Typy; stałe; operatory

Typy zmiennych

Zmienna jest to pewne miejsce w pamięci komputera, któremu można przechowywać różne wartości np znakowe czy liczbowe. Przed użyciem zmiennej w programie należy ją zadeklarować. Trzeba więc podać jej typ (rodzaj wartości jakie mogą być do niej przypisywane), dzięki czemu kompilator wie, ile miejsca potrzeba na jej przechowanie.

typ	wielkość pamięci w bajtach	**opis**
char	1	pojedynczy znak ASCI oraz liczby od -128 do 127
int	4	liczba całkowita z zakresu -2 147 483 648 do 2 147 483 647
short	2	liczba całkowita z zakresu -32 768..32 767
unsigned int	4	liczba całkowita z zakresu 0..4 294 967 295
unsigned short	2	liczba całkowita z zakresu 0..65 535
float	4	liczba zmiennoprzecinkowa z zakresu 1.2E38..3.4E38 (pamiętane 7 cyfr)
double	8	liczba zmiennoprzecinkowa z zakresu 2.2E308..1.8E308 (pamiętane 19 cyfr)

Typ wyliczeniowy - enum

Typ wyliczeniowy przydaje się wtedy, gdy pewną rzecz chcemy reprezentować przy pomocy skończonego określonego zestawu wartości. Przykład zastosowania w programie:

```
1  enum kierunki_swiata {wschod, poludnie, zachod, polnoc} ;
2  kierunki_swiata kurs ;
3  kurs = wschod ;
```

Tablice znakowe

Ciągi znaków reprezentowane są w C++ przez tablicę znaków:

```
1  char imie[15] ;
```

Jak wiadomo, tablicę można zainicjować bez podania jej wielkości: Ciągi znaków reprezentowane są w C++ przez tablicę znaków:

```
1  char imie[] = jan //tablica trzyelementowa
```

Na tablicach znaków można operować funkcjami:

- strcat(,) - łączy dwa łańcuchy
- strlen() - długość łańcucha
- strupr() - zamienia w łańcuchu małe litery na duże
- strcmp(,) - porównanie dwóch łańcuchów

Inicjowanie zmiennych

Zmienną można zainicjować w momencie jej deklarowania. Wartości przypisujemy dzięki znakowi równości (=) (operator podstawienia).

Nazwy zmiennych

Nadając nazwy naszym zmiennym musimy uwzględnić kilka reguł:

- nazwy mogą zawierać cyfry, litery oraz znaki podkreślenia;
- słowa kluczowe języka C nie mogą być używane jako nazwy zmiennych (np. int, float, for, switch, void, case, char, return, if...);
- pierwszym znakiem nazwy musi być litera.

Sposobem na zwiększenie zrozumiałości programu jest stosowanie znaczących nazw zmiennych, które opisują ich rolę.

Przykład użycia różnych typów zmiennych:

```
 1 #include <stdio.h>
 2
 3 main()
 4 {
 5   int i, j, k;
 6   char slash = '/';
 7   short month = 4;
 8   int year = 2001;
 9   long population = 38870000;
10
11   cout << "Data = " << month << slash << year;
12   cout << "W Polsce mieszka " << population;
13   return 0;
14 }
```

Stałe

W C++ istnieją dwa sposoby definiowania stałych: przy pomocy deklaracji *const* i przy pomocy dyrektywy *define*.

```
1 #define PI 3.14
2 #define MAXTAB 100
```

Zwyczajowo (choć niekoniecznie) stałę definiowane przy pomocy #Define psiane są dużymi literami. "Define" oznacza zastapienie ciągu znaków innym.

```
1 const float pi = 3.14
2 const int maxtab = 3.14
```

const powioduje powstanie w pamięci nowego obiektu o etykiecie "nie zmieniać wartości". Wszelkie próby przypisania nowej wartości do stałej uważane są błąd.

Operatory języka C/C++

Operatory należą do jednej z dwu grup:

- operatory jednoargumentowe (unarne) wiązane z jednym operandem
- operatory dwuargumentowe (binarne) wiązane z dwoma operandami.

Operatory arytmetyczne

- operatory: **+**, **-**, *****, **/** nie wymagają szczegółowego wyjaśnienia.
- operator **%** czyli modulo
- operatory jednoargumentowe **+** ,**-** Operator + właściwie nic nie robi, natomiast operator - zmienia wartość danego wyrażenia na przeciwną.
- operatory inkrementacji i dekrementacji **++**, **--** Operacje zwiększania i zmniejszania o jeden.

Operatory dekrementacji i inkrementacji mogą mieć dwie formy:

- przedrostkową (prefix): ++i;
- końcówkową (postfix): i++;

W obu przypadkach wynikiem jest zmiana wartości i, ale wyrażenie ++i zwiększa i przed użyciem jej wartości, natomiast wyrażenie i++ zwiększa zmienną i dopiero po użyciu jej poprzedniej wartości. W kontekście, w którym ważna jest wartość zmiennej i, a nie tylko jej zmiana, wyrażenia ++i oraz i++ są różne np:

```
1 i = 10;
2 x = i++;
3 cout<<" X = "<<x;
4 /* daje w rezultacie X = 10, ale */
5 i = 10;
6 x = ++i;
7 /* cout<<" X = "<<x; */
8 daje w rezultacie X = 11.
```

Operator przypisania

- operator przypisania **=** Powoduje on, że do obiektu stojącego po lewej stronie operatora przypisania zostaje zapisana wartość wyrażenia stojącego po prawej stronie. Jest to operator dwuargumentowy. Z założenia, operandy stojące po obu stronach powinny mieć taki sam typ. Jeśli sąróżnego typu, to - jeśli to tylko możliwe - wykonana zostaje niejawna konwersja (dopasowanie) typów (patrz Przekształcanie typów).

Operatory logiczne

- operatory relacyjne: **>**, **>=**, **<**, **<=** W wyniku działania tych operatorów otrzymujemy odpowiedź: prawda (true - 1) lub fałsz (false - 0). Priorytet tych operatorów jest taki sam.
- operatory przyrównania **==**, **!=** Ich priorytet jest niższy niż priorytet operatorów relacji
- operatory sumy i iloczynu logicznego, **||** - realizujący sumę logiczną (LUB - alternatywa), **&&** - realizujący iloczyn logiczny (I - koniunkcja)

```
 1 #include<iostream.h>
 2 main()
 3 {
 4 int ch;
 5 ch = getchar();
 6 if((ch >='a' && ch <='z') || (ch >='A' && ch <='Z'))
 7 cout<<"wprowadzony znak jest literą";
 8 else
 9 cout<<"Wprowadzony znak jest innym znakiem";
10 return 0;
11 }
```

- operator negacji **!** Jest to operator jednoargumentowy i stoi zawsze po lewej stronie operandu.

Przekształcanie typów

Jeżeli argumentami operatora są obiekty różnych typów, to są one przekształcane do jednego wspólnego typu według kilku reguł. Ogólna zasada mówi że: automatycznie wykonuje się tylko takie przekształcenia, w których argument "ciasniejszy jest zamieniany na obszerniejszy bez utraty informacji: np zamiana liczby całkowitej na zmiennopozycyjną.

Oprócz automatycznego przekształcania typów, w dowolnym wyrażeniu można jawnie wymusić przekształcenie typów za pomocą:

- jednoargumentowego operatora rzutowania. Operator ten może mieć dwie formy:
  - (nazwa_typu) zmienna //forma języka C/C++
  - nazwa_typu(zmienna) //forma C++

Operator ten upewnia niejako komputer, że rzeczywiście chcemy przeprowadzić "niedozwoloną" operacje konwersji.

Przykład:

```c
#include <stdio.h>

main()
{
  int i1 = 4;
  int i2 = 4;
  float f = 2.8;


  // Rzutowanie
  i1 = f*i1;
  printf("\nwynik : %i\n", i1);

  i2 = (int)f* i2;
  printf("wynik : %i\n", i2);
}
```

Operator sizeof()

Często ważne jest znanie rozmiarów typów. o tych celów zaimplementowano w języku C/C++ bardzo wygodny operator: sizeof(), podający rozmiary zmiennych wywołanie: sizeof(nazwa_typu)

Operator przecinek

Ostatnim operatorem języka C/C++ jest przecinek. Jeśli kilka wyrażeń stoi obok siebie i są oddzielone przecinkami, to całość też jest wyrażeniem, a wartością tego wyrażenia jest wartością prawego argumentu. Poszczególne wyrażenia obliczane są od lewej do prawej.

(2+4, a*4, 3<6, 77+2) wart.wyrażenia: 79

Jest on np. stosowany w pętli for do równoległego sterowania dwoma indeksami np:

```c
{
int c,i,j;
for(i = 0, j = strlen(s)-1; i<j;i++,j--)
{
c = s[i];
s[i] = s[j];
s[j] = c;
}
```

Priorytet i łączność operatorów

Priorytety operatorów i ich łączność (m mniejsza liczba tym wyższy priorytet) podaje zestawienie:

Priorytet	Operatror	Łączność
1.	() [] ->	Lewostronna

2.	! ~ ++ -- + - * & sizeof	Prawostronna
3.	* / %	Lewostronna
4.	+ -	Lewostronna
5.	<< >>	Lewostronna
6.	< <= > >=	Lewostronna
7.	== !=	Lewostronna
8.	&	Lewostronna
9.	^	Lewostronna
10.	|	Lewostronna
11.	&&	Lewostronna
12.	| |	Lewostronna
13.	= += -= *= /= %= ^= |= <<= >>=	Prawostronna
14.	,	Lewostronna