Operatory w języku C++, podstawowe wiadomości

Omawianie zagadnienia operatorów w języku C++ rozpoczniemy od analizy poniższego programu (plik *operatory.cpp*)

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

int main()
{
int a=10,b=3;
float fa=10,fb=3;

   cout<<"Program demonstruje dzialanie niektorych operatorow C++"<<endl;
   cout<<"Operatory: + - * / dzialaja standardowo, ale..."<<endl<<endl;

   cout<<"a+b="<<a+b<<" \tfa+fb="<<fa+fb<<endl;
   cout<<"a-b="<<a-b<<" \tfa-fb="<<fa-fb<<endl;
   cout<<"a*b="<<a*b<<" \tfa*fb="<<fa*fb<<endl;
   cout<<"a/b="<<a/b<<" \tfa/fb="<<fa/fb<<endl;
   cout<<endl;

   cout<<"Operator \"reszta z dzielenia\" - modulo - %"<<endl;
   cout<<"a%b="<<a%b<<endl<<endl;

   cout<<"Operator inkrementacji ++ oraz dekrementacji --:"<<endl<<endl;
   cout<<"Wartosci zmiennych przed wykonaniem instrukcji: a="<<a<<" b="<<b<<endl;
   cout<<"Na a i b podzialamy operatorem ++ w dwoch \"wersjach\" :"<<endl;
   cout<<"\ta++ daje wartosc "<<a++<<"\n\t++b daje wartosc "<<++b<<endl;
   cout<<"Wartosci zmiennych po wykonaniu instrukcji: a="<<a<<" b="<<b<<endl<<endl;

   cout<<"Operator sizeof podaje, ile (bajtow) pamieci potrzeba "
         "na zmienna danego typu"<<endl;
   cout<<"sizeof(int) ma wartosc "<<sizeof(int)<<endl;
   cout<<"sizeof(float) ma wartosc "<<sizeof(float)<<endl;
   cout<<"sizeof(double) ma wartosc "<<sizeof(double)<<endl;
}
```

Po uruchomieniu powyższego programu na konsoli pojawia się:

```
Program demonstruje dzialanie niektorych operatorow C++
Operatory: + - * / dzialaja standardowo, ale...

a+b=13  fa+fb=13
a-b=7   fa-fb=7
a*b=30  fa*fb=30
a/b=3   fa/fb=3.33333

Operator "reszta z dzielenia" - modulo - %
a%b=1

Operator inkrementacji ++ oraz dekrementacji --:

Wartosci zmiennych przed wykonaniem instrukcji: a=10 b=3
Na a i b podzialamy operatorem ++ w dwoch "wersjach" :
        a++ daje wartosc 10
        ++b daje wartosc 4
Wartosci zmiennych po wykonaniu instrukcji: a=11 b=4

Operator sizeof podaje, ile (bajtow) pamieci potrzeba na zmienna danego typu
sizeof(int) ma wartosc 4
sizeof(float) ma wartosc 4
sizeof(double) ma wartosc 8
```

W programie definiujemy dwie pary zmiennych: *a* i *b*, oraz *fa* i *fb*, odpowiednio typów: *int* oraz *float*. W języku *C++* operatory arytmetyczne: sumy, różnicy, iloczynu i ilorazu mają oznaczenia: + - * /. Istotny jest fakt, że działanie operatorów jest uzależnione od typu jego argumentów. Widać to na przykładzie wyników operatora dzielenia. W obu przypadkach dzielimy liczbę 10 przez 3, lecz gdy argumenty dzielenia są liczbami całkowitymi, to rezultat jest również liczbą całkowitą. Analogicznie, dzielenie przez siebie argumentów typu *float* daje wynik, będący również liczbą typu *float*. Jest to jeden z powodów, dla których konieczna jest definicja zmiennych: wiele operacji kompilator musi zrealizować na różne sposoby, w zależności od tego, na jakich argumentach są realizowane. Kolejny powód, to konieczność rezerwacji pamięci na zmienne, co wymaga znajomości typu zmiennej, aby wyznaczyć konieczną ilość pamięci. Ta wiedza jest niezbędna kompilatorowi, jednakże często jest również potrzebna w trakcie działania programu - tak jest np. w sytuacji zapisu danych na dysk. Do wyznaczenia rozmiaru w bajtach (ściślej: w porównaniu z wielkością zmiennej typu *char*, mającej zazwyczaj rozmiar jednego bajta) służy operator *sizeof*, postaci *sizeof wyrażenie* lub *sizeof(nazwa typu)*. W zależności od użytej formy kompilator ocenia, ile pamięci potrzeba na zapisanie wartości, jaką ma podane wyrażenie, lub podaje ile bajtów pamięci zajmuje zmienna danego typu danych (zawartego w języku lub zdefiniowanego przez użytkownika). W programie była użyta druga forma tego operatora (jest ona częściej używana w praktyce) dla typów wbudowanych języka: *int*, *float* i *double*.

Przejdźmy do ciekawszego elementu języka, jakim są operatory *inkrementacji* i *dekrementacji*. Oba mają tę ciekawą cechę, że mogą występować zarówno po (*a++* tzw. *Postinkrementacja*), jak i przed (*++b* tzw. *preinkrementacja*) obiektem, na który działają. Obie te postaci zwiększają o jeden wartość obiektu, na który działają, zaś różnica polega na tym, że operator *preinkrementacji* zwraca jako wynik nową wartość (zwiększoną), zaś operator *postinkrementacji* zwraca poprzednią wartość zmiennej. Efektem działania obu postaci operatora jest zwiększenie wartości obiektu o jeden, postaci te różnią się wartością zwracaną, co ma znaczenie, jeśli stanowią część bardziej złożonego wyrażenia. Istnieje analogiczna para operatorów *dekrementacji*, --.

Pozostał jeszcze do omówienia operator *modulo*, czyli reszty z dzielenia, oznaczany %. Działa on tylko na zmiennych typu całkowitego. Wartości zwracane przez operator *modulo* mieszczą się w zakresie *0 ... dzielnik - 1* (z dokładnością do znaku), gdzie *dzielnik* to drugi argument operatora (czyli występujący po nim w wyrażeniu). Do czego wykorzystywany jest operator *modulo*? Poza ściśle numerycznymi zastosowaniami, można za jego pomocą wykonywać pewne czynności w pętli co zadaną ilość kroków np. aby pewną czynność wykonać w pętli co pięć jej kroków, wystarczy sprawdzać, czy reszta z dzielenia wartości zmiennej indeksującej pętlę przez 5 wynosi zero np. w następujący sposób:

```
if(i%5==0) cout<<endl;
```

Instrukcja *cout<<endl;* wykona się w krokach pętli, dla których i=5, 10, 15...

Przejdźmy do analizy kolejnego programu (plik *logika.cpp*)

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

// Funkcja ma zwracac wieksza z dwoch podanych jej wartosci
int max(int a, int b);
// Funkcja "komentuje" wartosc wyrazenia logicznego
void wypisz(bool);

int main()
{
int a=10, b=3, c=7;

   cout<<"Program demonstruje operatory logiczne i wyrazenie warunkowe"<<endl;
   cout<<"Przyjmujemy a=10 b=3 c=7"<<endl<<endl;
```

```
   cout<<"a<b - "; wypisz(a<b); cout<<endl;
   cout<<"a>b - "; wypisz(a>b); cout<<endl;
   cout<<"a<=b - "; wypisz(a<=b); cout<<endl;
   cout<<"a>=b - "; wypisz(a>=b); cout<<endl;
   cout<<endl;

   cout<<"a==10 - "; wypisz(a==10); cout<<" (czy rowne)"<<endl;
   cout<<"a!=10 - "; wypisz(a!=10); cout<<" (czy rozne od)"<<endl;
   cout<<endl;

   cout<<"Warunki zlozone konstruujemy poprzez && || ! :"<<endl<<endl;
   cout<<"a<b && c==7 - "; wypisz(a<b && c==7);
   cout<<" (falsz I prawda daje falsz)" <<endl;
   cout<<"a<b || c==7 - "; wypisz(a<b || c==7);
   cout<<" (falsz LUB prawda daje prawda)"<<endl;
   cout<<"!(a<b || c==7) - "; wypisz( !(a<b||c==7));
   cout<<" (ZANEGOWANA prawda daje falsz - patrz poprzedni)"<<endl<<endl;

   cout<<"Na zakonczenie wyrazenie warunkowe ? : "<<endl;
   cout<<"a<7 ? 2 : 5 ma wartosc "<< ( a<7 ? 2 : 5 ) <<endl;
   cout<<"a>b ? 2 : 5 ma wartosc "<< ( a>b ? 2 : 5 ) <<endl;
}

int max(int a, int b)
{
   // jesli a jest wieksze od b, to zwroci a, jesli zas nie - zwroci b
   return a>b ? a : b;
}

void wypisz(bool wartosc)
{
   if(wartosc) cout<<"PRAWDA";
   else cout<<"FALSZ";
}
```

Po uruchomieniu programu na konsoli pojawi się:

```
Program demonstruje operatory logiczne i wyrazenie warunkowe
Przyjmujemy a=10 b=3 c=7

a<b - FALSZ
a>b - PRAWDA
a<=b - FALSZ
a>=b - PRAWDA

a==10 - PRAWDA (czy rowne)
a!=10 - FALSZ (czy rozne od)

Warunki zlozone konstruujemy poprzez && || ! :

a<b && c==7 - FALSZ (falsz I prawda daje falsz)
a<b || c==7 - PRAWDA (falsz LUB prawda daje prawda)
!(a<b || c==7) - FALSZ (ZANEGOWANA prawda daje falsz - patrz poprzedni)

Na zakonczenie wyrazenie warunkowe ? :
a<7 ? 2 : 5 ma wartosc 5
a>b ? 2 : 5 ma wartosc 2
```

Operatory porównań w języku C++ to:

- < mniejsze od ...
- > większe od ...
- <= mniejsze lub równe od ...

- >= większe lub równe od ...
- == równe ... (Uwaga! To są dwa znaki =)
- *!=* różne od ...

, dostarczające wartości logicznej: prawda albo fałsz.

Do konstruowania złożonych warunków logicznych stosuje się operatory:

- *&&* - logiczne *AND* (iloczyn logiczny) - wartością jest prawda, jeśli wartości obu argumentów są prawdą.
- || - logiczne *OR* (suma logiczna) – wartością jest prawda, jeśli wartością co najmniej jednego z dwóch argumentów jest prawda.
- *!* - logiczne *NOT* (negacja) – negacja jest operatorem jednoargumentowym, wartością jest prawda, jeśli wartością argumentu jest fałsz.