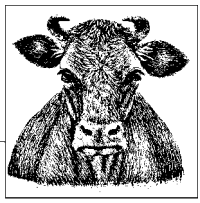
# 2

# Tipovi



Programi moraju na različite načine čuvati i obrađivati razne vrste podataka, na primer cele brojeve (engl. *integers*) i brojeve s pokretnim zarezom (engl. *floating-point numbers*). Prevodilac mora znati kakvog je tipa podataka data vrednost.

U jeziku C, pojam *objekat* (engl. *object*) označava mesto u memoriji čiji sadržaj može da predstavlja vrednost. Objekti koji imaju imena zovu se i *promenljive* (engl. *variables*). Tip objekta određuje koliko prostora objekat zauzima u memoriji i kako se kodiraju njegove moguće vrednosti. Na primer, ista šema bitova može predstavljati potpuno različite cele brojeve, zavisno od toga da li se tumači kao *označena* (engl. *signed*) – tj. pozitivna ili negativna – ili *neoznačena* (engl. *unsigned*) vrednost, u kom slučaju ne bi mogla da predstavlja negativnu vrednost.

## Tipologija

Tipovi se u jeziku C mogu klasifikovati na sledeći način:

- Osnovni tipovi
  - Standardni i prošireni celobrojni tipovi
  - Realni i kompleksni tipovi u formatu s pokretnim zarezom
- Nabrojivi tipovi
- Tip `void`
- Izvedeni tipovi
  - Pokazivači
  - Nizovi
  - Strukture
  - Unije
  - Funkcije

Osnovni i nabrojivi tipovi su takozvani *aritmetički tipovi* koji zajedno s pokazivačima predstavljaju *skalarne tipove*. Nizovi i strukture poznati su kao *agregati*. (Unije nisu agregati, jer u jednom trenutku samo jedan njihov član može da ima vrednost.)

*Tip funkcije* (engl. *function type*) opisuje interfejs funkcije, to jest tip njene povratne vrednosti, a može da određuje i tipove svih parametara koji se prosleđuju funkciji prilikom njenog pozivanja.

Svi ostali tipovi opisuju objekte. Taj opis može da obuhvata i skladišnu veličinu (engl. *storage size*) objekta: ako je određuje, to je *objektni tip* (engl. *object type*); u suprotnom, reč je o *nepotpunom tipu* (engl. *incomplete type*). Primer nepotpunog tipa mogla bi biti eksterno definisana promenljiva tipa niza:

```
extern float fArr[];     // Spoljna deklaracija
```

U prethodnom redu deklarisan je niz `fArr` sa elementima tipa `float`. Pošto ovde nije zadata veličina niza, tip niza `fArr` je nepotpun. Dokle god zadata veličina na drugom mestu u programu definiše globalni niz `fArr` – na primer, u drugoj izvornoj datoteci – ova deklaracija je dovoljna da biste koristili niz u datoj oblasti važenja. (Više detalja o spoljnim deklaracijama potražite u poglavlju 11.)

U ovom poglavlju opisujemo osnovne i nabrojive tipove i tip `void`. Izvedeni tipovi predstavljeni su od sedmog do desetog poglavlja.

Neki tipovi označeni su pomoću više rezervisanih reči, na primer tip `unsigned short`. U tim slučajevima, rezervisane reči možete napisati proizvoljnim redosledom. U ovoj knjizi koristimo konvencionalan red reči.

## Celobrojni tipovi

Postoji pet označenih celobrojnih tipova. Većina se može navesti pomoću sinonima prikazanih u tabeli 2-1.

*Tabela 2-1. Standardni označeni celobrojni tipovi*

| Tip | Sinonimi |
|---|---|
| `signed char` | |
| `int` | `signed, signed int` |
| `short` | `short int, signed short, signed short int` |
| `long` | `long int, signed long, signed long int` |
| `long long` *(C99)* | `long long int, signed long long, signed long long int` |

Svakom od pet označenih celobrojnih tipova iz tabele 2-1, odgovara neoznačen tip koji zauzima isti deo memorije i na isti način se *smešta u nju*: drugim rečima, ako prevodilac smešta objekte tipa `signed int` na parne adrese, i objekti tipa `unsigned int` raspoređuju se na parne adrese. Neoznačeni tipovi navedeni su u tabeli 2-2.

*Tabela 2-2. Neoznačeni standardni celobrojni tipovi*

| Tip | Sinonimi |
|---|---|
| `_Bool` | `bool` (definisan u zaglavlju *stdbool.h*) |
| `unsigned char` | |
| `unsigned int` | `unsigned` |
| `unsigned short` | `unsigned short int` |
| `unsigned long` | `unsigned long int` |
| `unsigned long long` | `unsigned long long int` |

U standardu C99 uveden je neoznačen celobrojni tip `_Bool` za predstavljanje vrednosti. Logička vrednost *true* (tačno) kodira se kao 1, a vrednost *false* (netačno) kao 0. Ukoliko u program uključite datoteku zaglavlja *stdbool.h*, možete koristiti i identifikatore `bool`, `true` i `false`, koji su poznati programerima na jeziku C++. Makro `bool` je sinonim za tip `_Bool`, dok su `true` i `false` simboličke konstante vrednosti 1 i 0.

Tip `char` je takođe jedan od standardnih celobrojnih tipova. Zavisno od prevodioca, ime tipa koje se sastoji od jedne reči – `char` – sinonim je za tip `signed char` ili za tip `unsigned char`. Pošto to zavisi od implementacije, `char`, `signed char` i `unsigned char` formalno su tri različita tipa.

Ako vaš program zavisi od toga da li tip `char` čuva vrednosti manje od nule ili veće od 127, trebalo bi da umesto njega koristite tip `signed char` ili `unsigned char`.

S promenljivama tipa `char` možete obavljati aritmetičke operacije. Vi odlučujete da li program tumači broj u takvoj promenljivoj kao znakovni kôd ili kao nešto drugo. Na primer, u narednom kratkom programu promenljiva tipa `char` koristi se kao ceo broj i kao znak, ali ne istovremeno:

```
char ch = 'A';               // Promenljiva tipa char.
printf("Znak %c ima kod %d.\n", ch, ch);
for ( ; ch <= 'Z'; ++ch )
  printf("%2c", ch);
```

U naredbi `printf( )`, promenljiva `ch` prvo se tumači kao znak koji se prikazuje, potom kao numerička vrednost koda datog znaka. Slično tome, u petlji `for` promenljiva `ch` tumači se kao ceo broj u instrukciji `++ch`, odnosno kao znak prilikom pozivanja funkcije `printf( )`. Na sistemima sa sedmobitnim ASCII kodom, ili proširenom verzijom tog skupa, rezultat prethodnih naredaba je:

```
Znak A ima znakovni kod 65.
 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Vrednost tipa `char` uvek zauzima jedan bajt – drugim rečima, vrednost izraza `sizeof(char)` uvek je 1 – a bajt ima najmanje osam bitova. Svaki znak u skupu osnovnih znakova može se predstaviti kao pozitivna vrednost objekta tipa `char`.

C definiše samo *minimalnu* veličinu memorijskog prostora za druge standardne tipove: `short` može imati najmanje dva bajta, tip `long` četiri bajta, dok tip `long long` ne zauzima manje od osam bajtova. Povrh toga, iako celobrojni tipovi mogu biti veći od zadate minimalne veličine, mora se poštovati sledeći odnos veličina:

```
sizeof(short) ≤ sizeof(int) ≤ sizeof(long) ≤ sizeof(long long)
```

Tip `int` je celobrojni tip koji se najbolje prilagođava arhitekturi odredišnog sistema, i njegova veličina i format bitova odgovaraju CPU registru.



Interno predstavljanje celobrojnih tipova je binarno. Označeni tipovi mogu se predstaviti kao *znak i veličina*, kao *nepotpuni komplement* (engl. *one's complement*) ili kao *potpuni komplement* (engl. *two's complement*). Najčešće se koristi potpuni komplement. Nenegativne vrednosti označenog tipa nalaze se u opsegu vrednosti odgovarajućeg neoznačenog tipa, a binarno predstavljanje nenegativne vrednosti isto je za označene i za neoznačene tipove. U tabeli 2-3 prikazane su različite interpretacije predstavljanja bitova za označene i neoznačene celobrojne tipove.

*Tabela 2-3. Binarno predstavljanje označenih i neoznačenih 16-bitnih celih brojeva*

| Binarno | Decimalna vrednost kao neoznačena celobrojna | Decimalna vrednost kao označena celobrojna u nepotpunom komplementu | Decimalna vrednost kao označena celobrojna u potpunom komplementu |
|---|---|---|---|
| 00000000 00000000 | 0 | 0 | 0 |
| 00000000 00000001 | 1 | 1 | 1 |
| 00000000 00000010 | 2 | 2 | 2 |
| ... | | | |
| 01111111 11111111 | 32,767 | 32,767 | 32,767 |
| 10000000 00000000 | 32,768 | –32,767 | –32,768 |
| 10000000 00000001 | 32,769 | –32,766 | –32,767 |
| ... | | | |
| 11111111 11111110 | 65,534 | –1 | –2 |
| 11111111 11111111 | 65,535 | –0 | –1 |

U tabeli 2-4 prikazane su veličine i opsezi vrednosti standardnih celobrojnih tipova.

*Tabela 2-4. Uobičajene veličine i opsezi vrednosti standardnih celobrojnih tipova*

| Tip | Veličina memorijskog prostora | Minimalna vrednost | Maksimalna vrednost |
|---|---|---|---|
| `char` | (isto za tipove signed char i unsigned char) | | |
| `unsigned char` | jedan bajt | 0 | 255 |
| `signed char` | jedan bajt | –128 | 127 |
| `int` | dva ili četiri bajta | –32,768<br>ili -2,147,483,648 | 32,767<br>ili 2,147,483,647 |
| `unsigned int` | dva ili četiri bajta | 0 | 65,535<br>ili 4,294,967,295 |
| `short` | dva bajta | –32,768 | 32,767 |
| `unsigned short` | dva bajta | 0 | 65,535 |
| `long` | četiri bajta | –2,147,483,648 | 2,147,483,647 |
| `unsigned long` | četiri bajta | 0 | 4,294,967,295 |

*Tabela 2-4. Uobičajene veličine i opsezi vrednosti standardnih celobrojnih tipova (nastavak)*

| Tip | Veličina memorijskog prostora | Minimalna vrednost | Maksimalna vrednost |
|---|---|---|---|
| `long long` *(C99)* | osam bajtova | –9,223,372,036, 854,775,808 | 9,223,372,036, 854,775,807 |
| `unsigned long long` *(C99)* | osam bajtova | 0 | 18,446,744,073, 709,551,615 |

U narednom primeru, svaka celobrojna promenljiva `iIndex` i `iLimit` zauzima četiri bajta na 32-bitnom računaru:

```
int iIndex,           // Definiše dve promenljive tipa int
    iLimit = 1000;    // i inicijalizuje drugu.
```

Tačnu veličinu tipa ili promenljive kazuje operator `sizeof`. Vrednost izraza `sizeof(`*`tip`*`)` i `sizeof` *`izraz`* jeste veličina memorijskog prostora za objekat ili tip u bajtovima. Ako je operand izraz, rezultat je veličina tipa rezultat izraza. U prethodnom primeru, izraz `sizeof(int)` ima istu vrednost kao izraz `sizeof(iIndex)`: u ovom slučaju 4. Izraz `iIndex` ne mora da bude u zagradi.

Opsege vrednosti celobrojnih tipova koje koristi prevodilac jezika C, možete saznati pomoću makroa `INT_MIN`, `INT_MAX`, `UINT_MAX` itd. (pogledajte poglavlje 15), iz datoteke zaglavlja *limits.h*. U programu iz primera 2-1 koristimo te makroe da bismo prikazali minimalne i maksimalne vrednosti za tipove `char` i `int`.

*Primer 2-1. Opsezi vrednosti za tipove char i int*

```
// limits.c: Prikazuje opsege vrednosti za tipove char i int.
// ------------------------------------------------
#include <stdio.h>
#include <limits.h>     // Sadrži makroe CHAR_MIN, INT_MIN itd.

int main( )
{
  printf("Skladišne veličine i opsezi vrednosti za tipove char i int\n\n");
  printf("Tip char je %s.\n\n", CHAR_MIN < 0 ? "označen" :"neoznačen");

  printf(" Veličina   tipa (u bajtovima)   Minimum         Maksimum\n"
         "-----------------------------------------------\n");
  printf(" char %8d %20d %15d\n", sizeof(char), CHAR_MIN, CHAR_MAX );
  printf(" int  %8d %20d %15d\n", sizeof(int), INT_MIN, INT_MAX );
  return 0;
}
```

U aritmetičkim operacijama s celim brojevima može doći do premašaja (engl. *overflow*) ili podbačaja (engl. *underflow*). To se dešava kada je rezultat operacije izvan opsega vrednosti datog tipa. U aritmetičkim operacijama sa neoznačenim celobrojnim tipovima, premašaji i podbačaji se ignorišu. U matematičkom smislu, to znači da je konačan rezultat celobrojne operacije jednak ostatku deljenja vrednošću *`UTYPE_MAX`* + 1, gde je *`UTYPE_MAX`* maksimalna vrednost kojom se može predstaviti dati neoznačeni tip. Na primer, naredno sabiranje dovodi do podbačaja:

```
unsigned int ui = UINT_MAX;
ui += 2;                      // Rezultat: 1
```

C definiše ovakvo ponašanje samo za neoznačene celobrojne tipove. Za sve ostale tipove, rezultat izlaženja van opsega nije definisan. Na primer, izlaženje van opsega može se zanemariti ili, ukoliko se ne obradi, može pobuditi signal kojim se program prekida.

## Celobrojni tipovi poznate širine (C99)

Širina celobrojnog tipa definisana je kao broj bitova kojim se predstavlja vrednost, uključujući označen bit. Tipične širine su 8, 16, 32 i 64 bita. Na primer, tip `int` je širok najmanje 16 bitova.

U standardu C99, celobrojni tipovi definisani su u datoteci zaglavlja *stdint.h* zbog potrebe za poznatom širinom tipa. Naveli smo ih u tabeli 2-5. Tipovi čija imena počinju slovom `u` su neoznačeni. Implementacije standarda C99 ne moraju da definišu tipove koji po priloženoj tabeli „nisu obavezni“ (tj. opcioni su).

*Tabela 2-5. Celobrojni tipovi definisane širine*

| Tip | Značenje | Implementacija |
|---|---|---|
| `int`*N*`_t`<br>`uint`*N*`_t` | Celobrojni tip čija širina iznosi tačno *N* bitova | Nije obavezan |
| `int_least`*N*`_t`<br>`uint_least`*N*`_t` | Celobrojni tip čija širina iznosi najmanje *N* bitova | Obavezan za *N* = 8, 16, 32, 64 |
| `int_fast`*N*`_t`<br>`uint_fast`*N*`_t` | Onaj tip veličine najmanje *N* bitova koji se najbrže obrađuje | Obavezan za *N* = 8, 16, 32, 64 |
| `intmax_t`<br>`uintmax_t` | Najširi implementirani celobrojni tip | Obavezan |
| `intptr_t`<br>`uintptr_t` | Celobrojni tip dovoljno širok da čuva vrednost pokazivača | Nije obavezan |

Na primer, `int_least64_t` i `uint_least64_t` celobrojni su tipovi širine najmanje 64 bita. Ako je definisan opcioni označeni tip (bez prefiksa `u`), onda je odgovarajući neoznačeni tip (s početnim slovom `u`) obavezan, i obrnuto. U narednom primeru definiše se i inicijalizuje niz elemenata tipa `int_fast32_t`:

```
#define ARR_SIZE 100
int_fast32_t arr[ARR_SIZE];       // Definiše niz arr
                                  // sa elementima tipa int_fast32_t
  for ( int i = 0; i < ARR_SIZE; ++i )
    arr[i] = (int_fast32_t)i;     // Inicijalizuje svaki element
```

Tipovi navedeni u tabeli 2-4 obično se definišu kao sinonimi postojećih standardnih tipova. Na primer, datoteka zaglavlja *stdint.h* jednog prevodioca jezika C sadrži naredni red:

```
typedef signed char    int_fast8_t;
```

U ovoj deklaraciji definiše se nov tip `int_fast8_t` (najbrži osmobitni označeni celobrojni tip) kao ekvivalent tipu `signed char`.

Implementacija može da definiše i proširene celobrojne tipove poput `int24_t` ili `uint_least128_t`.

Označeni tipovi `intN_t` imaju posebnu karakteristiku: moraju se binarno prikazivati u potpunom komplementu. Zato je njihova minimalna vrednost $-2^{N-1}$, a maksimalna $2^{N-1}-1$.

Opseg vrednosti tipova definisanih u datoteci zaglavlja *stdint.h*, takođe možete lako saznati: u njoj su definisani i makroi za najveću i najmanju vrednost koja se može predstaviti datim tipom. Imena makroa se grade od imena tipova napisanih velikim slovima, sa sufiksom `_MAX` ili `_MIN` (pogledajte poglavlje 15) umesto `_t` (za *tip*). Na primer, u narednoj definiciji inicijalizuje se promenljiva `i64` pomoću najmanje moguće vrednosti:

```
int_least64_t i64 = INT_LEAST64_MIN;
```

Datoteka zaglavlja *inttypes.h* obuhvata datoteku zaglavlja *stdint.h* i nudi dodatne elemente poput specifikatora proširenih celobrojnih tipova koji se mogu koristiti pri pozivanju funkcija `printf( )` i `scanf( )` (pogledajte poglavlje 15).

## Tipovi u formatu s pokretnim zarezom

Jezik C sadrži i posebne numeričke tipove koji mogu da predstave necelobrojne vrednosti s decimalnom tačkom na proizvoljnom mestu. Evo standardnih *tipova u formatu s pokretnim zarezom* (engl. *floating-point types*) koji se koriste u računskim operacijama s realnim brojevima:

`float`
: Za promenljive jednostruke preciznosti (engl. *single precision*)

`double`
: Za promenljive dvostruke preciznosti (engl. *double precision*)

`long double`
: Za promenljive proširene preciznosti (engl. *extended precision*)

Sačuvana vrednost u formatu s pokretnim zarezom ograničene je preciznosti, što je određeno binarnim formatom koji je za tu vrednost upotrebljen i veličinom memorije iskorišćene za njeno smeštanje. Preciznost se izražava kao broj značajnih cifara. Na primer, „preciznost od šest decimalnih cifara“ ili „šestocifrena preciznost“ znači da je binarno prikazivanje tipa dovoljno precizno za prikazivanje realnog broja od šest decimalnih cifara; ako se vrednost tog tipa konvertuje u decimalni broj, dobiće se šest izvornih cifara. Mesto decimalnog zareza (decimalne tačke) nije bitno, a vodeće i prateće nule ne spadaju u tih šest cifara. Na primer, tip sa šestocifrenom preciznošću može uspešno predstaviti brojeve 123,456,000 i 0.00123456.

U jeziku C, aritmetičke operacije s brojevima u formatu s pokretnim zarezom izvode se interno s dvostrukom ili većom preciznošću. Na primer, naredni proizvod računa se pomoću tipa `double`.

```
float height = 1.2345, width = 2.3456;  // Promenljive tipa float
                                        // su jednostruke preciznosti.
double area = height * width;           // Broj se izračunavanje
                                        // s dvostrukom (ili većom)
                                         // preciznošću.
```

Ukoliko rezultat dodelite promenljivoj tipa `float`, vrednost se, ako treba, zaokružuje. Više detalja o aritmetici brojeva u formatu s pokretnim zarezom naći ćete u odeljku „math.h“, u poglavlju 15.

Jezik C definiše samo minimalne veličine memorijske oblasti i binarni format za tipove s pokretnim zarezom. Ipak, često se koristi format koji je 1989. godine definisan u standardu IEC 60559 Međunarodne elektrotehničke komisije (International Electrotechnical Commission, IEC) za binarnu aritmetiku tipova u formatu s pokretnim zarezom. Ovaj standard je zasnovan na standardu IEEE 754 Instituta za inženjere elektronike i elektrotehnike (Institute of Electrical and Electronics Engineers) iz 1985. godine. Prevodioci jezika C mogu ukazati na to da podržavaju IEC standard za tipove u formatu s pokretnim zarezom ako definišu makro `__STDC_IEC_559__`. Tabela 2-6 prikazuje opsege vrednosti i preciznost realnih tipova u formatu s pokretnim zarezom u skladu sa standardom IEC 60559; vrednosti su prikazane u decimalnom obliku.



*Tabela 2-6. Realni tipovi u formatu s pokretnim zarezom*

| Tip | Veličina memorijske oblasti | Opseg vrednosti | Najmanja pozitivna vrednost | Preciznost |
|---|---|---|---|---|
| `float` | 4 bajta | ±3.4E+38 | 1.2E–38 | 6 cifara |
| `double` | 8 bajtova | ±1.7E+308 | 2.3E–308 | 15 cifara |
| `long double` | 10 bajtova | ±1.1E+4932 | 3.4E–4932 | 19 cifara |

Datoteka zaglavlja *float.h* definiše makroe koji omogućavaju da u programu koristite ove vrednosti, i ostale detalje o binarnom prikazivanju realnih brojeva. Makroi `FLT_MIN`, `FLT_MAX` i `FLT_DIG`, otkrivaju opseg vrednosti i jednostruku preciznost. Odgovarajući makroi za tipove `double` i `long double` počinju prefiksima `DBL_` i `LDBL_`. Ti makroi i binarno prikazivanje brojeva u formatu s pokretnim zarezom, opisani su u odeljku o datoteci zaglavlja *float.h* u poglavlju 15.

Program iz primera 2-2 najpre ispisuje tipične vrednosti za tip `float`, potom ilustruje grešku u zaokruživanju, nastalu usled pridruživanja broja u formatu s pokretnim zarezom promenljivoj tipa `float`.

*Primer 2-2. Jednostruka preciznost*

```
#include <stdio.h>
#include <float.h>

int main( )
{
  puts("\nKarakteristike tipa float\n");

  printf("Veličina memorijske oblasti: %d bytes\n"
         "Najmanja pozitivna vrednost: %E\n"
         "Najveća pozitivna vrednost: %E\n"
         "Preciznost: %d decimalnih cifara\n",
         sizeof(float), FLT_MIN, FLT_MAX, FLT_DIG);

  puts("\nPrimer preciznosti tipa float:\n");
  double d_var = 12345.6;       // Promenljiva tipa double.
  float f_var = (float)d_var;   // Inicijalizuje promenljivu tipa float
                                // pomoću vrednosti d_var.
```

*Primer 2-2. Jednostruka preciznost (nastavak)*

```
  printf("Broj u formatu s pokretnim zarezom    "
         "%18.10f\n", d_var);
  printf("čuva se u promenljivoj\n"
         "tipa float kao vrednost   "
         "%18.10f\n", f_var);
  printf("Greška pri zaokruživanju je        "
         "%18.10f\n", d_var - f_var);

  return 0;
}
```

Poslednji deo ovog programa obično daje sledeći rezultat:

```
Broj u formatu s pokretnim zarezom    12345.6000000000
čuva se u promenljivoj
tipa float kao vrednost   12345.5996093750
Greška pri zaokruživanju je            0.0003906250
```

U ovom primeru, vrednost koja se može predstaviti, a najpribližnija je decimalnom broju 12,345.6, jeste 12,345.5996093750. Ta vrednost možda ne liči na zaokružen broj u decimalnom obliku, ali u internom binarnom prikazivanju tipa u formatu s pokretnim zarezom ona je predstavljena potpuno tačno, za razliku od vrednosti 12,345.60.

## Kompleksni brojevi u formatu s pokretnim zarezom (C99)

Standard C99 podržava matematičke operacije s kompleksnim brojevima. U standardu iz 1999. godine uvedeni su kompleksni tipovi u formatu s pokretnim zarezom i dodate su aritmetičke funkcije u matematičku biblioteku. Ove funkcije deklarisane su u datoteci zaglavlja *complex.h* i sadrže, na primer, trigonometrijske funkcije `csin( )`, `ctan( )` itd. (pogledajte poglavlje 15).

Kompleksan broj $z$ može se predstaviti Dekartovim koordinatama u obliku $z = x + y \times i$, gde su $x$ i $y$ realni brojevi, a $i$ je *imaginarna jedinica*, definisana jednačinom $i^2 = -1$. Broj $x$ je realan deo broja $z$, a $y$ je imaginaran deo broja $z$.

U jeziku C, kompleksan broj je predstavljen parom vrednosti (realnim i imaginarnim delom) u formatu s pokretnim zarezom. Oba dela su istog tipa – `float`, `double` ili `long double`. U skladu s tim, postoje tri tipa kompleksnih promenljivih u formatu s pokretnim zarezom:

- `float _Complex`
- `double _Complex`
- `long double _Complex`

Sva tri tipa jednake su veličine i na isti način se smeštaju u memoriju kao niz dva elementa tipa `float`, `double` ili `long double`.

U datoteci zaglavlja *complex.h* definisani su makroi `complex` i `I`. Makro `complex` je sinonim za rezervisanu reč `_Complex`. Makro `I` predstavlja imaginarnu jedinicu *i* i tipa je `const float _Complex`:

```
#include <complex.h>
// ...
double complex z = 1.0 + 2.0 * I;
z *= I;    // Rotira z za 90° u smeru suprotnom od kretanja kazaljke na satu.
```



# Nabrojivi tipovi

*Nabrajanja* ili *nabrojivi tipovi* (engl. *enumerations*, *enumerated types*) jesu celobrojni tipovi koje programer sam definiše u programu. Definicija nabrajanja počinje rezervisanom rečju `enum`, iza koje obično sledi identifikator tipa i lista imena mogućih vrednosti:

```
enum [identifikator] { lista-vrednosti };
```

U narednom primeru definisano je nabrajanje tipa `enum boja`:

```
enum boja { crna, crvena, zelena, žuta, plava, bela=7, siva };
```

Identifikator `boja` je *identifikaciona oznaka* ovog nabrajanja. Identifikatori u listi – `crna`, `crvena` i tako dalje – predstavljaju *konstante nabrajanja* (engl. *enumeration constants)* i tipa su `int`. Ove konstante možete koristiti bilo gde u njihovoj oblasti važenja – na primer, kao konstante `case` u naredbi `switch`.

Svaka konstanta datog nabrajanja predstavlja jednu vrednost, što je određeno posredno njenom pozicijom u listi ili neposredno – inicijalizacijom pomoću konstantnog izraza. Vrednost neinicijalizovane konstante je 0 ako je prva na listi, ili za jedan veća od vrednosti prethodne konstante. Zato u prethodnom primeru navedene konstante imaju vrednosti 0, 1, 2, 3, 4, 7, 8.

U oblasti važenja, tip nabrajanja možete koristiti u deklaracijama:

```
enum color bgBoja = plava,         // Definiše dve promenljive
           fgColor = žuta;         // tipa enum color.
void setFgBoja( enum color fgc );  // Deklariše funkciju s parametrom
                                   // tipa enum color.
```

Nabrajanje uvek odgovara standardnom celobrojnom tipu. Zato program na jeziku C može da obavlja uobičajene aritmetičke operacije s promenljivama nabrajanja. U zavisnosti od definisanih vrednosti konstanti nabrajanja, prevodilac će odabrati odgovarajući celobrojni tip. U prethodnom primeru, sve vrednosti nabrajanja `enum color` mogu se predstaviti tipom `char`.

Različite konstante tipa nabrajanja mogu imati iste vrednosti:

```
enum { OFF, ON, STOP = 0, GO = 1, CLOSED = 0, OPEN = 1 };
```

Prethodni primer pokazuje da identifikaciona oznaka nije obavezna u definiciji tipa nabrajanja. Izostavljanje identifikatora ima smisla jedino ako konstante definišete, ali ih ne deklarišete. Konstante je obično bolje definisati na ovaj način, umesto pomoću duže liste direktiva `#define`, pošto tip nabrajanja prosleđuje prevodiocu imena konstanti i njihove numeričke vrednosti. Ta imena su, na primer, naročito pogodna prilikom prikazivanja poruka programa za otklanjanje grešaka.

# Tip void

Specifikator tipa `void` označava da nema dostupnih vrednosti tog tipa. Zato ne možete deklarisati promenljive ili konstante tog tipa. U narednim odeljcima opisujemo u kojim prilikama možete koristiti tip `void`.

## Tip void u deklaracijama funkcija

Funkcija bez povratne vrednosti ima tip `void`. Na primer, standardnu funkciju `perror( )` deklariše prototip:

```
void perror( const char * );
```

Rezervisana reč `void` u listi parametara prototipa funkcije ukazuje na to da funkcija nema parametre:

```
FILE *tmpfile( void );
```

Zato prevodilac prikazuje poruku o grešci ako pokušate da funkciju pozovete na sledeći način: `tmpfile("name.tmp")`. Da je funkcija deklarisana bez specifikatora `void` u listi parametara, prevodilac ne bi imao informaciju o parametrima funkcije i ne bi mogao odrediti je li pravilno pozvana.

## Izrazi tipa void

Izraz tipa `void` nema vrednost. Na primer, poziv funkcije bez povratne vrednosti predstavlja izraz tipa `void`:

```
char filename[] = "memo.txt";
if ( fopen( filename, "r" ) == NULL )
  perror( filename );            // Izraz tipa void.
```

U operaciji konverzije `(void)`*`izraz`* eksplicitno se odbacuje vrednost izraza, kao što je povratna vrednost funkcije:

```
(void)printf("Šta će mi povratna vrednost ove funkcije!\n");
```

## Pokazivači na tip void

Pokazivač vrednosti tipa `void *` predstavlja adresu objekta, ali ne i njegov tip. Takvi pokazivači koji prividno nemaju tip, uglavnom se koriste za deklarisanje funkcija koje mogu da rade s pokazivačima različitih tipova kao argumentima ili koje vraćaju „višenamenski“ pokazivač. Jednostavan primer su standardne funkcije za upravljanje memorijom:

```
void *malloc( size_t size );
void *realloc( void *ptr, size_t size );
void free( void *ptr );
```

Iz primera 2-3 vidi se da vrednost pokazivača tipa `void` možete dodeliti drugom pokazivaču na neki objektni tip, i obrnuto, bez eksplicitno zadate konverzije tipa.

*Primer 2-3. Korišćenje tipa void*

```
// usingvoid.c: Prikazuje kako se koristi tip void
// ------------------------------------------------------
#include <stdio.h>
#include <time.h>
#include <stdlib.h>  // Uključuje naredne prototipove funkcija:
                     // void srand( unsigned int seed );
                     // int rand( void );
                     // void *malloc( size_t size );
                     // void free( void *ptr );
                     // void exit( int status );

enum { ARR_LEN = 100 };

int main( )
{
  int i,                                // Zauzima prostor u memoriji.
      *pNumbers = malloc(ARR_LEN * sizeof(int));

  if ( pNumbers == NULL )
  {
    fprintf(stderr, "Nedovoljno memorije.\n");
    exit(1);
  }

  srand( (unsigned)time(NULL) );        // Inicijalizuje generator
                                        // slučajnih brojeva.

  for ( i=0; i < ARR_LEN; ++i )
    pNumbers[i] = rand( ) % 10000;      // Smešta pojedine slučajne brojeve.

  printf("\n%d slučajni brojevi između 0 i 9999:\n", ARR_LEN );
  for ( i=0; i < ARR_LEN; ++i )         // Izlazna petlja:
  {
    printf("%6d", pNumbers[i]);         // Ispisuje jedan broj po iteraciji
                                        // petlje
    if ( i % 10 == 9 ) putchar('\n');   // i prelazi u novi red posle
                                        // svakih 10 brojeva.
  }
  free( pNumbers );                     // Oslobađa prostor u memoriji.
  return 0;
}
```