Podstawy programowania: Laboratorium nr 8
Przestrzenie nazw. Przeciążanie operatorów.

2017-2018

mgr inż. Przemysław Walkowiak
dr inż. Michał Ciesielczyk

Instrukcja

W czasie pisania programu pamiętaj o:

1. dbaniu o czytelność kodu (odpowiednie formatowanie kodu, nazewnictwo zmiennych adekwatne do ich znaczenia, komentarze),
2. dbaniu o czytelność interfejsu z użytkownikiem (w sposób jawny pytaj użytkownika jakie dane ma podać oraz opisuj wyniki, które zwracasz),
3. przed fragmentem implementującym poszczególne zadania umieść komentarz: `/*Zadanie X */` oraz wypisz na ekranie analogiczny komunikat (X jest numerem zadania): `std::cout << "Zadanie X"<< std::endl;`,
4. umieszczeniu wszystkich rozwiązań w jednym pliku, chyba, że w poleceniu napisano inaczej.
5. w zadaniach wymagających udzielenia komentarza bądź odpowiedzi, należy umieścić go w kodzie programu (np. w postaci komentarza albo wydrukować na ekranie).

Wprowadzenie

Listing 1: Przykładowy plik nagłówkowy point.hpp.

```
#ifndef POINT_HPP
#define POINT_HPP
#include <iostream>

struct Point
{
    double x;
    double y;

    Point();
    Point(double x, double y);
};

// overloading plus (+) operator
Point operator+(const Point& a, const Point& b);

// overloading left shift («) operator
std::ostream& operator << (std::ostream& stream, const Point& p);

#endif // POINT_HPP
```

Listing 2: Przykładowy plik źródłowy point.cpp z implementacją dla point.hpp.

```
#include "point.hpp"

Point::Point() : x(0.0), y(0.0) {}

Point::Point(double x, double y) : x(x), y(y) {}

Point operator+(const Point& a, const Point& b) {
    return Point(a.x + b.x , a.y + b.y );
}

std::ostream& operator << (std::ostream& stream, const Point& p) {
    return stream << "(" << p.x << ", " <<  p.y << ")";
}
```

Przeciążanie operatorów wykorzystywane jest w celu minimalizacji redundancji kodu oraz zwiększenia jego czytelności. Załóżmy, że program ma na celu dodanie do siebie trzech punktów.

```
    Point p1, p2, p3, result;
```

Wykorzystanie typowej implementacji (np. jak ta przedstawiona na poprzednim laboratorium) może wyglądać następująco:

```
    result = add(p1, add(p2, p3));
```

Z wykorzystaniem przeciążonego operatora dodawania (linia 15 na listingu 1) kod można zmodyfikować jak poniżej:

```
    result = p1 + p2 + p3;
```

Przeciążanie operator wstawienia ((linia 18 na listingu 1) umożliwia w intuicyjny sposób odwoływanie się na przykład do strumienia standardowego `std::cout`. Jeżeli zadanie polega na wyświetleniu zawartość struktury można wykorzystać klasyczną metodę:

```
    Point p1;
    cout << p1.x << " " << p1.y << std::endl;
```

lub operator <<:

```
    Point p1;
    cout << p1 << std::endl;
```

Zadania

Zadanie 1

Wykorzystaj definicję liczb zespolonych z poprzednich laboratoriów (struktura `complex`). Umieść ją w oddzielnej przestrzeni nazw `math`. Po wprowadzeniu tej modyfikacji, w głównym pliku źródłowym utwórz dwie przykładowe zmienne typu `complex`, a następnie dodaj je do siebie.

Dodatkowe informacje:

- Przestrzenie nazw: `http://en.cppreference.com/w/cpp/language/namespace`

Zadanie 2

Korzystając z implementacji obsługi liczb zespolonych (struktura `complex`) z poprzednich laboratoriów oraz wykorzystując przeciążanie operatorów zaimplementuj operacje arytmetyczne działające na liczbach zespolonych (dodawanie, odejmowanie, mnożenie oraz przyrównanie). Przetestuj swoją implementację sprawdzając wszystkie operacje na przykładowych danych.

Zadanie 3

Zaimplementuj dla struktury liczby zespolonej `complex` operator przekierowania strumienia << oraz >>. Wykorzystaj je odpowiednio do wczytania i wyświetlenia liczby zespolonej. Przetestują swoją implementację wczytując oraz wyświetlając przykładową liczbę zespoloną.

Zadanie 4

Zaprojektuj struktury przechowujące informacje o:

a) kołach - średnica, szerokość, ogumienie zimowe/letnie,
b) silniku - pojemność, typ paliwa (benzyna/diesel),

oraz strukturę `Samochod` składającą się z pól powyższych typów. Dla wszystkich trzech struktur zaimplementuj funkcję do wczytywania zawartości obiektów oraz przeciąż operatory pomagające w wyświetleniu tych obiektów. Ostatecznie wyświetlanie ma być realizowane w następujący sposób:

```
Samochod samochod;
cout << samochod;
```

Zadanie 5

Zaimplementuj plikową bazę danych studentów. Program ma realizować poniższą funkcjonalność:

a) dodawanie nowego studenta na listę,
b) wyświetlanie na ekranie informacji o studencie na podstawie jego numeru indeksu,
c) zapisywanie listy studentów do pliku,
d) odczytywanie z pliku listy studentów,
e) usuwanie z listy studentów o podanym numerze indeksu.

Przeciąż odpowiednio operatory przekierowania strumienia << oraz >> do wczytania i wyświetlenia danych o studencie.

Zapewnij wybór czynności poprzez odpowiednie menu. W przypadku, gdy nie ma studenta o podanym numerze indeksu, wyświetl stosowną informację.

Wskazówka Do przechowywania listy studentów w pamięci możesz wykorzystać np. `std::vector`.

Na następne zajęcia

- RAII – `http://en.cppreference.com/w/cpp/language/raii`
- Smart pointers – `https://msdn.microsoft.com/en-US/library/hh279674.aspx`