TIPOVI VARIJABLI, KONSTANTI I FUNKCIJA

Uz pojam varijable uvijek su vezani pojmovi: tip i ime.

Ime predstavlja oznaku adrese varijable u memoriji, pa se zove i referenca varijable, ili referenca memorijskog objekta.

Tip označava skup vijednosti, način kodiranja i operacije koje se mogu vršiti s vrijednostima tog tipa, te veličinu zauzeća memorije.

Oznaka tipa u C++ jeziku	interval vrijednosti	zauzeće memorije
[signed] char unsigned char	-127 .. 128 0 .. 255	1 bajt 1 bajt
[signed] int [signed] short [signed] long	-2147483648.. 2147483647 -32768 .. 32767 -2147483648.. 2147483647	4 bajta 2 bajta 4 bajta
unsigned [int] unsigned short unsigned long	0 .. 4294967295 0 .. 65535 0 .. 4294967295	4 bajta 2 bajta 4 bajta
float	min. ± 1.175494351e-38 maks ± 3.402823466e+38	4 bajta
double	min. ± 2.2250738585072014e-308 maks. ± 1.7976931348623158e+308	8 bajta
bool	false..true (0 .. 1)	1

Označavanje standardnih tipova podataka u C++ jeziku

```
// sizeof.cpp - Program koji ispisuje zauzeće memorije
//              za sve proste tipove C++ jezika

#include <iostream>
using namespace std;


int main()
{
cout << "Sizeof(bool)          = "  << sizeof(bool)          << endl;
cout << "Sizeof(unsigned char) = " << sizeof(unsigned char) << endl;
cout << "Sizeof(char)          = " << sizeof(bool)          << endl;
cout << "Sizeof(short)         = " << sizeof(short)         << endl;
cout << "Sizeof(unsigned short)= " << sizeof(unsigned short)<< endl;
cout << "Sizeof(int)           = " << sizeof(int)           << endl;
cout << "Sizeof(unsigned int)  = " << sizeof(unsigned int)  << endl;
    cout << "Sizeof(long)          = " << sizeof(long)          << endl;
    cout << "Sizeof(unsigned long) = " << sizeof(unsigned long) << endl;
    cout << "Sizeof(float)         = " << sizeof(float)         << endl;
    cout << "Sizeof(double)        = " << sizeof(double)        << endl;
    return 0;
}
```

```
/* limits.cpp
 * Program koji ispisuje maksimalnu i minimalnu vrijednost
 * numeričkih tipova C++ jezika
 */

#include <iostream>
#include <limits.h>
#include <float.h>
using namespace std;


int main( void )
{
    cout << "Tip         " << "Interval vrijednosti\n"
    cout << "bool        " << false    << "..."   << true     << endl;
    cout << "char        " << CHAR_MIN << "..."   << CHAR_MAX << endl;
    cout << "short int   " << SHRT_MIN << "..."   << SHRT_MAX << endl;
    cout << "int         " << INT_MIN  << "..."   << INT_MAX  << endl;
    cout << "long int    " << LONG_MIN << "..."   << LONG_MAX << endl;
    cout << "float       " << FLT_MIN  << "..."   << FLT_MAX  << endl;
    cout << "double      " << DBL_MIN  << "..."   << DBL_MAX  << endl;
    cout << "long double " << LDBL_MIN << "..."   << LDBL_MAX << endl;
    return 0 ;
}
```

Zapis konstanti kompatibilan s jezikom C

- **numeričke konstante**
- **znakovne konstante**
- **literalni string**

tip konstante	zapis	značenje
char	'A' 'a' '\035' '\x29' '\n'	znakovna konstanata a znakovna konstanata A znakovna konstanata 35 oktalno znakovna konstanata 29 heksadecimalno kontrolni znak nova linija
int	156 0234 0x9C	decimalna notacija oktalna notacija cjelobrojne konstante heksadecimalna notacija
unsigned	156U 0234U 0x9cU	decimalno oktalno (prefiks U određuje kardinalni broj) heksadecimalno
float	15.6F 1.56e1F	realni broj – jednostruki format određen je primjenom sufiksa F ili f
double	15.6 1.56E1L	konstante su tipa "double" ukoliko se ne koristi prefiks F. Nije nužno pisati sufiks L.

Deklaracija – inicijalizacijom varijable

Inicijalizacija varijable je deklaracija u kojoj se pored tipa određuje i početna vrijednost varijable.

U C++ se varijabla može inicijalizirati i iskazom u kojem se početna vrijednost zapisuje u zagradama () iza imena varijable. Iskaz

```
int a(76);      // novost: tzv konstruktor varijable
```

je ekvivalentan iskazu:

```
int a = 76;
```

Deklaracija simboličkih konstanti (const)

1. Simboličke konstante se deklariraju na isti način kao što se vrši inicijalizacija varijabli, s tom razlikom da deklaracija mora započeti ključnom riječi const. Primjerice,

```
const double PI = 3.141592653589793;
const char tab = '\t';
const zip = 12440;
```

U slučaju da nije specificiran tip kompajler podrazumijeva da je to konstanta tipa int. Simboličku se varijablu može koristiti u svim izrazima u kojima je dozvoljeno koristiti literalnu konstantu. To ujedno znači da kompajler ne smije tretirati simboličku varijablu kao memorijski objekt. Zbog toga se simboličkoj konstanta ne može mijenjati vrijednost, odnosno ona ne smije biti napisana s lijeve strane iskaza pridjele vrijednosti.

2. Može se koristiti makro definicije za imenovanje konstanti

```
#define PI 3.141592653589793
```

3. Može se koristiti enum definicije konstati

```
enum dani_t {Nedjelja, Ponedjeljak, Utorak, Srijeda, Cetvrtak, Petak,
Subota};
```

`dani_t` se tretira kao tip podataka

Unos podataka sa standardnog ulaza - cin

Postoje razni načini unosa podataka: pomoću tipkovnice, pomoću miša ili očitanjem sadržaja datoteke ili mreže. Mi ćemo sada upoznati jednostavni način za dobavu podataka koje korisnik programa unosi pomoću tipkovnice. Koristit ćemo standardni ulazni objekt cin, koji je deklariran u iostream.

```
#include <iostream>
using namespace std;

int main(void)
{
// 1. definiraj varijablu čiju vrijednost će unositi korisnik
   int unos;

// 2. ispiši poruku korisniku da program očekuje unos broja:
   std::cout << "Molim otipkajte jedan cijeli broj >";

// 3. Za unos podataka koristi se standardni ulazni objekt - cin
   std::cin >> unos;

// 4. obavi radnje s tom varijablom ......

// 5. ispiši rezultat obrada
   std::cout "Otkucali ste broj" << unos << std::endl ;
    return 0;
}
```

Kada se pokrene ovaj program, na monitoru se ispisuje poruka:

```
c:> Molim otipkajte jedan cijeli broj >_
```

i program čeka da korisnik otkuca jedan broj. Unos završava kada korisnik pritisne tipku <Enter>.
Primjerice, ako korisnik otipka 12345<Enter>, program će završiti s porukom:

```
Otkucali ste broj 12345.
```

Standardni ulazni objekt cin prepoznaje djelovanje "get-to" operatora >>, kojim se simbolički označava tok podataka od standardnog ulaza ka varijabli unos.
Pomoću cin objekta može se odjednom unijeti vrijednost više varijabli. Primjerice, unos jednog cijelog broja, jednog realnog broja i jednog znaka, može se ostvariti na sljedeći način:

```
int i;
double x;
char c;
....
cin >> i >> x >> c;
```

Pri unosu cjelih i realnih brojeva, podrazumjeva se da unos pojedinog broja završava tzv. "bijelim" znakovima (razmak, tab, nova linija). Svi bijeli znakovi uneseni ispred broja se odbacuju. To nije slučaj kada se unosi znak, jer i bijeli znakovi predstavljaju znak. Unos završava pritiskom tipke <Enter>.

Tipovi i funkcije

Mogu se koristiti sve funkcije iz standardne biblioteke C jezika

```
#include <math.h>

int main()
{
    double x,y;
    .........
    x = sin(5.6);
    y = sin(x)+4.6;
    .....
}
```

Najprije je napisana leksička direktiva da se u proces kompajliranja uključi datoteka `math.h` u kojoj je specificiran prototip **(ili deklaracija) funkcije `sin`. Pregledom te datoteke može se pronaći specifikacija prototipa funkcije `sin` oblika:**

```
double sin(double);
```

Njime je iskazano da argument funkcije mora biti vrijednost tipa `double` i da funkcija u izraze vraća vrijednost tipa `double`.

Općenito, funkcija može imati više argumenata. Oni se navode u zagradama iza imena funkcije i odvajaju zarezom. Tip vrijednosti kojim rezultira izvršenje funkcije uvijek se navodi ispred imena funkcije. Deklaracija prototipa završava znakom točka-zarez.

Korisnički definirane funkcije

Definicija funkcije se sastoji od "zaglavlja" i "tijela" funkcije.

Zaglavlje funkcije je deklaracija u kojoj se redom navodi

1. **oznaka tipa koji funkcija vraća u izraze,**
2. **ime funkcije,**
3. **deklaracija liste parametara (formalnih argumenata) funkcije napisanih unutar zagrada.**

Tijelo ili blok funkcije je složeni iskaz naredbi i deklaracija varijabli, koji definiraju implementaciju. Piše se unutar vitičastih zagrada.

Primjerice, definicija funkcije kojom se računa kvadrat cjelobrojnog argumenta glasi

```
int kvadrat(int y)
{
    return y * y;
}
```

Ključna riječ `return` označava mjesto na kojem se prekida izvršenje funkcije, na način da se prethodno izračuna vrijednost izraza koji je napisan iza riječi `return`. Vrijednost tog izraza je vrijednost koju funkcija vraća u izraz iz kojeg je pozvana.

“void” funkcije

U programskim se jezicima često koristi dva tipa potprograma: funkcije i procedure. Procedura **je potprogram koji vrši neki proces, ali ne vraća nikakovu vrijednost. Pošto se u C-jeziku svi potprogrami nazivaju funkcije, onda se kaže da je procedura funkcija koja vraća ništa (eng. void). Primjerice, drugom poglavlju smo koristili funkciju `void Hello();` koja nam je služila za ispis poruke "Hello World!".**

```
void Hello()
{
   cout << "Hello world";
   cout << endl;
}
```

Pomoću riječi `void` označava se da je tip vrijednosti koji funkcija vraća "ništa", odnosno da je nevažan. Poziv procedure se vrši njezinim imenom. Pošto procedure ne vraćaju nikakovu vrijednost, ne mogu se koristiti u izrazima. U proceduri se ne navodi ključna riječ `return`, iako se može koristiti (bez argumenta) ako se želi prekinuti izvršenje procedure prije izvršenja svih naredbi koje su pozivaju u proceduri.

“void” parametri

Kada funkcija ne koristi nikakove parametre tada se i za oznaku argumenta može koristi tip `void`, primjerice:

```
void Hello(void);
```

što eksplicitno označava da funkcija ne koristi niti jedan argument.

Istoimene funkcije s različitim parametrima (eng. overloaded functions)

Dvije ili više funkcija mogu imati isto ime ako se razlikuju tipovi njihovih parametara, ili ako se razlikuju po broju parametara. Tijela funkcija mogu biti definirana po volji.

```
//Datoteka: overload.cpp
#include <iostream>
using namespace std;

int Podijeli (int a, int b)
{  return (a/b); }

float Podijeli (float a, float b)
{  return (a/b); }

int main ()
{
  int   n=5, m=2;
  float x=5.0, y=2.0;
  cout << Podijeli(n, m) << endl;
  cout << Podijeli(x, y) << endl;
  return 0;
}

Rezultat:
 2
 2.5
```

Ako se definiciju funkcija napiše iza tijela main(), tada prije main() treba izvršiti deklaraciju obje funkcije;

```
int Podijeli (int a, int b);
float Podijeli (float a, float b);

int main ()
{
     int   n=5, m=2;
     float x=5.0, y=2.0;
     cout << Podijeli(n, m) << endl;
     cout << Podijeli(x, y) << endl;
     return 0;
}

int Podijeli(int a, int b)
{ return (a/b);  }

float Podijeli(float a, float b)
{ return (a/b); }
```

Pristup podacima pomoću pokazivača i referenci

Adresni operator &

U jeziku C++, ime varijable ujedno označava i adresu varijable. Koja je to adresa? Brigu o tome vodi kompajler. Adresa se može odrediti pomoću posebnog operatora &, koji se naziva adresni operator. On se koristi kao unarni operator tako da se zapisuje ispred imena varijable.

Operator indirekcije *

Komplementarno adresnom operatora u C jeziku se koristi unarni operator indirekcije koji se označava znakom *. On se zapisuje ispred izraza čija vrijednost predstavlja neku adresu. Značaj operatora indirekcije je u tome da se pomoću njega dobije vrijednost koja je upisana na toj adresi. To znači da

ako je `y` jednako `777` onda je `*(&y)` također jednako `777`

jer operator indirekcije daje vrijednost koja se nalazi na adresi varijable y. Ovo pravilo lako možemo provjeriti tako da u prethodnom programu dopišemo naredbu:

```
int y = 777;
cout << "Vrijednost varijable e:" << y << endl;
cout << "Adresa varijable je" << &y << endl;
cout << "Vrijednost dobijena indirekcijom je:" << *(&y) << endl;
```

Tada bi dobili ispis:

```
Vrijednost varijable je 777
Adresa varijable je 0X0063FDF4
Vrijednost dobijena indirekcijom je 777
```

Pokazivači

Varijable kojima je vrijednost adresa neke druge varijable ili funkcije nazivaju se pokazivači ili pointeri. Deklaracija pokazivača vrši se tako da se između oznake tipa i imena pokazivača obvezno zapisuje znak indirekcije '*', primjerice:

```
int * p;          /* p je pokazivač na objekt tipa int */
unsigned *q;      /* q je pokazivač na objekt tipa unsigned */
```

Ovim deklaracijama definirane su dvije pokazivačke varijable. Njihova vrijednost je neodređena, jer im nije pridjeljena adrese nekog realnog memorijskog objekta.

Pokazivače prije upotrebe treba inicijalizirati, odnosno mora im se pridjeliti vrijednost adrese postojećeg memorijskog objekta. Pri tome, tip pokazivača mora biti jednak tipu memorijskog objekta. To se ostvaruje adresnim operatorom '&', primjerice u programskom insertu

```
int suma;         /* deklaracija varijable suma */
int *p;           /* deklaracija pokazivača na objekt tipa int */
sum = 777;        /* inicijaliziranje vrijednosti varijable suma */
p = &suma;        /* p inicijaliziran na adresu varijable suma */
```

najprije je izvršena deklaracija varijable suma i pokazivača p. Zatim je varijabli suma pridjeljena vrijednost 777, a pokazivač p je inicijaliziran da pokazuje na tu varijablu. Ako bi dalje koristili naredbe:

```
cout << suma;
cout << *p;
```

dobili bi isti ispis, jer se indirekcijom pokazivača dobiva vrijednost na koju on pokazuje, a to je vrijednost varijable suma (*p ⇔*&suma).

“const” osigurači

Deklaracija	ime je..
const *T* ime = ...	konstanta tipa *T*
T *const ime = ...	konstantni pokazivač na tip *T*
const *T* *ime = ...	pokazivač na konstantu tipa *T*
const *T* *const ime = ...	konstantni pokazivač na konstantu tipa *T*

Razmotrimo učinak primjene const osigurača u nekim deklaracijama:

```
int i;                          // varijabla tipa int
int *ip;                        // pokazivač na int
int * const cp = &i;            // konstantni pokazivač na int
const int ci = 7;               // konstanta tipa int
const int *cip;                 // pokazivač na konstantu tipa int
```

Mogu se koristiti sljedeće naredbe pridjele vrijednosti:

```
i = ci;       // pridjeljuje se konstanta varijabli
*cp = ci;     // isto, jer *cp indirektno predstavlja varijablu
cip = &ci;    // pokazivač na konstantu tipa int
```

Sljedeći iskazi nisu ispravni:

```
ci = 8;       // ne smije se mijenjati vrijednost konstante,
*cip = 7;     // ni u slučaju kada je referencirana pokazivačem
cp = &ci;     // ne smije se mijenjati konstantni pokazivač,
ip = cip;     // jer bi se mogla posredno mijenjati vrijednost konst.
```

Reference

Ime varijable je njena referenca. Pri izvršenju programa to ime predstavlja fiksnu adresu na kojoj se nalazi vrijednost varijable, pa referenca ima značaj pokazivačke konstante.
U C++ jeziku dozvoljeno je deklariranje referenci. Pošto referenca mora označavati već postojeću adresu ona se, kao i simboličke konstante, prilikom deklaracije uvijek mora inicijalizirati na vrijednost postojeće reference. To ujedno znači da se deklaracijom reference stvara sinonim postojeće reference. Referenca se deklarira pomoću adresnog operatora & prema pravilu:

deklaracija reference:

oznaka_tipa & ***ime_reference = postojeća_referenca*** ;

Primjerice, deklaracijama

```
int x = 10;
int &refx = x;
```

dobije se referenca refx s kojom se može operirati na isti način, na koji se operira s varijablom x.
Nakon izvršenja naredbi:

```
refx = x + 10;
cout << "Vrijednost od x iznosi" << x;
```

dobije se ispis:

```
Vrijednost od x iznosi 20;
```

što potvrđuje da refx i x predstavljaju referencu iste varijable.

Pokazivači i reference kao argumenti funkcije.

Upotreba deklariranih referenci uvedena je u C++ prvenstveno zbog potrebe jednostavnijeg rada s argumentima funkcija i korisnički definiranim tipovima – klasama.

Standardni je način prijenosa argumenata u funkciju po vrijednosti (call by value). Mehanizam "prijenosa po vrijednosti" osigurava da se iz pozvane funkcije ne može mijenjati varijabli pozivne funkcije. Ponekad će ipak trebati direktno mijenjati vrijednost varijabli pozivne funkcije. Za tu namjenu mogu se kao argumenti funkcije deklarirati pokazivači na varijablu ili reference.

Mehanizam prijenosa parametara pomoću referenci naziva se "prijenos po referenci" (call by reference).

Primjer: funkcija swap **– kojom se treba izvršiti zamjena vrijednosti dvije varijable:**

prijenos parametara pomoću reference

```
// Datoteka: swap-r.cpp
#include <iostream>
using namespace std;

void swap( int &x, int &y)
{
  int t;
  t = x;
  x = y;
  y = t;
}

int main()
{
    int a = 1, b = 2;
    cout << a <<' '<< b << endl;
    swap(a, b);
    cout << a <<' '<< b << endl;
    return 0;
}

Ispis:
1 2
2 1
```

prijenos parametara pomoću pokazivača

```
// Datoteka: swap-p.cpp
#include <iostream.h>
using namespace std;

void swap( int *px, int *py)
{
  int t;
  t   = *px;
  *px = *py;
  *py = t;
}

int main()
{
    int a = 1, b = 2;
    cout << a <<' '<< b << endl;
    swap(&a, &b);
    cout << a <<' '<< b << endl;
    return 0;
}

Ispis:
1 2
2 1
```

U radu s memorijskim objektima koristit ćemo i pokazivače i reference, a ponekad ćemo raspolagati s oba načina pristupa objektu. Primjerice, neka je neki objekt x označen referencom refx i pokazivačem px:

```
int x;
int *px = &x;
int &refx = x;
```

Tada u pristupu objektu vrijedi ekvivalentnost:

```
*px ⇔  refx
 px ⇔ &refx
```

To je ilustrirano u datoteci swap-pr.cpp

```
int main()
{
    int a = 1, b = 2;
    int *pa = &a, *pb = &b;
    cout << a <<' '<< b << endl;
    swap(a , b);       // reference varijabli
    cout << a <<' '<< b << endl;
    swap(*pa , *pb);  // dereferencirani pokazivač kao referenca
    cout << a << ' '<< b << endl;
    return 0;
}

Rezultat je:
1 2
2 1
1 2
```

Definiranje tipa pomoću *typedef*

Kada se ispred deklaracije napiše typedef, primjerice

```
typedef int integer;
```

time se označava da identifikator, u ovom slučaju integer, neće biti deklariran kao varijabla ili funkcija, već da taj identifikator postaje sinonim za tip koji je opisan deklaracijom. U ovom primjeru, identifikator integer postaje sinonom za tip int, pa ga se u kasnije može koristiti u drugim deklaracijama, na isti način kako bi koristili i originalni tip:

```
integer i;
```

Važno je napomenuti da se pomoću typedef deklaracije stvaraju sinonimi tipova; a ne neki novi tipovi. Njihova je upotreba korisna za povećanje apstraktnosti programskog zapisa.
Prema ANSI standardu u C jeziku je definirano nekoliko pomoćnih tipova pomoću typedef kako bi se bolje označilo područje njihove primjene. Primjerice, size_t predstavlja tip `unsigned int`, kojim će se označavati veličina, u bajtima, objekata smještenih u datotakama ili u memoriji. Implementacija je provedena deklaracijom

```
typedef unsigned int size_t;
```

u datoteci `stddef.h`.

Drugi primjeri su FILE, time_t, ptrdiff_t i wchar_t (pogledaj njihovo značenje u opisu standardne C-biblioteke).