07. Operatory, konwersja typów

1 Operatory

W języku C++ istnieje możliwość definiowania własnych operatorów (w szczególności operatorów matematycznych):

```
class Vector2D {
  public:
    Vector2D operator-(Vector2D other) {
      return Vector2D(x - other.x, y - other.y);
    }
  //...
```

Powyższa definicja operatora udostępni operacje odejmowania na obiektach klasy:

```
Vector2D v1(1.0, 1.0), v2(2.0, 0.5);
Vector2D v3 = v1 - v2;
```

Zapoznaj się z kodem zawartym w pliku *pob07-vector3d.cpp*.

Zadanie 01 Dodaj do klasy `Vector3D` następujące operatory:

- binarny + — pozwalający na dodawanie dwóch wektorów, który zastąpi metodę `add(Vector3D)`,
- binarny * — pozwalający mnożyć wektor przez skalar typu `float`,
- unarny - — zwracający wektor przeciwny,
- binarny * — mnożący dwa wektory i zwracający iloczyn skalarny.

Zadanie 02 Operator przypisania = jest generowany automatycznie i w większości przypadków jest on wystarczający. Kiedy nie jest i dlaczego? Zwróć uwagę na podobieństwo tej sytuacji z tą, która powoduje, że również domyślny konstruktor kopiujący nie jest wystarczający.

Zadanie 03 Do klasy `Vector3D` dodaj operator « pozwalający na wypisanie wektora przez strumień `std::cout` (klasa `std::ostream`). Powinien on być zdefiniowany poza klasą.

2 Konwersja typów

Konwersje typów definiuje się jako operatory bezargumentowe, których nazwa jest typem na który chcemy konwertować:

```
class Vector2D {
  public:
    operator float() { return x + y; }
  // ...
};
```

Przykładowa konwersja:

```
Vector2D vec(1.0, 2.0);
std::cout << (float)vec;
```

Zadanie 04 Do klasy `Vector3D` dodaj konwersję do typu `float` — zwrócona powinna zostać długość wektora ($\sqrt{x^2 + y^2 + z^2}$).

Napisz fragment kodu, który będzie demonstrował konwersje obiektu typu `Vector3D` do liczby zmiennopozycyjnej. Dlaczego kompilator zwraca ostrzeżenie (ew. błąd)?

Zadanie 05 Dodaj klasę `Vector2D`, która w przeciwieństwie do klasy `Vector3D` będzie posiadała tylko dwa atrybuty: `x` i `y`. Klasa ta powinna mieć również konstruktor dwuargumentowy oraz metodę `print()`;

Do klasy `Vector3D` dodaj konwersję do klasy `Vector2D` poprzez pominięcie składowej `z`.

Zadanie domowe 06 (1 pkt) Napisz klasę `Stack`, która będzie reprezentowała stos przechowujący liczby typu `float` (można wykorzystać fragmenty kodu z poprzednich zajęć w wersji tablicowej). Powinna istnieć możliwość tworzenia stosu na podstawie przekazanej do konstruktora tablicy (i jej rozmiaru), np.:

```
float tab[] = {1.0, 2.0, 3.0}
Stack stack(tab, 3);
```

Do stosu dodaj następujące operatory:

- `< <` do dodawania elementów do stosu,
- `> >` do ściągania elementów ze stosu,
- `[ ]` zwracający n-ty elementy w stosie licząc od góry (obsłuż sytuację, gdy n jest większe od liczby wszystkich elementów lub ma wartość ujemną).

Poprawnie zaimplementowane operatory « oraz » powinny umożliwiać działanie na stosie w następujący sposób:

```
stack << 4.0 << 5.0;  // dodanie dwoch elementow

float x, y, z;
stack >> x >> y >> z; // sciagniecie elementow 5.0, 4.0 i 3.0

float u = stack[1];   // dostep do elementu 1.0
```

Do klasy dopisz również odpowiedni operator przypisania, który przydzieli nową pamięć dla elementów na stosie. Napisz fragment kodu demonstrujący jego użycie.

Zadanie dodatkowe 07 (1 pkt) Do klasy `Vector2D` dodaj również konwersję w drugą stronę, czyli do typu `Vector3D` — zainicjalizuj współrzędną z wartością 0.0. W funkcji `main()` przetestuj działanie konwersji między obiema klasami w obie strony.