Języki Programowania, Laboratorium 9, 10.12.2012

1. Zaimplementować klasę `Figura`, będącą klasą bazową dla rzeczywistych figur rysowanych w programie. Klasa `Figura`, przechowuje następujące pola:

`kolor krawędzi figury` - użyć typu string,
`grubość krawędzi figury` (grubości linii przyjmują wartości rzeczywiste, typ `float*`),

UWAGA: dla przećwiczenia wskaźników pole **gr_krawedzi** ma być zadeklarowane jako wskaźnik na `float`! Należy pamiętać o alokowaniu pamięci dla tego pola i odpowiednim zwalnianiu jej w destruktorze.

Pola dostępne są tylko dla klas potomnych.
W części publicznej klasa ta zawiera deklarację wirtualnej metody `Wypisz()`.

2. Jako przykład klasy odpowiadającej danemu rodzajowi figur **zaimplementować klasę `Okrag`**, będącą klasą potomną klasy `Figura`, zawierającą pola danych potrzebne do odrysowania figury (współrzędne środka, promień) oraz odpowiednie metody. Zdefiniować konstruktor domyślny (wszystkie wartości = 0), jak również konstruktor inicjalizujący wszystkie pola klasy `Okrag` (wraz z kolorem krawędzi oraz jej grubością).

- Metoda `Wypisz` powinna wypisać na ekran informację "`Okrag kolor grubosc (x,y,R)`", gdzie kolejno mamy wartości „x y R” w każdej linijce dla osobnego okręgu.

3. Napisać program, testujący działanie klasy:
- Stworzyć wskaźnik (wraz z obiektem) `f1` na klasę `Figura` (używając konstruktora domyślnego). Wypisać (używając metody `Wypisz`).
- Stworzyć obiekt `o1` klasy `Okrąg` (używając konstruktora domyślnego). Wypisać (używając metody `Wypisz`).
- Stworzyć wskaźnik (wraz z obiektem) `o2` na klasę `Okrąg` (x = 2, y= 2, R = 5, kolor zielony, grubość 1). Wypisać.

Następnie należy stworzyć nowy wskaźnik na klasę `Figura`.
```
Figura* wskfig;
```
I przypisać do niego okręgi o1 i o2 oraz wywołać na takim wskaźniku metodę `Wypisz`, np.:
```
wskfig = &o1;
wskfig->Wypisz();
```
Program należy ponownie skompilować i odpalić. Co się dzieje, jeśli usuniemy słowo virtual z klasy Figura (należy przetestować i opowiedzieć prowadzącemu)?

Jakie daje nam to możliwości?
- Możemy tworzyć metody przyjmujące referencje, których zachowanie będzie zależne od typu klasy pochodnej.
- Jeśli pojawi się kiedykolwiek konieczność rozszerzenia programu o nowe obiekty, to dodanie nowej klasy będącej pochodną od już istniejącej nie wymaga tysiąca zmian w różnych miejscach programu.

Należy dodać klasę Prostokąt, zachowującą się podobnie do poprzedniej, wywołać na niej gotową już metodę `Wypisz`.

Podsumujmy:
Jeśli kompilator natrafi na **obiekt** danej klasy, np. Figura, uruchamia dla niego funkcję składową z właściwej mu klasy.
Jeśli kompilator natrafi na **wskaźnik lub referencję** klasy Figura, ma dwie możliwości wywołania funkcji składowej:
1) Skoro jest to wskaźnik do klasy Figura, wywołuje metodę składową klasy Figura. Jest to domyślne zachowanie kompilatora.
2) Kompilator widzi, że jest to wskaźnik do klasy Figura, ale zamiast na ślepo sięgać do klasy Figura *używa swojej inteligencji*: nie daje się zwieźć typem i sprawdza, na co faktycznie wskaźnik wskazuje. Wtedy może się zorientować, że wskaźnik tak naprawdę wskazuje na obiekt klasy pochodnej Okrag, zatem **orientując się według typu obiektu** uruchamia funkcję składową okręgu. Takie zachowanie wymusza słowo kluczowe virtual przed nazwą funkcji składowej.

4 Następnie **zamieniamy metodę wirtualną z klasy podstawowej na czysto wirtualną**
```
virtual void Wypisz() = 0;
```
i poprawiamy program tak, by się skompilował – czego nie możemy wtedy w programie zrobić?
Wskazówka: wszystkie metody czysto wirtualne muszą istnieć w klasach pochodnych.
Klasa, która ma co najmniej jedną funkcję czysto wirtualną nazywamy klasą **abstrakcyjną**.

Po co tworzyć klasy abstrakcyjne?
Czasem mamy jakiś obiekt, który łączy cechy kilku innych (jak np. nasza klasa Figura) ale sama nie przedstawia swoją istotną żadnego konkretnego obiektu. Mamy okręgi, prostokąty, trójkąty i inne – jak by mógł zareagować matematyk, gdybyśmy kazali policzyć pole *figury*?
Klasa abstrakcyjna jest klasą jakby nieskończoną. Jej dokończenie realizowane jest przez klasy pochodne.

Należy zwrócić jeszcze uwagę na pewną zasadę, którą należy stosować:
Jeśli klasa deklaruje jedną ze swoich funkcji jako virtual, wówczas jej destruktor deklarujemy także jako virtual. Skoro w klasie deklarujemy jakąś funkcję wirtualną, to znaczy, że na obiekty klas pochodnych zamierzamy czasem mówić jak na obiekty klasy podstawowej, co przy późniejszym niszczeniu obiektów mogłoby być problemem – nie zwalnialibyśmy pamięci dla niektórych składników klas pochodnych.

5. Należy dopisać do klasy Okrąg metody Zapisz oraz Wczytaj:

- Metoda `Zapisz(char*)` powinna zapisać do pliku, którego nazwa podana jest jako jej parametr, współrzędne środka okręgu oraz promień.

  Zapisywanie do pliku:

```
#include <fstream>

ofstream ofile;
ofile.open("file.txt");
ofile<<"aaa"<<123<<endl;
ofile.close();
```

- Zapisać dane obiektu o2 do pliku „figury.txt".

- Metodę `Wczytaj(char* a, int b)` powinna wczytać z pliku, którego nazwa podana jest jako jej parametr (`a`), z linijki o numerze podanym jako drugi parametr (`b`), współrzędne środka okręgu oraz promień, oraz przpisać je odpowiednim składnikom `Okręgu`. Przykład pliku: `figury.txt`:

```
4 5 6
2 3 8
3 2 7
```

  Wczytywanie z pliku (np. liczby):

```
#include <fstream>

fstream ifile; int val;
ifile.open("figury2.txt");
while(ifile>>val)
{
   cout<<"val: "<<val<<endl;
}
ifile.close();
```

- Wczytać do obiektu `o1` dane z drugiej linijki z przykładowego pliku „figury2.txt". Wypisać. Przykładowy plik należy stworzyć samodzielnie.

6. Należy stworzyć Makefile, umożliwiajacy kompilację napisanego programu, definiując zmienną CXX definiującą kompilator (g++) oraz CXXFLAGS definiującą flagę (-Wall), a tworzone klasy powinny być wstępnie kompilowane do plików *.o.

Należy ponadto zapoznać się z treścią pliku: http://www.if.pw.edu.pl/~majanik/data/JP/2012/makefile.pdf.