Operatory arytmetyczne i relacyjne, inkrementacja i dekrementacja w C++

Zadanie 1.
Stworzyć plik `pierwszy.cpp` i wpisać w nim następujący kod:

```
#include<iostream>
using namespace std;
int main()
{
  int liczba1;
  int liczba2;
  int suma;
  cout << "Podaj pierwsza liczbe: ";
  cin >> liczba1;
  cout << "Podaj druga liczbe: ";
  cin >> liczba2;
  suma = liczba1 + liczba2;
  cout << "Suma podanych liczb wynosi: " << suma << endl;
  system("pause");
  return 0;
}
```

następnie skompilować plik oraz uruchomić program.

Powyższy program prosi najpierw o dwie liczby całkowite i zwraca później wynik ich dodawania. Bardzo ważne są dwa wiersze powyższego kodu:

```
int suma;
suma = liczba1 + liczba2;
```

Pierwszy wiersz jest jak już wiemy deklaracją zmiennej `suma`. W drugim wierszu następuje przypisanie do niej sumy dwóch liczb wcześniej wczytanych.

Rodzaje operatorów arytmetycznych

Znak +, którego użyliśmy w napisanym przed chwilą programie, jest jednym z kilkunastu operatorów języka C++. **Operator** to jeden lub kilka znaków (zazwyczaj nie będących literami), które mają specjalne znaczenie w języku programowania.

Operatory dzielimy na kilka grup; jedną z nich są właśnie operatory arytmetyczne, które służą do wykonywania prostych działań na liczbach. Odpowiadają one podstawowym operacjom matematycznym. W C++ obowiązują takie same zasady jak w matematyce odnośnie kolejności wykonywania działań. Jeśli chcemy zmienić kolejność, używamy nawiasów okrągłych.

Operator	opis
+	dodawanie
-	odejmowanie
*	mnożenie
/	dzielenie
%	reszta z dzielenia

Pierwsze trzy pozycje są intuicyjne. Przyjrzymy się za to bliżej operatorom związanym z dzieleniem. Operator `/` działa na dwa sposoby w zależności od tego, jakiego typu liczby dzielimy. Rozróżnia on bowiem dzielenie całkowite, kiedy interesuje nas jedynie wynik bez części po przecinku, oraz rzeczywiste, gdy życzymy sobie uzyskać dokładny iloraz. Rzecz jasna, w takich przypadkach jak `25/5`, `33/3` czy `221/13` wynik będzie zawsze liczbą całkowitą. Gdy jednak mamy do czynienia z liczbami niepodzielnymi przez siebie, sytuacja nie wygląda już tak prosto.

Kiedy zatem mamy do czynienia z którymś z typów dzielenia? Zasada jest bardzo prosta – jeśli obie dzielone liczby są całkowite, wynik również będzie liczbą całkowitą; jeżeli natomiast choć jedna jest rzeczywista, wtedy otrzymamy iloraz wraz z częścią ułamkową.
Wynika stąd zatem, że takie przykładowe działanie

```
float wynik = 11.5/2.5;
```

da nam prawidłowy wynik `4.6`. Co jednak zrobić, gdy dzielimy dwie niepodzielne liczby całkowite i chcemy uzyskać dokładny rezultat?… Musimy po prostu obie liczby zapisać jako rzeczywiste, a więc wraz z częścią ułamkową – choćby była równa zeru, przykładowo:

```
float Wynik = 8.0/5.0;
```

Uzyskamy w ten sposób prawidłowy wynik `1.6`.

Operator `%` związany jest ściśle z dzieleniem całkowitym, mianowicie oblicza nam resztę z dzielenia jednej liczby przez drugą. Przykładowo

```
int reszta = 7%3;
```

da wynik 1.

Rodzaje operatorów relacyjnych

Kolejną grupą operatorów są operatory relacyjne. W jej skład wchodzą operatory logiczne i operatory porównania. Pozwalają one porównywać ze sobą liczby oraz tworzyć złożone warunki, których używamy m.in. w instrukcjach warunkowych. Poniższa tabelka przedstawia najczęściej używane operatory logiczne i operatory porównania:

Operator	Opis
<	mniejsze niż
>	większe niż
<=	mniejsze lub równe
>=	większe lub równe
==	równe
!=	różne
&&	iloczyn logiczny (and)
||	suma logiczna (or)

!	negacja (not)

Zadanie 2.
Napisać program (w pliku o nazwie `operatory.cpp`), który zawierałby operacje arytmetyczne wykorzystujące wszystkie powyższe operatory arytmetyczne (analogicznie jak zadanie 1).

Zadanie 3.
Napisać program (w pliku o nazwie `waluta.cpp`), który:

1. pobiera od użytkownika: kwote_w_dolarach;
2. deklaruje zmienne: kwota_w_złotowkach, kurs_dolara (zadeklarować tę zmienną jako stałą równą 2.84) oraz wypisuje informację: „DD dolarow to ZZ zlotych".

Oczywiście przyjmujemy założenie, że kwota_w_złotowkach = kwota_w_dolarach * kurs_dolara.

Zadanie 4.
Napisać program, który przelicza wprowadzoną przez użytkownika liczbę sekund na minuty tak, aby otrzymać komunikat postaci: " SS sekund to MM minut ".

Zadanie 5.
Napisać program (w pliku o nazwie `pensja.cpp`), który:

3. pobierałby od użytkownika pensje_brutto;
4. deklaruje zmiennei: pensja_netto, podatek (zadeklarować tę zmienną jako stałą równą 19).

Przyjmujemy założenie, że pensja_netto = pensja_brutto – pensja_brutto * podatek / 100.

Operatory inkrementacji i dekrementacji

Zanim omówimy operatory wymienione w tytule spróbujmy rozszyfrować zapis

```
int a=5;
a=a+2;
```

Pierwszy wiersz to jak wiemy deklaracja zmiennej całkowitej o nazwie `a` i przypisanie do niej wartości 5. Drugi wiersz natomiast odczytujemy następująco: „dodaj do zmiennej `a` (która w tym momencie wynosi 5) liczbę 2 i wynik przypisz do zmiennej `a`". Oznacza to, że obecnie zmienna `a` ma wartość 7.
Twórcy języka C++ dodali kilka mechanizmów skracających tego typu zapis. Z jednym z nich, bardzo często wykorzystywanym, zapoznamy się za chwilę.
Otóż instrukcje w rodzaju

```
zmienna = zmienna + zmienna1;
x = x * 10;
i = i + 1;
j = j - 1;
```

mogą być napisane nieco krócej. Zauważmy, że we wszystkich przedstawionych przykładach po obu stronach znaku = znajdują się te same zmienne. Instrukcje powyższe nie są więc przypisywaniem zmiennej nowej wartości, ale modyfikacją już przechowywanej liczby.
Korzystając z tego faktu, pierwsze dwie linijki możemy zapisać jako

```
zmienna += zmienna1;
x *= 10;
```

Jak widzimy, operator `+` przeszedł w `+=`, zaś `*` w `*=`. Podobna „sztuczka" możliwa jest także dla trzech pozostałych znaków działań. Sposób ten nie tylko czyni kod krótszym, ale także przyspiesza jego wykonywanie. Jeżeli chodzi o następne wiersze, to oczywiście dadzą się one zapisać w postaci

```
i += 1;
j -= 1;
```

Można je jednak skrócić (i przyspieszyć) nawet bardziej. Dodawanie i odejmowanie jedynki są bowiem na tyle częstymi czynnościami, że dorobiły się własnych operatorów `++` oraz `--` (tzw. inkrementacji i dekrementacji), których używamy tak:

```
i++;
j--;
```

lub tak:

```
++i;
--j;
```

Wykorzystamy je przy okazji omawiania pętli `for`.