# Programowanie obiektowe

wersja 2009/2010

# Laboratorium **3.**

## Zagadnienia

### 1. Dzielenie kodu

*`Extern` w praktyce, pliki nagłówkowe `.h`, dyrektywy preprocesora: `#include`, `#define`, `#ifndef`, `#endif`, kompilowanie wielu plików.*

Co w pliku nagłówkowym?

- `#include`;
- `#define` (makrodefinicje, stałe jawne);
- prototypy/deklaracje funkcji lub definicje funkcji `inline`;
- definicje zmiennych ustalonych (`const`) typów wbudowanych;
- deklaracje zmiennych `extern double zmienna`;
- deklaracje `struct`, `union`, `enum`, `class`;
- szablony funkcji;

Co poza plikiem nagłówkowym?

- definicje zmiennych;
- definicje funkcji i metod;

### 2. Komentowanie kodu

Komentarze implementacyjne

- `/* ... */` komentarze blokowe (wielolinijkowe)
- `// ...` komentarze jednolinijkowe

Komentarze dokumentacyjne

Odpowiednio zapisane komentarze dokumentacyjne (np. w stylu *Javadoc*) ułatwią analizę kodu, a przede wszystkim pozwolą na automatyczne tworzenie dokumentacji.

- `/** ... */` komentarze wielolinijkowe
- `/// ...` komentarze jednolinijkowe

#### I. Opisywanie pliku

```
/**
* @file mojplik.h
* @brief Krotki opis, jakies jedno zdanie.
*
* Tutaj opis moze byc obszerniejszy, jak w zwyklym
* komentarzu wielolinijkowym (blokowym)
*
* @author Imie Nazwisko Nazwa itp.
* @date 09.03.2010
* @version 0.9 Laborka nr 3
```

```
*/
```

### II. Opisywanie klasy

```
/**
* @class Nazwa_Klasy
* @brief Krotki opis, jakies jedno zdanie.
*
* Tutaj opis moze byc obszerniejszy, jak w zwyklym
* komentarzu wielolinijkowym (blokowym) (mozna uzywac \n \e)
*/
```

### III. Opisywanie metody (funkcji)

```
/**
* @brief Krotki opis, jakies jedno zdanie.
*
* Tutaj opis moze byc obszerniejszy, jak w zwyklym
* komentarzu wielolinijkowym (blokowym) (mozna uzywac \n \e)
*
* @param parametr1 - opis parametru 1
* @param parametr2 - opis parametru 2
* @return opis tego co jest zwracane
*/
```

### IV. Opisywanie zmiennej (pola)

```
/**
* @brief Krotki opis, jakies jedno zdanie.
*
* Tutaj opis moze byc obszerniejszy, jak w zwyklym
* komentarzu wielolinijkowym (blokowym) (mozna uzywac \n \e)
*/
```

Szczegółowy opis sposobów i form komentowania dokumentacyjnego można znaleźć z instrukcji programu *Doxygen* (www.doxygen.org) *oraz w przewodniku „How to Write Doc Comments for the Javadoc Tool”* (*http://java.sun.com/j2se/javadoc/writingdoccomments/*)

3. **Tablice dynamiczne – utrwalenie wiadomości**
   *Tablice trzywymiarowe, przesyłanie tablicy do funkcji*
4. **Konstruktor i destruktor – wstęp**
   *Użycie konstruktora; konstruktor bezparametrowy; konstruktor domniemany; konstruktor wieloargumentowy; lista inicjalizacyjna; lista inicjalizatorów; użycie* `this`;
5. **Funkcje i pola statyczne**

## **Zadanie 1.** (Dzielenie kodu, kompilowanie wielu plików)

1. Stwórz projekt z plikami **`Matematyka.h`** oraz **`Matematyka.cpp`**;
2. W odpowiedni sposób w plikach zdefiniuj stałe matematyczne **`PI`**, **`E`**;
3. Zaimplementuj funkcję **`abs`**, która zwróci wartość bezwzględną liczby. Funkcję należy przeciążyć dla następujących typów: `int`, `float`, `double`;
4. Zaimplementuj funkcję **`int max(int a, int b)`**, która zwróci maksimum z dwóch liczb `int`;
5. Analogicznie zaimplementuj funkcję **`min`**;
6. Zaimplementuj przeciążoną funkcję **`int max(int* tab, int rozmiarTab)`**, która zwróci największą wartość z tablicy typu `int`;
7. Napisz makrodefinicję **`ROZM(t)`**, która obliczy rozmiar stałej tablicy jednowymiarowej dowolnego typu;

## **Zadanie 2.** (Zmienne statyczne)

1. Zaimplementuj klasę **`Mat`** (nie trzeba w osobnych plikach), która będzie zawierała statyczną zmienną ustaloną **`PI`** oraz zmienną ustaloną (niestatyczną) **`E`**;
2. Porównaj użycie (odwołanie się) do wartości **`PI`** oraz **`E`** z klasy **`Mat`**;
3. Czy zmienna statyczna może być ustalona?

## **Zadanie 3.** (Konstruktor i destruktor)

1. Sprawdź, czy destruktor może znajdować się w sekcjach `private`, `protected`, `public`;
2. Wyjaśnij, czy destruktor i konstruktor może przyjmować parametry, czy może być przeciążony, jakie typy może zwracać;
3. Wyjaśnij, czy konstruktor i destruktor może być składnikiem statycznym klasy;
4. Czy konstruktor może wywołać inny konstruktor – poprzyj odpowiednim przykładem;
5. Czy w konstruktorze mogą być argumenty domniemane;
6. W jaki sposób można sprawdzić adres konstruktora/destruktora;
7. Czy destruktor likwiduje obiekt?
8. Co to jest lista inicjalizacyjna, czym natomiast jest lista inicjalizatorów?
9. Niech istnieje klasa **`Dom`**, w jaki sposób można dynamicznie stworzyć tablicę 20 obiektów tego typu?
10. W jaki można odnieść się do składników (pól i metod) klasy gdy używamy obiektu, referencji na obiekt, wskaźnika na obiekt?
11. Czy można zdefiniować jednocześnie konstruktory bezparametrowy i domniemany? Uzasadnij;

12. Czy konstruktor bezparametrowy jest zawsze zdefiniowany niejawnie?

## **Zadanie 4.*** (Dodatkowe)

Niech **`tab`** jest tablicą tworzoną dynamicznie. Zaproponuj sposób, którym wykażesz, czy usunięcie tablicy za pomocą **`delete tab`** zwolni wcześniej zaalokowaną pamięć.

## **Zadanie 5.*** (Dodatkowe - Tablice dynamiczne)

Napisz program, który dynamicznie stworzy trzywymiarową tablicę dla typów **`int`**. Użytkownik powinien mieć możliwość zdefiniowania wymiarów tablicy. Tablicę wypełnij wartościami losowymi z zakresu <-10;10> (innymi z każdym uruchomieniem programu).

## **Zadanie 6.** (Komentarze dokumentacyjne)

1. Klasę **`Data`** umieść w osobnych plikach.
2. Wszystkie pliki, klasy, funkcje i atrybuty opatrz stosownymi komentarzami dokumentacyjnymi (nie dotyczy pliku zawierającego funkcję `main`).
3. Zaimplementuj funkcję sprawdzającą czy rok jest przestępny.

### **Dodatkowe***

4. Zaimplementuj metodę **`char* czytajDzien()`**, która zwróci nazwę dnia tygodnia;
5. Zaimplementuj sprawdzanie poprawności daty (np. 30 lutego, -9 maja, 17 miesiąc itp.) - sprawdzanie poprawności powinno nastąpić w momencie tworzenia obiektu i podczas próby przypisania wartości, 29 lutego powinien być dostępny wyłącznie dla roku przestępnego.

## **Zadanie 7.** (Modelowanie i implementacja)

Zaprojektuj i zaimplementuj klasę **`Punkt2D`**, która będzie reprezentowała współrzędne punktu na płaszczyźnie. Zaimplementuj przynajmniej jedną funkcję statyczną. Model przedstaw na diagramie UML. Zaprezentuj użycie konstruktorów bezparametrowego i wieloargumentowego, użyj listy inicjalizacyjnej. Klasę dołącz w osobnych plikach, zaopatrzonych w komentarze dokumentacyjne (i jeżeli jest potrzeba także w implementacyjne).

## Zadanie 8.* (Dodatkowe)

Jednym z najpopularniejszych problemów optymalizacyjnych jest problem komiwojażera (*travelling salesman problem*), którego przedmiotem jest znalezienie najkrótszej drogi przechodzącej przez wszystkie miasta w taki sposób, aby każde miasto odwiedzić dokładnie jeden raz, a na końcu powrócić do początku. Na stronie Uniwersytetu Heidelberg (http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/) dostępne są instancje testowe dla problemu (interesuje nas wersja *Symmetric TSP*).

Napisz program, który wczyta wybrany przez użytkownika plik .tsp, a w przypadku, jeśli parametr EDGE_WEIGHT_TYPE będzie zdefiniowany jako EUC_2D dynamicznie stworzy tabelę odległości między miastami (dla EUC_2D zdefiniowane są współrzędne miast położonych na płaszczyźnie euklidesowej). Wykorzystaj klasę **`Punkt2D`** z Zadania 7. Na życzenie użytkownika program wydrukuje odpowiednie wartości (ze stworzonej tablicy).

## Zadanie 9. (Quiz)

Zredaguj 5 pytań testowych (np. jednokrotnego wyboru, wielokrotnego wyboru, dopasowanie odpowiedzi itp.), które pozwolą sprawdzić wiedzę nabytą podczas tego laboratorium.

## Zaliczenie ćwiczenia

1. Zadania obowiązkowe: 1, 2, 6, 7
2. Raport: Zadanie 3, 9, modele UML

Życzę owocnej pracy