. `C` ima 32 identifikatora.

`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

Primjeri ispravno napisanih identifikatora.

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

Primjeri neispravno napisanih identifikatora.

```
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 osnovnih tipova podataka u `C`-u su **ključne riječi**.

- `int` - **cjelobrojni** podatak. Tipično zauzima 4 bajta.
- `char` - **znakovni** podatak. Sadržava 1 znak i 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 ili 8 bajtova.

Kratki i dugi tip int

Cjelobrojni tip `int` može se modificirati pomoću **kvalifikatora** `short` i `long`.

- `short int` ili `short` je **kratki** cjelobrojni podatak. U memoriji zauzima **manje** ili jednako prostora ($\leq$) od tipa `int` pa mu je i **manji** (jednak) raspon prikazivih cijelih brojeva.
- `long int` ili `long` je **dugi** cjelobrojni podatak. U memoriji zauzima **više** ili jednako prostora ($\geq$) kao i tip `int` pa mu je i **veći** (jednak) raspon prikazivih cijelih brojeva.

Ponekad je dozvoljen i tip `long long`, **vrlo dugi** cijelobrojni podatak.

Tipovi za brojeve bez predznaka

Cjelobrojne tipove za brojeve bez predznaka dobivamo upotrebom **kvalifikatora** `unsigned`. Zauzimaju **isti memorijski prostor** kao **osnovni tipovi** podataka (`int`, `short`, `long`). Reprezentiraju **samo nenegativne cijele brojeve**.

Imaju otprilike **dvostruko veći raspon pozitivnih cijelih brojeva** od osnovnih tipova.

Cjelobrojni tipovi za brojeve **bez predznaka:**

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

Veličine cjelobrojnih tipova

`C` propisuje samo **minimalnu duljinu** cjelobrojnih tipova (npr. `int` minimalno 16 a `long` minimalno 32 bita).

Broj bajtova korištenih za prikaz određenog tipa dobivamo korištenjem **operatora** `sizeof`.

`sizeof(short)` $\leq$ `sizeof(int)` $\leq$ `sizeof(long)`.

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

Tipovi char i signed char

Osnovni tip `char` služi primarno za prikaz **znakova**. Znakovi se prikazuju ASCII kodom (**vrlo kratki** cijeli broj bez predznaka). Zauzima 1 bajt, uz raspon 0 do 255.

Može se **modificirati** korištenjem **kvalifikatora** `signed`, tako da tip `signed char` sadrži vrlo kratke cijele brojeve s predznakom. Standardno zauzima 1 bajt uz raspon -128 do 127.

Tip `signed char` se koristi kad treba *gusto* pakirati podatke u **složenijim strukturama**, npr. pri komunikaciji sa specijalnim vanjskim uređajima. Osim **uštede memorije**, nema veću ulogu pri računanju zbog **automatskog pretvaranja tipova**.

Realni tipovi

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

- `long float` = `double`,
- `long double` ima četverostruku preciznost (ako postoji).

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

Operandi u izrazima

Osnovni **operandi** u jeziku `C` su: a) konstante, b) varijable.
Možemo imati i **vrijednosti funkcija**.

Operandi imaju: a) ime, b) vrijednost, c) tip.
Ime varijable je ujedno i **identifikator**.

Konstante nemaju posebno ime (njihova vrijednost je njihovo ime).

Cjelobrojne konstante

Cjelobrojne konstante mogu biti zapisane u tri brojevna sustava: a) decimalnom (baza 10), b) oktalnom (baza 8), c) heksadecimalnom (baza 16).

Baza zapisa prepoznaje se po **početnim znakovima** zapisa konstante (početna 0 za oktalnu, $0x$ ili $0X$ za heksadekadsku).

Binarni zapis konstanti nije predviđen (predugačak zapis).

Decimalne (dekadske) konstante:
sadrže znamenke $0 - 9$, dozvoljeni predznaci $+, -$, ako konstanta ima više od jedne znamenke, prva znamenka nije 0.

```
0 12 234 12312 -245232
```

Nije dozvoljeno koristiti decimalnu točku (dobijemo realni broj). 17 po tipu (`int`) nije isto kao 17. (`double`). Razlike pri dijeljenju.

Cjelobrojne konstante

Oktalne konstante:
sadrže znamenke $0 - 7$, dozvoljeni predznaci $+, -$, prva znamenka je **uvijek** 0.

```
0 01 0234 012312 -0245232
```

Heksadecimalne konstante:
sadrže znamenke $0 - 9$, mala slova a-f, velika slova A-F, dozvoljeni predznaci $+, -$. Slova predstavljaju znamenke $a = 10$, $b = 11$, $\ldots$, $f = 15$. Uvijek počinju s $0x$ ili $0X$.

```
0x0, 0x1, -0x7FBF, 0X1FCF, 0xabcd, 0XABCD, -0x23bb
```

Ako specijalno ne navedemo drugačije, konstanta ima tip `int`).
Konstante **ostalih** cjelobrojnih tipova definiraju se **dodavanjem sufiksa** (na kraju konstante).

Cjelobrojne konstante

- `long` — na kraj konstante dodamo L ili l,
- `unsigned` — na kraj konstante dodamo U ili u,
- unsigned long — na kraj konstante dodamo U (ili u) i L (ili l) (u bilo kojem poretku).

Primjeri konstanti.

```
310000U /* unsigned (decimalna) */
123456787L /* long (decimalna) */
123456782ul /* unsigned long (decimalna) */
123456782LU /* unsigned long (decimalna) */
01234571 /* long (oktalna) */
0X3000FU /* unsigned (heksadecimalna) */
123456789012ull /* unsigned long long (dec.) */
```

Oznake *konverzije* za formatirano čitanje i pisanje:
%d, %ld, %lld (decimalno),
%u, %lu, %llu (unsigned).

Znakovne konstante

Znakovna konstanta je jedan znak napisan u **jednostrukim** navodnicima.

```
'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** (označavaju kod znaka, npr. `ASCII` kod).

Zadavanje znaka kodom

Proizvoljni znak se može zadati i svojim kodom, u obliku:

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

Koristimo za prikaz znakova kojih nema na tipkovnici.

Primjeri kodova.

```
\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 se u C-u reprezentiraju 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 - radi i ? */
\" /* navodnik */
\' /* jednostruki navodnik */
\\ /* obrnuta kosa crta (backslash) */
\a /* ASCII znak BELL ili alert */
```

\a nema nikakvog efekta kod zapisa na ekran. Koristio se na **teleprinterima**. Zapis **svih specijalnih znakova** počinje s \.

Teleprinter

Konstantni znakovni nizovi (stringovi)

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

Oblik zapisa s \ smijemo koristiti unutar znakovnih konstanti ('\n', '\b', '\\', '\170', '\x78') pa tako i unutar konstatnih znakovnih nizova.

Primjeri konstatnih znakovnih nizova.

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

Zadnji znak u nizu je **nul-znak** ('\0').

'a' nije isto što i "a". 'a' je tipa `char` i sadrži 1 znak (a), dok je "a" niz (ili polje) od 2 znaka (a i \0).

Konstantni znakovni nizovi (stringovi)

Kako pisati dugačke stringove koji ne stanu uredno u jedan red programa?

Dvije mogućnosti:

- koristimo znak \ na kraju linije, kao oznaku da će se string nastaviti na početku sljedećeg reda,
- rastavimo znakovni niz u nekoliko nizova koji se nadovežu (spoje, ili konkateniraju) u jedan.

Primjeri razlamanja stringa.

```
1 char s1[] = "Vrlo␣dugacak␣\
2 niz␣znakova";
3 char s2[] = "Vrlo␣dugacak␣"
4             "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).

```
0. 1. -0.5 1234.345
```

```
300000. 3e5 3E5 3.0e+5 .3e6 30E4
```

Ako **ne navedemo** drugačije **realna** konstanta ima tip `double`.

Konstante **ostalih** realnih tipova definiraju se **dodavanjem sufiksa** na **kraju** konstante (`float` - `f` ili `F`, `long double` - `l` ili `L`).

```
3.f 0.337f -3.e4f 3e-3f 1.345E-8F -0.2e31 5000.0L
```

Realne konstante

Oznake konverzije za formatirano **čitanje**:

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

Oznake konverzije za formatirano **pisanje**:

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

Simboličke konstante

Simboličke konstante su imena koje **pretprocesor** zamjenjuje zadanim nizom znakova. Najčešće se definiraju na **početku** programa. Svrha: **olakšavaju** razumijevanje programa (čitljivost).

Sintaksa pisanja simboličkih konstanti.

```
1 #define ime tekst
```

- **ime** - ime **simboličke konstante**,
- **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.

Primjer simboličkih konstanti.

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

Simboličke konstante

Prvi primjer u prethodnom primjeru je korektan ali ima **premalu točnost**.

Alternativa: $PI = 3.14159265358979323846264 33$; ili $PI = 4.0 * atan(1.0)$; (koristiti `math.h`).

Simboličke konstante bi trebalo koristiti **samo za konstante** a **ne za složenije izraze**.
Razlog: zbog doslovne supstitucije zamjenskog teksta, složeniji izraz se svaki puta **ponovo** računa. Može doći i do **neželjenih grešaka**, zato zamjenski izraz treba zatvoriti u **oble zagrade**.

Primjer simboličke konstante PI.

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

Bolje koristiti **inicijalizaciju** varijable i kvalifikator `const`.

```
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). Imena koja počinju _ treba **izbjegavati** da ne dođe do kolizije sa sistemskim ili internim imenima.

Deklaracija varijable

Deklaracija određuje **ime** i **tip varijable**.

```
tip ime;

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

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

Inicijalizacija varijabli

Varijable se mogu inicijalizirati u trenutku **deklaracije**.

```
1 tip varijabla = izraz;
```

Izraz se računa **odmah** (sve varijable u njemu moraju imati **definiranu vrijednost**.
Znak = je **operator pridruživanja** vrijednosti.

Primjeri inicijalizacija u trenutku deklaracije.

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

Varijable `a,c,c1,d` su **inicijalizirane**, a `b` **nije**.
Deklaracija rezervira prostor (dodjeljuje adresu varijabli), međutim **vrijednost** varijable **nije definirana** osim kad tu varijablu eksplicitno **inicijaliziramo** u **deklaraciji**.

Inicijalizacija varijabli

Varijable se mogu **inicijalizirati** i kvalifikatorom **const** (ključna riječ) na **početku** deklaracije. Prevoditelj tada **ne dozvoljava izmjenu** vrijednosti te varijable u programu (ta varijabla ima **konstantnu** vrijednost).

Primjeri konstantnih varijabli.

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

Točnije bi bilo koristiti $e = exp(1.0)$;. funkcija `exp` ($= e^x$) je definirana u zaglavlju `<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:

Deklaracija polja.

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

- **tip** - **tip** podataka svakog **elementa** polja,
- **ime** - **ime** polja (zajedničko ime svih elemenata),
- **dimenzija** - broj elemenata polja.

Pojedini **elementi** polja razlikuju se po **indeksu** koji se piše unutar **uglatih** zagrada. Kod polja: `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 inicijalizacije polja.

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

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

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

Primjer inicijalizacije polja znakova stringom.

```
1 char tekst[] = {"Init"};
```

Inicijalizacija polja

Ekvivalentne inicijalizacije polja znakova.

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

Deklarira se polje od 5 znakova i na kraj se doda **nul-znak** \0.

Deklaracija pokazivača

Pokazivači su **varijable** koje sadrže **adrese** drugih varijabli (nekog zadanog tipa). **Deklaracija** pokazivača sadrži * (operator **dereferenciranja**).

Deklaracija pokazivača.

```
1 tip *ime;
```

- **ime** - ime **pokazivača**,
- * - označava da identifikator `ime` `nije` varijabla tipa tip nego **pokazivač** na varijablu tipa **tip**.

Deklaracija varijable nekog tipa i pokazivača.

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

Deklaracija i inicijalizacija pokazivača

Deklaracija varijable nekog tipa i pokazivača.

```
1 double u;
2 double *pu;
3
4 double *pu, u;
5 double* pu, u;
```

Operator * djeluje na **prvu sljedeću** varijablu.

Inicijalizacija varijable tipa pokazivač adresom deklarirane varijable.

```
1 float u = 4.4f, *pu = &u;
```

& je unarni operator **adresiranja**.

Pristupanje vrijednosti na adresi koju sadrži pokazivač.

```
1 *pu /*ista vrijednost kao u - 4.4f*/
```

Enumeracije

Enumeracije su vrste konstanti u C-u. **Enumeracijske konstante** su **simbolička imena** za **cjelobrojne konstante**, pišu se kao **identifikatori**.

Enumeracija je **tip** koji: a) **počinje** ključnom riječi **enum**, b) sadrži **popis imena** za cjelobrojne konstante unutar **vitičastih** zagrada. Prvom identifikatoru se pridružuje konstanta 0, drugom konstanta 1 i tako redom.

Enumeracije su alternativa uvođenju konstanti koristeći `#define`.

Primjer enumeracije.

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

Vrijednosti identifikatora: `FALSE` $= 0$, `TRUE` $= 1$. Slično kao:

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

Ime tipa enumeracije

Enumeraciji možemo dati **ime** odmah iza riječi **enum** (označava **tip podatka** koji sadrži **samo vrijednosti** iz te **enumeracije**). Takav tip se još zove i **pobrojani tip**.

Imenovanje enumeracije.

```
enum logical {FALSE, TRUE};

/*mozemo deklarirati varijable gornjeg tipa*/
enum logical x,y;
//mogu poprimiti samo vrijednosti navedene u
//enumeraciji

/*Koristenje: */
x = FALSE;
if(x == TRUE) y = FALSE;
```

Enumeracije

Vrijednosti koje se dodjeljuju pojedinim **identifikatorima** mogu se **modificirati** eksplicitnom **inicijalizacijom**.

Eksplicitna inicijalizaija enumeracija.

```
enum posebni\_znakovi {BACKSPACE = '\b',
TAB = '\t', NEWLINE = '\n', RETURN = 'r'};
//znakovi su zapravo cijeli brojevi!

enum boje {plavo = -1, zuto, crveno, zeleno = 0,
ljubicasto, bijelo};
//identifikatori moraju biti razliciti ali ne i
//vrijednosti

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

Enumeracije

Uvođenje imena za mjesece u godini.

```
enum mjeseci {SIJ = 1, VELJ, OZU, TRA, SVI, LIP,
SRP, KOL, RUJ, LIST, STUD, PROS};
```

Probajte napraviti enumeraciju za **dane u tjednu**.

U svim do sada navedenim primjerima, deklaraciju varijable tipa enumeracije moramo raditi navođenjem `enum ime`. To možemo skratiti definiranjem **novog imena** tog tipa (koristeći `typedef`).

```
typedef stari_tip novi_tip; //generalno
typedef float real;
real a,b;
enum logical {FALSE, TRUE};
typedef enum logical boolean;
boolean x, y, z;
typedef enum {FALSE, TRUE} boolean; //skraceno
```

Izrazi

Izrazi su posebne jezične konstrukcije programskog jezika koje se **evaluiraju** da bi im se utvrdila vrijednost. Definiranje pravila pisanja strukture izraza čine **najkompliciraniji** dio gramatike programskog jezika `C`.

Gdje se **javljaju** izrazi?

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

Svaki **izraz** ima **tip** i **vrijednost** a formira se od **operanada** i **operatora**.

Operand je objekt (vrijednost) nekog tipa (konstanta, varijabla, vrijednost funkcije, podizraz itd.).
Operatori djeluju na **operande** (određenih tipova) i vraćaju **vrijednost** nekog tipa kao rezultat.

Operator pridruživanja

Osnovni operator **pridruživanja** je $=$. Ima niži **prioritet** od **većine** ostalih operatora. Razlog je da se kod naredbe **pridruživanja** oblika `varijabla = izraz;` prvo izračuna **izraz** na desnoj strani a onda se ta **vrijednost** pridruži varijabli na lijevoj strani operatora.

Primjeri pridruživanja.

```
1 x = 3.17;
2 y = x + 12.2;
3 a = a+1;
4 b = 'd';
5 g = 2*x+17*y - 44;
```

Aritmetički operatori

U programskom jeziku `C` postoji 5 aritmetičkih operatora:

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

Operatori $+$ i $-$ imaju dva različita **značenja** ovisno o poziciji (načinu pisanja) i operandima.

Operator **promjene** predznaka je **unarni** operator $-$. Piše se **ispred** operanda (`-operand`). Slično vrijedi i za **unarni** operator $+$ ali on ne mijenja predznak (može se koristiti za pretvorbu varijable cjelobrojnog tipa u `int`).
Ostali **aritmetički** operatori su **binarni** i pišu se **između** dva operanda (`operand_1 operacija operand_2`).

Izrazi

Aritmetički operatori djeluju na **numeričke** operande raznih tipova (**cjelobrojnog**, **realnog**, **znakovnog**).

Što ako operandi nisu istoga tipa?
Dolazi do **konverzije tipova** po određenim pravilima (navest ćemo ih kasnije).

Cjelobrojno dijeljenje i ostatak pri dijeljenju

Operacija **dijeljenja** / u slučaju kada su oba operanda **cjelobrojna** daje **cjelobrojni** rezultat. Po `C99` standardu, rezultat se uvijek dobiva zaokruživanjem kvocijenta **prema nuli** ($3/2 = 1,\ -3/2 = -1$). Po `C90` standardu isto vrijedi za **pozitivne** operande inače **ovisi o implementaciji**.

Ako je **bar jedan** operand realan broj, dijeljenje je uobičajeno dijeljenje **realnih** brojeva ($3.0/2 = 1.5,\ 3/2.0 = 1.5$).

Operator % djeluje **samo** na **cjelobrojnim** operandima i kao rezultat daje **cjelobrojni ostatak** pri **cjelobrojnom** dijeljenju operanada (za $x = 10,\ y = 3$ dobivamo `x / y = 3, x % y = 1`).

Ostatak se računa tako da uvijek za $y \neq 0$ **vrijedi**:
`(x / y) * y + x % y == x`.
Ostatak ima predznak **prvog** operanda.

Konverzije (pretvaranja) tipova

Konverzije ili **pretvaranja** tipova događaju se **automatski** 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**.

Prije operacije, operand **nižeg** ili **užeg** tipa se **promovira** (**pretvara**) u **viši** ili **širi** tip. Zatim se izvršava operacija nad operandima **istog** tipa i **rezultat** operacije ima taj **isti** (zajednički) tip.

Konverzije (pretvaranja) tipova

Pretvaranja se rade **redom izvršavanja operacija** u izrazu, tj. po **prioritetu** operacija (bit će objašnjeno kasnije).

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

- Ako `sizeof(short)<sizeof(int)`, konverzija ide u `int`. To 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`.

Poredak tipova po širini (od najužeg do najšireg) je:
`int`, `unsigned int`, `long`, `unsigned long`, `long long`, `unsigned long long`, `float`, `double`, `long double`.
Iznimka: ako je `long` jednak `int` tada je `unsigned int` isto što i `unsigned long` pa je širi od `long`.

Konverzije (pretvaranja) tipova

Uži tipovi od `int` se uvijek pretvaraju automatski u `int` ili `unsigned int`.

Kod konverzija iz **užeg** tipa u **širi** tip ne dolazi do **gubitaka** informacija osim eventualno kod pretvaranja **cjelobrojnog** tipa u **realni** (dugački cjelobrojni tip u kratki realni tip).

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.

Tim postupkom **može doći do gubitka informacija!** (pretvaranje šireg tipa u uži)

Najčešći primjer postupka konverzije uz gubitak informacija je pretvorba **realnog** tipa u **cjelobrojni** tip. To se radi **odbacivanjem decimala** (zaokruživanjem **prema nuli**).

Konverzije (pretvaranja) tipova

Primjer konverzije kod pridruživanja.

```
1 double x = 3.0;
2 float y = 4.0f;
3 int z;
4
5 z = x+y;
```

U gornjem primjeru imamo **dvije** konverzije tipova:

- y se prije zbrajanja pretvara iz `float` u `double`,
- rezultat zbrajanja je tipa `double`,
- u **naredbi pridruživanja** se rezultat pretvara u tip `int` varijable z (zaokruživanje prema nuli).

Konverzije (pretvaranja) tipova

Do konverzije pri **prijenosu argumenata** u funkciju dolazi ako su **stvarni** i **formalni** argument **različitog** tipa. **Stvarni** argument je izraz, nakon izračunavanja tog izraza se određuje tip argumenta. Ako se taj tip razlikuje od tipa pripadnog **formalnog** argumenta, dolazi do **pretvaranja**.

Pravilo pretvaranja ovisi o tome ima li funkcija **prototip** (zaglavlje) ili nema.

- Ako funkcija **nema** prototip 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 (pretvaranja) tipova

Primjer konverzije kod prijenosa argumenata.

```
1 void f(float);
2
3 f(2.4);
```

Dolazi do **konverzije** iz tipa `double` (2.4 je `double` konstanta) u tip `float` (tip **formalnog argumenta** funkcje) i funkcija radi s $2.4f$ (dolazi do gubitka točnosti).

Vrijednost izraza može se **eksplicitno** pretvoriti u **željeni tip** tako da **ispred** izraza u **zagradama** navedemo **ime** tog **tipa**.

Eksplicitna konverzija tipa.

```
1 (tip_podataka) izraz
```

`(tip_podataka)` je **unarni cast** operator ili operator **eksplicitne pretvorbe** tipa.

Konverzije (pretvaranja) tipova

Kod operatora eksplicitne konverzije, **prvo** se računa **vrijednost** izraza a onda se radi **konverzija**. Ako izraz sadrži druge **operatore** treba ga zapisati u **zagradama** `(izraz)` zato što operator pretvorbe ima **visok prioritet**.

Eksplicitne konverzije tipa.

```
1  double x;
2  float y;
3  int i,j;
4
5  x = (double) y; //(1)
6  j = ((int) (i+x))%2;//(2)
```

Kod eksplicitne konverzije (1), **vrijednost** varijable y se eksplicitno pretvara u tip `double`. Eksplicitna konverzija **nije** potrebna pošto dolazi do iste konverzije i po **standardnim** pravilima (pravila naredbe pridruživanja).

Konverzije (pretvaranja) tipova

Kod eksplicitne konverzije (2) je eksplicitna konverzija **nužna**. Razlog je što $i + x$ ima tip `double` dok operandi operacije `%` moraju imati **cjelobrojni** tip. Zagrade oko izraza koji se pretvara su nužne zbog visokog prioriteta operatora eksplicitne pretvorbe. Vanjske zagrade se mogu i **ispustiti**.

Funkcija drugi korijen iz matematičke biblioteke (`math.h`) je `double sqrt(double)`. Ukoliko funkciju želimo pozvati s argumentom tipa `int`, možemo koristiti eksplicitnu konverziju `x = sqrt((double)n);`, međutim konverzija u širi tip će se dogoditi i automatski (konverzija pri prijenosu argumenata u funkciju). Korektno radi recimo i `x = sqrt(5)`.

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.
- 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 se kod naredbe pridruživanja **prvo** izračuna **izraz** a **onda** pridruži vrijednost **objektu** na **lijevoj strani**.

Zagrade () imaju najviši prioritet izvođenja.

```
1 x = 2+4/2; //rezultat: x = 4
2 x = (2+4)/2;//rezultat: x = 3
```

Asocijativnost operatora

Zagrade () koristimo da bismo **eksplicitno grupirali** operande oko operatora. Ukoliko **nema zagrada**, operandi se grupiraju oko operatora koji (u tom trenutku) ima **najviši** prioritet.

Kada više operatora ima **isti** prioritet, **redoslijed** izvršavanja određen je pravilom **asocijativnosti** te **grupe** operatora.

Asocijativnost može biti:

- **s lijeva na desno** $L \rightarrow D$ (uobičajeno kako čitamo)
- **s desna na lijevo** $D \rightarrow L$ (obratno od čitanja)

Ako imamo dva operatora **istog** prioriteta, čija je **asocijativnost s lijeva nadesno** ($L \rightarrow D$), onda se operandi **prvo grupiraju** oko **lijevog** operatora.

Asocijativnost operatora

Aditivni operatori imaju uobičajenu asocijativnost $L \rightarrow D$.

```
a-b+c
//ekvivalentno kao
(a-b)+c
//a ne
a - (b+c)
```

Trenutna tablica prioriteta operatora

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

Asocijativnost svih **unarnih** operatora je $D \rightarrow L$. Obično se pišu **ispred** operanda (**prefiks** notacija). Izuzetak su $++$ i $--$ koji se mogu pisati i **iza** operanda (**postfiks** notacija).

Nema pravila za redoslijed operanada!

Kod **binarnih** operatora **nema pravila** kojim **redom** se računaju **operandi** - **osim** za logičke operatore: a) `i (&&)`, b) `ili (||)`.

Generalno kod binarnih operacija ne postoji određen redoslijed operanada.

```
izraz_1 operacija izraz_2
```

U gornjem primjeru **nema pravila** koje bi odredilo treba li se prije izvršiti `izraz_1` ili `izraz_2`.

To uglavnom ne stvara probleme kod jednostavnijih izraza, međutim kod složenijih izraza može izazvati poteškoće:
Kod poziva funkcija koje **mijenjaju** okolne (globalne) varijable, npr. $f(x) + g(y)$.