Języki i metody programowania 2

Lab 5

Metody const

W odniesieniu do metod klasy oznacza, że dana metoda nie może modyfikować elementów składowych klasy jak również nie może wywoływać innych metod niż te zdeklarowane jako **const**. Jeśli można, warto zadeklarować metodę jako *const*. Umożliwi to poprawne korzystanie z obiektów tej klasy zadeklarowanych jako stałe.

Zmienne/obiekty const

W odniesieniu do zmiennych obiektów i innych zmiennych, oznaczają że nie można modyfikować zmiennej/obiektu. Innymi słowy nie można wywoływać na rzecz danego obiektu innych metod niż zadeklarowane jako **const.**

Funkcje zaprzyjaźnione

Funkcja zaprzyjaźniona definiowana jest poza zasięgiem klasy, ale informacja o tym że jest ona *przyjacielem* musi znaleźć się w klasie:

```
//Plkik Matrix.h
class Matrix{
  double** data;
  ...
  friend void reset(Matrix& m);
  ...
};

// Plik main.cpp

// Funkcja może modyfikować prywatne dane klasy
// Matrix, ponieważ została określona jako friend
void reset(Matrix& matrix){
  for(int r  = 0; r < matrix.rows; r++)
    for(int c  = 0; c < matrix.cols; c++)
      matrix.data[r][c] = 0;
}
```

Klasy zaprzyjaźnione

Analogicznie klasy zaprzyjaźnione mają nieograniczony dostęp do wszystkich pól i metod klasy które są przyjaciółmi. Jeśli klasa *Node* ma być przyjacielem klasy *Lista*, to w tej drugiej należy umieścić następującą deklarację:

```
class Lista{
  ...
  friend class Node;
  ...
};
```

Pola statyczne

Pola statyczne są swego rodzaju zmiennymi globalnymi, należącymi jednak do zasięgu klasy. Zmienna statyczna jest współdzielona przez wszystkie obiekty klasy. Do zmiennych statycznych można odwoływać się nie mając utworzonego obiektu.

Słowo kluczowe **static** dodaje się tylko i wyłącznie podczas deklaracji zmiennej. Użycie słowa kluczowego *static* podczas inicjalizacji zmiennej powoduje błąd składniowy.
Deklaracja zmiennej statycznej w pliku nagłówkowym:

```
class Matrix{
  public:
    static int counter;
    Matrix(){counter++;}
   ~Matrix(){counter--;}
};
```

Zmienna statyczna musi zostać zainicjalizowana w przestrzeni pliku. Nie można jej inicjalizować w konstruktorze, lub innej funkcji.

Inicjalizacja zmiennej statycznej w pliku cpp:

```
#include "Matrix.h"

int Matrix::counter = 0;
...
```

Odwoływanie się do pól statycznych:

```
int main(){
  Matrix m;
  cout << Matrix::counter <<endl;
  cout << Matrix.counter << endl;
}
```

Metody statyczne

Metody statyczne można wywoływać bez konieczności tworzenia obiektów klasy. Można z tego wywnioskować, że metody statyczne **nie posiadają** wskaźnika **this**. Nie można zatem wewnątrz metod statycznych wywoływać żadnych innych metod niestatycznych, ani odwoływać się do pól niestatycznych.
Metody statyczne należą jednak do zasięgu klasy i mają dostęp do wszystkich pól klasy:

```
class Matrix{
  static void print(Matrix& m);
};

void Matrix::print(Matrix & m){
  for(int r  = 0; r < matrix.rows; r++){
    for(int c  = 0; c < matrix.cols; c++)
      cout << matrix.data[r][c] << endl;
    cout << endl;
  }
}
```

Zadania

1. Napisz dwie klasy: *Parent* i *Child*. Klasa *Parent* powinna mieć takie pola jak imię, nazwisko, wiek, oraz dziecko (zakładamy dla uproszczenia, że rodzic ma tylko jedno dziecko). Dziecko powinno mieć takie pola jak imię, nazwisko, wiek, szkoła. Zdefiniuj klasę *Parent* jako zaprzyjaźnioną klasy *Child*. Przetestuj, czy można modyfikować zmienne prywatne klasy *Child* z poziomu metod klasy *Parent*. Napisz na przykład metodę *assignToOtherSchool(string name)* która będzie zmieniać szkołę dziecka operując bezpośrednio na jego danych.

2. Napisz klasę *Martian* (Marsjanin), która będzie miała statyczne pole *martianCount*, określające liczbę stworzonych obiektów klasy *Martian*. Każdy Marsjanin powinien atakować gdy liczba wszystkich Marsjan jest większa od 5 i ukrywać się w przeciwnym wypadku. Napisz program który w pętli nieskończonej będzie tworzył lub usuwał obiekty klasy *Martian* i wywoływał metodę *Attack* dla wszystkich Marsjan. Metoda *Attack* będzie zwracać informację czy atak się powiódł (czy Marsjanin przeżył).

3. Zaimplementuj klasę o nazwie *Matrix*, która będzie reprezentować macierz o dowolnych rozmiarach. Klasa *Matrix* powinna posiadać konstruktor parometrowy określający jej wymiary, konstruktor bezparametrowy, oraz kopiujący. Dopisz konstruktor, który będzie przyjmować napis const char* (lub obiekt klasy string z biblioteki string.h) w notacji Matlaba i parsował go, aby można było stworzyć obiekt Matrix w taki sposób:

```
Matrix matrix("[1i3 2i5 3; 3 4 5; 6 7 8]");
```

   Klasa Matrix powinna udostępniać metody pozwalające na ustawianie/pobieranie jej elementów.
   Klasa Matrix powinna udostępniać metodę umożliwiającą dodawanie macierzy. Metoda ta powinna pobierać jako parametr inny obiekt klasy Matrix i zwracać wynik będący sumą macierzy reprezentowanej przez obiekt na rzecz którego wywoływana jest metoda i macierzy reprezentowanej przez obiekt podany jako parametr. Przykładowe wywołanie:

```
Matrix firstMatrix("[1 2 3;3 4 5; 2 3 4]");
Matrix secondMatrix("[3 2 1; 5 4 3; 7 6 5]");
Matrix result = firstMatrix.add(secondMatrix);
```