Laboratoria Programowania w C++

Generated by Doxygen 1.9.1

Chapter 1

Laboratorium 4 - Polimorfizm na przykładzie rysowania kształtów

1.0.1 Tresc zadan dla Panstwa (aktualniejsza jest w <tt>README.md</tt>)

Zadanie 0: absolutnie obowiazkowe, chociaz bez punktow

1. Pierwsza rzecza jest poprawa bledow kompilacji, czyli wpisanie poprawnych Panstwa danych w pliku↩ : main.cpp
2. Oddanne zadanie musi sie bezwzglednie kompilowac na systemie Linux:
   - Jesli sie nie skompiluje to jest to 0 punktow za zadanie!
   - Oczywiscie w razie problemow z kompilacja prosze się zgłaszać/pisać.
   - Dobrze, jesli nie byloby warningow kompilacji, ale za to nie obnizam punktow.
   - Aby się upewnić, że się kompiluje można skorzystać z `narzędzia Bobot`
3. Oddane zadanie nie powinno crashować na żadnym teście, jeśli crashuje proszę zrobic implementacje -fake, ktora nie dopusci do crasha nawet jesli test bedzie failowal, ale za to testy nie beda sie crashowaly. W przypadku crasha biorę pod uwagę tylko tyle testów, ile przejdzie do czasu crasha!
4. Mam program antyplagiatowy, dlatego prosze pracowac samodzielnie!
   - Osoby ktore udostepniaja swoje rozwiazania rowniez beda mialy kare!
   - Na ukaranie prowadzący ma czas 2 tygodnie po terminie oddania, czyli nawet jak ktoś otrzyma punkty wcześniej ma pewność, że za oszustwa/łatwowierność dosięgnie go niewidzialna ręka sprawiedliwości.
5. Zadanie z zalozenia bedzie sprawdzane automatycznie, przez testy jednostkowe dostepne w pliku↩ : `matrixTests.cpp`,
6. ∗Dobrze jakby nie bylo warningow kompilacji (flagi: `-Wall -Wextra -pedantic -Werror`, a dla hardcorów jeszcze: `-Weffc++`)
7. Punkty beda odejmowane za wycieki pamieci (jest podpiety `valgrind`)
8. Niewykluczone jest sprawdzanie reczne - zaleznie od prowadzacego dana grupe.

Tresc do implementacji - szukaj w plikach ∗.h

Chapter 2

Shapes drawing

W zadaniu chodzi o to aby użyć polimorfizmu w C++ w poprawny sposób, pamiętając o koniecznych elementach. Przy okazji prostej implementacji powstaje ciekawa kompozycja umożliwiająca rysowanie złożonych kształtów jak przykładowo:

```
21:            *******
20:           ***********
19:          *************
18:         ***************
17:        *****************
16:       *******************
15:       *******************
14:      **********************
13:      **********************
12:      ***      ***      ***
11:      ***      ***      ***
10:      ***      ***      ***
 9:      ***      ***      ***
 8:      ***      ***      ***
 7:      ***      ***      ***
 6:      ***      *************
 5:      ***      *************
 4:      ***      *************
 3:      ***      *************
 2:      ***      *************
 1:      **********************
```

2.1 Klasa abstrakcyjna Shape:

Implementacja klasy czysto abstrakcyjnej `Shape`, majacej funkcje opisane niżej. Zadanie ma na celu "ugryzienie" polimorfizmu dynamicznego, w tym również o kompozycje klas w formie drzewa.

1. Klasa `Shape` powinna mieć metodę `bool isIn(int x, int y) const = 0;`, ktora zwraca informacje czy dany punkt jest wewnatrz figury czy nie
2. Proszę zaimplementować klasę `Rectangle` dziedziczącą po `Shape` i implementującą powyższą metodę.
   (a) Implementacja klasy powinna być dokonana w nowo-utworzonych plikach: `rectangle.h` i `rectangle.cpp`
   (b) Konstruktor powinien przyjmować położenia współrzędnych dwóch: dolnegolewego i gornegoprawego `(xFrom, yFrom, xTo, yTo)`
   (c) Odpowiada to prostokątowi o bokach równoległych do osi wykresu
3. Proszę zaimplementować klasę `Circle` dziedziczącą po `Shape` i implementującą jej metodę.
   (a) Implementacja klasy powinna być dokonana w nowo-utworzonych plikach: `circle.h` i `circle.cpp`

   (b) Konstruktor powinien przyjmować wspolrzedne środka, oraz promien `(int xCenter, int yCenter, int radius)`

4. Proszę zaimplementować klasę-kompozyt `ShapeComposite` dziedziczącą po `Shape` i implementującą jej metodę. Klasa ta w konstruktorze powinna przyjąć:
   (a) dwie instancje `shared_ptr<Shape>`
   (b) operacje na zbiorach `enum class ShapeOperation`: `INTERSECTION`, `SUM`, `DIFFERENCE`
   (c) w oparciu o to bedzie mozna cala hierarchie figur polaczyc w jedno drzewo, dla ktorego bedzie mozna zapytac czy dany punkt jest w hierarchii, czy nie (metoda `isIn`).
5. Opcjonalnie mozna sobie zaimplementowac klase `Stage` rysujaca na konsole.

Informacje o co chodzi w paczce, na co zwrócić uwagę, jak czytać testy znajdują się w materiale `wideo`.

2.2 Uwaga:

Wszystkie atrybuty powinny być prywatne, konstruktory i metody - publiczne, metody większe niż 1-linijkowe powinny być zadeklarowane w klasie, zdefiniowane poza klasą, obiekty typów klasowych powinny być w miarę możliwości przekazywane w argumentach funkcji przez referencję, proszę też stosować słówko "const" w odpowiednich miejscach. Wszystkie metody, które mogą być stałe proszę aby były.

1. Mozna tworzyc dowolna ilosc metod pomocniczych, jednakze aby byly one prywatne.
2. Gettery i settery operujace na liczbach, ktore nie rzucaja wyjatku, warto zadeklarowac jako `noexcept`.
3. Co się da na listę inicjalizacyjną konstruktora.
4. Za złe zarządzanie pamięcią (wycieki, pisanie poza pamięcią) powodują odejmowanie punktów

`Bardziej szczegółowe informacje jak pisać programy w ładnym stylu` dla zaawansowanych.

2.3 Podpowiedzi:

1. Warto sobie stworzyć pomocniczą strukturę Point do trzymania współrzędnych.
2. Proszę pamiętać o dodawaniu klas w przestrzeni nazw Shapes
3. Klasa Shape powinna mieć zdefiniowaną prze nas jedną specjalną metodę poza `isIn`, każda klasa bazowa w polimorfiźmie powinna.
4. Pamiętajcie o słówku kluczowym `override` przy metodzie `isIn`.

2.4 Ocenianie:

1. Ocenia `Bobot`, na ten moment w następujący sposób:
   (a) Kompilacja nadesłanego rozwiązania - bez tego zero punktów. Bobot pracuje na Linuxie, używa kompilatora g++.
   (b) Uruchamianie testów - za każdy test, który przejdzie są punkty, ale mogą być odjęte w kolejnych krokach.
   (c) Jeśli program się wywala na którymś z testów (to się pojawia często u osób pracujących na Windowsie - ten system pozwala pisać po nie-swojej pamięci, Linux nie pozwala) lub jest timeout - wtedy będzie przyznane tyle punktów ile przechodzi testów **minus dwa za karę**.
   (d) Jest odpalane narzędzie `valgrind`, które sprawdza czy umiemy obsługiwać pamięć w praktyce - jeśli nie to **minus punkt**.
   (e) Odpalane są też inne narzędzia takie jak `cppcheck`, czy `fawfinde` i inne. One nie odejmują punktów, no ale mają pomóc w pisaniu porządnych programów. Nie olewajmy tego.
   (f) Antyplagiat - za wykrycie plagiatu (jest specjalne narzędzie) otrzymuje się 0 punktów. Róbmy więc samemu!

2.5 Najczęstrze błędy/pytania/problemy:

1. Zaimplementowałem metodę klasy w pliku źródłowym dodałem `using namespace Shapes`, a linker sygnalizuje, że niezdefiniowałem Shapes::Klasa::metoda.
   (a) `using namespace` nie dodanie do danej przestrzeni nazw czegokolwiek, to jedynie powoduje dostęp do składowych tej przestrzeni nazw tak jakby jej nie było. Dlatego nie ma wyjścia - trzeba zdefiniować metodę w taki sposób `... Shapes::Klasa::metoda(...) {...}`
2. Dodałem plik z implementacją danej klasy, a testy uparcie twierdzą, że nie.
   (a) To wynika z faktu, że "aktywowanie" odpowiedniej części kodu odbywa się na etapie kompilacji - tam jest wykrywane czy plik istnieje czy nie. Aby skompilowały się testy musi się zmienić coś w pliku testów lub w pliku includowanym. **Konkretnie: po dodaniu pliku przebuduj cały projekt!**
3. Jak zaimplementowac `isIn` dla kola?
   (a) Matematyka - czy odległość punktu od środka koła jest nie większa niż promień.
4. Napisałem klasę `Rectangle`, która dziedziczy po `Shape`, a kompilator sygnalizuje jakby nie było dziedziczenia.
   (a) Domyślnie dziedziczenie w C++ jest prywatne, należy więc pamiętać o słówku `public`.
   (b) Czy zdefiniowano klasę w odpowiedniej przestrzeni nazw?

2.6 Pytania po implementacji ćwiczenia:

1. (Jak macie pomysł to podrzućcie)

2.7 Zadania, które warto zrobić (uwaga: nie będzie za to punktów, tylko coś cenniejszego - umiejętności)

1. (Jak macie pomysł to podrzućcie)

2.8 Jak skonfigurować sobie pracę nad paczką:

W formie `wideo` do poprzedniej paczki (link do projektu inny, reszta analogiczna).
Alternatywnie poniżej jest to spisane w kolejnej sekcji

2.8.1 Grading (section copied from Mateusz Ślażyński, of course he agreed):

- [ ] Make sure, you have a **private** group
  - `how to create a group`
- [ ] Fork this project into your private group
  - `how to create a fork`
- [ ] Add @bobot-is-a-bot as the new project's member (role: **maintainer**)
  - `how to add an user`

2.8.2 How To Submit Solutions

1. [ ] Clone repository: `git clone` (clone only once the same repository):
   ```bash git clone <repository url> ```
2. [ ] Solve the exercises
3. [ ] Commit your changes
   ```bash git add <path to the changed files> git commit -m <commit message> ```

4. [ ] Push changes to the gitlab main branch

   ```bash git push -u origin main ```

The rest will be taken care of automatically. You can check the `GRADE.md` file for your grade / test results. Be aware that it may take some time (up to one hour) till this file. Details can be found in `./logs/` directory where You can check compilation results, tests logs etc.

2.8.3 Project Structure

```
.
 CMakeLists.txt          # CMake configuration file - the file is to open out project in our IDE
 main.cpp                # main file - here we can test out solution manually, but it is not required
 shape.h                 # file to implement abstract class
 trescPdf.pdf            # documentation in PDF (generated by Doxygen)
 tests                   # here are tests for exercise, inner CMakeLists.txt, GTest library used by tests
    CMakeLists.txt      # iner CMake for tests - it is included by outter CMake
    lib                 # directory containing GTest library
    shapesTests.cpp     # tests v1
 doxyfiles               # here is logo for documentation generated by Doxygen
    cppLogo.png         # logo
 Doxyfile                # here is prepared file for Doxygen, to generate documentation when we type `doxygen
 Dockerfile              # this file contains instructions how to run tests in embedded Ubuntu
 README.md               # this file
```
```

Chapter 3

Todo List

Member FIRSTNAME

Uzupelnij swoje dane:

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Shapes . . . . . ??

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Shapes::Point . . . . . ??
Shapes::Shape . . . . . ??

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

main.cpp . . . . . ??
shape.h . . . . . ??
tests/shapesTests.cpp . . . . . ??

Chapter 7

Namespace Documentation

7.1 Shapes Namespace Reference

Classes

- struct Point
- class Shape

Chapter 8

Class Documentation

8.1 Shapes::Point Struct Reference

`#include <shape.h>`

Public Attributes

- int x_
- int y_

8.1.1 Detailed Description

Definition at line 88 of file shape.h.

8.1.2 Member Data Documentation

8.1.2.1 x_

`int Shapes::Point::x_`
Definition at line 90 of file shape.h.

8.1.2.2 y_

`int Shapes::Point::y_`
Definition at line 90 of file shape.h.
The documentation for this struct was generated from the following file:

- shape.h

8.2 Shapes::Shape Class Reference

`#include <shape.h>`

8.2.1 Detailed Description

Definition at line 93 of file shape.h.
The documentation for this class was generated from the following file:

- shape.h

Chapter 9

File Documentation

9.1 CMakeLists.txt File Reference

9.2 tests/CMakeLists.txt File Reference

Functions

- project (tests) add_subdirectory(lib) include_directories($

9.2.1 Function Documentation

9.2.1.1 project()

```
project (
            tests  )
```

Definition at line 1 of file CMakeLists.txt.

9.3 main.cpp File Reference

```
#include <iostream>
```

Include dependency graph for main.cpp:

9.4 README.md File Reference

9.5 shape.h File Reference

Classes

- struct Shapes::Point
- class Shapes::Shape

Namespaces

- Shapes

9.6 tests/shapesTests.cpp File Reference

```
#include <cmath>
#include <gtest/gtest.h>
```

Include dependency graph for shapesTests.cpp:

Functions

- TEST_F (RectangleTester, constructorOfRectangleStartingInBeginningOfSystemCoordinate)
- TEST_F (RectangleTester, isInMethodOfRectangleStartingInBeginningOfCoordinateSystem)
- TEST_F (RectangleTester, isInMethodOfRectangleStartingNotInBeginningOfCoordinateSystem)
- TEST_F (RectangleTester, constructorOfRectangleWithBothSitesEqualToZero)
- TEST_F (CircleTester, constructorOfCircleStartingInBeginningOfCoordinateSystem)
- TEST_F (CircleTester, isInMethodOfCircleStartingInBeginningOfCoordinateSystem)
- TEST_F (CircleTester, isInMethodOfCircleStartingNotInBeginningOfCoordinateSystem)
- TEST_F (CircleTester, constructorOfCircleWithRadiusEqualToZero)
- TEST_F (ShapeCompositeTester, sumOfSqhareAndCircle)
- TEST_F (ShapeCompositeTester, intersectionOfSqhareAndCircle)
- TEST_F (ShapeCompositeTester, differenceOfSqhareAndCircle)
- TEST_F (ShapeCompositeTester, drawingHouse)

9.6.1 Function Documentation

9.6.1.1 TEST_F() [1/12]

```
TEST_F (
            CircleTester ,
            constructorOfCircleStartingInBeginningOfCoordinateSystem  )
```

y 10: **** 9: ****** 8: ******* 7: ******** 6: ********* 5: ********** 4: ********** 3: ********** 2: ********** 1: ********** 0: ********** 0: 01234567890 x

Definition at line 148 of file shapesTests.cpp.

9.6.1.2 TEST_F() [2/12]

```
TEST_F (
            CircleTester ,
            constructorOfCircleWithRadiusEqualToZero  )
```

Definition at line 247 of file shapesTests.cpp.

9.6.1.3 TEST_F() [3/12]

```
TEST_F (
            CircleTester ,
            isInMethodOfCircleStartingInBeginningOfCoordinateSystem  )
```

circle: y 10: **** 9: ****** 8: ******* 7: ******** 6: ********* 5: ********** 4: ********** 3: ********** 2: ********** 1: ********** 0: ********** : 01234567890 x

Definition at line 178 of file shapesTests.cpp.

9.6.1.4 TEST_F() [4/12]

```
TEST_F (
            CircleTester ,
            isInMethodOfCircleStartingNotInBeginningOfCoordinateSystem  )
```

circle: y 12: ***** 11: ******* 10: ********* 9: ********** 8: ********** 7: ********** 6: ********** 5: ********** 4: ********* 3: ******* 2: ***** : 012345678901 x

Definition at line 215 of file shapesTests.cpp.

9.6.1.5 TEST_F() [5/12]

```
TEST_F (
            RectangleTester ,
            constructorOfRectangleStartingInBeginningOfSystemCoordinate  )
```

Definition at line 80 of file shapesTests.cpp.

9.6.1.6 TEST_F() [6/12]

```
TEST_F (
            RectangleTester ,
            constructorOfRectangleWithBothSitesEqualToZero  )
```

Definition at line 133 of file shapesTests.cpp.

9.6.1.7 TEST_F() [7/12]

```
TEST_F (
            RectangleTester ,
            isInMethodOfRectangleStartingInBeginningOfCoordinateSystem  )
```

Definition at line 95 of file shapesTests.cpp.

9.6.1.8 TEST_F() [8/12]

```
TEST_F (
            RectangleTester ,
            isInMethodOfRectangleStartingNotInBeginningOfCoordinateSystem  )
```

Definition at line 114 of file shapesTests.cpp.

9.6.1.9 TEST_F() [9/12]

```
TEST_F (
            ShapeCompositeTester ,
            differenceOfSqhareAndCircle  )
```

Definition at line 380 of file shapesTests.cpp.

9.6.1.10 TEST_F() [10/12]

```
TEST_F (
            ShapeCompositeTester ,
            drawingHouse  )
```

Definition at line 435 of file shapesTests.cpp.

9.6.1.11 TEST_F() [11/12]

```
TEST_F (
            ShapeCompositeTester ,
            intersectionOfSqhareAndCircle  )
```

Definition at line 323 of file shapesTests.cpp.

9.6.1.12 TEST_F() [12/12]

```
TEST_F (
            ShapeCompositeTester ,
```

```
            sumOfSqhareAndCircle  )
```

Definition at line 263 of file shapesTests.cpp.