Języki i paradygmaty programowania: Laboratorium nr 1

Podstawowe paradygmaty programowania obiektowego - wprowadzenie.

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. każde zadanie umieść w oddzielnej funkcji (w niej dopiero należy odwoływać się do zaimplementowanych funkcji i klas),
5. zaimplementuj menu wyboru zadania, a następnie wykorzystując pętle **do–while** oraz konstrukcję **switch** wykonaj odpowiedni fragment kodu,
6. w zadaniach wymagających udzielenia komentarza bądź odpowiedzi, należy umieścić go w kodzie programu (np. w postaci komentarza albo wydrukować na ekranie),
7. w zadaniach polegających na zaprojektowaniu klasy należy utworzyć jej instancję i wykorzystać zaimplementowaną funkcjonalność.

Wprowadzenie

```
struct Point
{
    double x;
    double y;

    //constructor
     Point() {
        x = 0.0;
        y = 0.0;
    }
    // constructor
    Point(double x, double y) {
        this->x = x;
        this->y = y;
    }
    // member function
    Point add( Point &other ) {
        return Point(x + other.x , y + other.y );
```

```
    }

    double getX() const {
        return x;
    }

    double getY() const {
        return y;
    }

    // destructor
    ~Point() {

    }
}
```

Konstruktor jest to metoda wewnątrz struktury wywoływana w momencie tworzenia instancji tej struktury. Konstruktor ma zawsze nazwę taką samą jak struktura i nie zwraca żadnej wartości. Wykorzystuje się ją do inicjalizacji wartości pól obiektu. Konstruktor domyślny (linia 7) nie posiada żadnych argumentów i jest wywoływany w przypadkach jak poniżej:

```
Point p1; // konstruktor domyślny nadaje polom x i y wartości 0
std::unique_ptr<Point> p2 = std::make_unique<Point>();
Point* p3 = new Point;
```

Konstruktory tak jak każda funkcja czy metoda może mieć dowolną liczbę argumentów. Konstruktor z linii 12 posiada dwa argumenty, które są wykorzystane do inicjalizacji pól obiektu. Przykład wykorzystania:

```
Point p1(2, 5); // konstruktor nadaje x i y wartości 2 i 5
std::unique_ptr<Point> p2 = std::make_unique<Point>(2, 5);
Point* p3 = new Point(2, 5);
```

Destruktor jest to metoda wywoływana przez program zawsze w momencie niszczenia instancji struktury. Obiekt może zostać zniszczony na dwa sposoby:

1. automatycznie - gdy mamy do czynienia z obiektem tworzonym automatycznie w dowolnym miejscu programu, konstruktor wywoływany jest w momencie zakończenia danego bloku instrukcji. Np.

```
{ // rozpoczęcie bloku instrukcji
    Point punkt(2, 5); // konstruktor nadaje x i y
                       // wartości 2 i 5
    Point punkt2(3,6);
    Point punkt3 = punkt.add(punkt2);

} // zakończenie bloku instrukcji
```

   W linii 7 zostanie wywołany destruktor dla każdego z obiektów automatycznych: `punkt`, `punkt2`, `punkt3`.

2. ręcznie - gdy obiekt był tworzony dynamicznie z wykorzystaniem operatora **new**, zniszczenie odbywa się w momencie wywołania operatora **delete**. Np.

```
    { // rozpoczęcie bloku instrukcji
        Point* punkt = new Point(2, 5); // konstruktor nadaje x i y
                                        // wartości 2 i 5
        Point* punkt2 = new Point(4,5);

        delete punkt; // wywoływany jest destruktor dla punkt
        delete punkt2; // wywoływany jest destruktor dla punkt2
} // zakończenie bloku instrukcji
```

(uwaga: funkcje `malloc` oraz `free` nie wywołują ani konstruktorów ani destruktorów).

W destruktorach powinno się umieszczać kod, który zwalnia zaalokowane w sposób dynamiczny zasoby, np. usuwa dynamiczne tablice, zamyka otwarte pliki, itp..

Funkcje `getX()` oraz `getY()` (linie 21 i 25) pozwalają na pobranie współrzędnych punktu bez zmiany jego stanu.

Zadania

Zadanie 1

Zaimplementuj strukturę `VerboseObject` posiadającą:

1. pole typu napisowego przechowujące nazwę obiektu,
2. konstruktor jednoargumentowy (poprzez argument należy zainicjalizować powyższe pole), wypisujący na ekranie informacje, że został wywołany (należy również wyświetlić nazwę obiektu),
3. destruktor, który w momencie niszczenia obiektu wyświetla na ekranie stosowną informację (również uwzględniającą nazwę obiektu),
4. metodę, która wypisuje na ekranie informację, że została wywołana (również uwzględniającą nazwę obiektu).

Przykładowo, wywołanie:

```
{
    VerboseObject o1("Object 1");
    o1.saySomething();
}
```

mogłoby wypisać na ekranie:

```
Object 1 constructed.
Object 1 says hello.
Object 1 destroyed.
```

Przeprowadź eksperyment (i napisz w komentarzu wnioski) polegający na przetestowaniu automatycznego (deklarując zmienną statyczną typu `VerboseObject` oraz z wykorzystaniem `std::unique_ptr`) i ręcznego (z wykorzystaniem operatora **new**) tworzenia oraz niszczenia kilku obiektów. Prześledź w jakiej kolejności dla obiektów automatycznych wywoływane są konstruktory, a w jakiej destruktory. Czy kolejność jest taka sama jak w przypadku ręcznego tworzenia obiektów?

Zadanie 2

Zaprojektuj i zaimplementuj strukturę `BinomialSolver` reprezentującą wielomian drugiego stopnia postaci $ax^2 + bx + c$ oraz pozwalającą na wyznaczenie pierwiastków równania kwadratowego postaci $ax^2+bx+c = 0$. Każdy obiekt będący instancją struktury reprezentuje jeden wielomian, czyli powinien przechowywać wszystkie współczynniki (w odpowiednich polach). Zaimplementuj poniższe funkcjonalności:

1. konstruktor przyjmujący wartości wszystkich współczynników (a, b oraz c),
2. szukanie pierwiastków równania kwadratowego $ax^2 + bx + c$ w dziedzinie liczb rzeczywistych, pierwiastki zapisz w osobnym polu/polach struktury,
3. metody dostępu – w trybie do odczytu – parametrów wielomianu (np. metody **double** `getA()` **const**),
4. metody dostępu – w trybie do odczytu – do wyliczonych pierwiastków równania (np. metody **double** `getX1()` **const**, **double** `getX2()` **const**),
5. metoda obliczającą wartość wielomianu dla zadanej zmiennej x (np. **double** `calculate(double x)` **const**).

Wskazówka 1 Obliczenia (wyznaczanie pierwiastków) możesz umieścić w konstruktorze lub wykonywać za pierwszym razem, gdy wywoływana jest jedna z metod `getX1()`/`getX2()`.

Przykładowe wykorzystanie obiektu `BinomialSolver` przedstawiono na listingu 1.

Listing 1: Przykładowe wykorzystanie obiektu BinomialSolver

```
#include <iostream>
#include "BinomialSolver.hpp"

using namespace std;

int main(){
    BinomialSolver b1(1.0, -5.0, 4.0);
    cout << "Pierwiastki rownania x*x - 5x + 4 = 0 to: " << endl;
    cout << "x1 = " << b1.getX1() << ", x2 = " << b1.getX2() << endl;

    cout << "Wartosc wielomianu x*x - 5x + 4 dla x = 1 to: ";
    cout << b1.calculate(1.0) << endl;
    cout << "Wartosc wielomianu x*x - 5x + 4 dla x = 3 to: ";
    cout << b1.calculate(3.0) << endl;

    BinomialSolver b2(1.0, 2.0, 1.0);
    cout << "Pierwiastki rownania x*x + 2x + 1 = 0 to: " << endl;
    cout << "x1 = " << b2.getX1() << ", x2 = " << b2.getX2() << endl;

    cout << "Wartosc wielomianu x*x + 2x + 1 dla x = 1 to: ";
    cout << b2.calculate(1.0) << endl;
    cout << "Wartosc wielomianu x*x + 2x + 1 dla x = -1 to: ";
    cout << b2.calculate(-1.0) << endl;

    getchar();
    return 0;
}
```

Po odpowiednim przygotowaniu implementacji struktury `BinomialSolver` oraz uruchomieniu przykładowego kodu z listingu 1 na ekranie wyświetlone zostanie:

```
Pierwiastki rownania x*x - 5x + 4 = 0 to:
x1 = 1, x2 = 4
Wartosc wielomianu x*x - 5x + 4 dla x = 1 to: 0
Wartosc wielomianu x*x - 5x + 4 dla x = 3 to: -2
Pierwiastki rownania x*x + 2x + 1 = 0 to:
x1 = -1, x2 = -1
Wartosc wielomianu x*x + 2x + 1 dla x = 1 to: 4
Wartosc wielomianu x*x + 2x + 1 dla x = -1 to: 0
```

Zadanie 3*

Znajdź sumę i iloczyn wszystkich liczb zespolonych zadanych w pliku wejściowym. Wskaż ponadto tę spośród liczb, która ma największy moduł. Każda liczba zadana jest w oddzielnym wierszu, w postaci dwóch wartości rzeczywistych, rozdzielonych spacjami, reprezentujących część rzeczywistą i urojoną liczby zespolonej. Liczbę zespoloną zaimplementuj jako strukturę (np. `Complex`) z odpowiednią metodą służącą do wyznaczania modułu (np. `modulus`).

Przykładowe dane wejściowe:

```
5.0 10.0
3.0 2.0
-4.0 6.0
2.5 -4.1
```

Wskazówka 1 Moduł liczby zespolonej postaci $z = a + bi$:

$$|z| = \sqrt{x^2 + y^2}$$

Na następne zajęcia

- Modyfikatory dostępu: **public** oraz **private**.
- Deklaracja struktur oraz klas.
- Konstruktor oraz destruktor.

Materiały: `http://www.cplusplus.com/doc/tutorial/classes/`.