Problem

Korzystając z przygotowanej w poprzednim zadaniu infrastruktury (tzn. odczytu figur z pliku) chcemy doprowadzić sprawę do końca, czyli wyświetlenia figur w oknie.
W tym odcinku pominiemy kwestię skalowania mapy – w dostarczonym pliku testowym współrzędne punktów figur są zadane w pikselowym układzie współrzędnych okna.
Sprawą, która przysporzy najwięcej pracy będzie reorganizacja (*refaktoring*) kodu: zastąpienie klasy `figure` „od wszystkiego” hierarchią klas, w której każdy rodzaj figury będzie mieć odpowiednią klasę, a `figure` będzie sprowadzone do bazy (abstrakcyjnej) całej hierarchii.
Sporo radości powinno też dostarczyć utworzenie projektu, w którym pliki nagłówkowe i biblioteki są rozrzucone po licznych folderach.

Projekt

Zmiany z punktu widzenia C++ są dość dokładnie opisane w materiałach wykładowych. Tu zajmiemy się stroną wykonawczą całego przedsięwzięcia.
Przyjrzyjmy się najpierw tworzeniu projektu na podstawie szablonu zapisanego w końcówce poprzedniego odcinka.
Zaczynam standardowo: `New > Project`, ale w oknie `New from template` wybieram `User templates` i dalej `graph_lib_project`. Z wyborem foldera projektu trzeba uważać – potrzebujemy nowy, pusty folder, którego jeszcze nie ma! Tworzę go w oknie wyboru foldera i upewniam się, że nazwę nowego foldera (u mnie `Zad_2`) widać w polu `Folder`.
`Code::Blocks` proponuje jeszcze zmianę nazwy projektu. Skrzętnie z tego korzystam i zmieniam ją na `zad_2`.

Projekt

Po utworzeniu projektu zmieniam natychmiast nazwę pliku main.cpp na `figure_test.cpp`. W drugim kroku kopiuję `figure_test.cpp` i `mapa_test02.txt` z archiwum ćwiczeń do foldera projektu.
Uwaga: warto utrzymywać strukturę folderów stosunkowo prostą i nie mieszać bez potrzeby projektów między sobą.
Ostatnim dodatkiem do projektu jest plik `figure.h` z zadania pierwszego dużego (to oznacza skopiowanie pliku do folderu projektu i dołożenie go do plików projektu `Project > Add files`).
Zaczniemy od odchudzenia pliku `figure.h`: powinny zostać tu jedynie definicje klas i deklaracje funkcji pomocniczych (nie należących do klas). Implementacje metod i funkcji trzeba przenieść do pliku `figure.cpp`.

Modyfikacje `figure.h` i `figure.cpp`

Wygląda na to, że możemy zająć się teraz konkretną pracą nad kodem, bo kompilator wyświetla:

```
error: 'get_figure' was not declared in this scope|
error: 'class figure' has no member named 'get_shape'|
```

Chciałbym podzielić sobie pracę i nie zajmować się wszystkim na raz. Ponieważ `get_shape` (utworzenie kształtów z `graph_lib`, które będzie można wyświetlić w oknie) jest rzeczą nową, odłożyłbym to na za chwilę biorąc w komentarz linie funkcji `main` odpowiedzialne za wyświetlenia okna i powiązanie z nim figur:

```
    //Simple_window wnd(Point(100, 100), 600, 400, "Okno");
    //for (auto pf : figs)
    //    wnd.attach(*(pf->get_shape()));
    //wnd.wait_for_button();
```

Refaktoring

Refaktoring kodu można podsumować w następujących krokach:

1. Implementacja funkcji `get_figure`, która ma zastąpić operator >> z klasy `figure`.
2. Implementacja klas `Rect` i `Circ` w minimalnej postaci (tzn. tylko z konstruktorem i statyczną `class_id`), co wymaga zmian także w klasie bazowej.
3. Poprawki operatora << w klasie `figure` (dołożenie wirtualnej funkcji `get_id` – czystej w klasie bazowej i zaimplementowanej w pochodnych). Nie mam już składowej `id` w `figure`!
4. Zmiana `bbox` na metodę wirtualną (+ przeniesienie fragmentu kodu do `bbox` w klasie `Circ`)

Nie trzeba chyba wspominać, że w `main` powieliłem sobie „echo” figur na konsolę, żeby na bieżąco śledzić, czy figury odczytują się właściwie.

Wyświetlanie

Jestem gotowy od usunięcia komentarza części wyświetlającej figury w oknie. Od razu trzeba mi zdefiniować w `figure`:

```
virtual Graph_lib::Shape* get_shape() const = 0;
```

Implementacje w `Rect` i `Circ` będą wołać konstruktory pochodnych po `Shape`, odpowiednio `Rectangle` i `Circle`. Zanim się wezmę za to, muszę poradzić sobie z masą błędów, które pojawiły się, kiedy włączyłem (`#include`) na początku `figure.h` plik `graph.h`. Powodem jest makrodefinicja z pliku `std_lib_facilities.h`:

```
#define vector Vector
```

Jest bardzo szkodliwa i należy ją wyłączyć (zakomentować):

```
//#define vector Vector
```

Po tym ruchu mogę wrócić do implementacji `get_shape`.

Wyświetlanie

Oba przypadki wymagają nieco pracy. Do `Rectangle` potrzebuję struktur `Graph_lib::Point` , bo takie dwa punkty są argumentami konstruktora. Najłatwiej będzie mi zdefiniować operator konwersji w `FPoint`:

```
operator Graph_lib::Point() const
```

W konstruktorze `Circle` drugi parametr jest promieniem koła, więc dokładam sobie funkcję:

```
float distance(const FPoint & lf, const FPoint & rt);
```

Zakładam, że przyda mi się dokładna (`float`) odległość między punktami; w wywołaniu konstruktora `Circle` dołożę konwersję wyniku tej funkcji do `int`.

W tym momencie kompilacja kończy się sukcesem i można się spodziewać wyświetlenia figur w oknie programu.

Oddawanie

Definicje wszystkich (czterech) klas i funkcji pomocniczych proszę umieścić w pliku `figure.h`. Implementacje metod klas (o ile są dłuższe, niż jedna instrukcja) oraz funkcji pomocniczych proszę zamieścić w pliku `figure.cpp`.

Rozwiązania (`figure.h` i `figure.cpp`) należy złożyć w `Moodle` do:

3 kwietnia 2022 23:59