Języki programowania Wtorek, 5.12.2013, 16:15-17:45
Kolokwium Testowe, grupa A

Program wysyłamy najpóźniej w godzinie wyjścia z sali (17:45) na adres:
lgraczyk@if.pw.edu.pl
W temacie wiadomości należy napisać:
Kolokwium Testowe z JP, wtorek 5.12.2013, imię nazwisko, grupa (A lub B)
Do maila należy dołączyć wszystkie pliki potrzebne do skompilowania.
W trakcie kolokwium nie można korzystać z Internetu.

Tworzymy klasę `Figura`, która będzie zawierała następujące pola:
`std::string fNazwa;`
`int fIlosc;` - ma być zdefiniowane jako pole statyczne, początkowo deklarowane na 0
`double *Srodek;` - dynamicznie tworzona tablica 2-elementowa (element [0] – współrzędna X środka, element [1] – współrzędna Y środka)
oraz konstruktory:

- domyślny/bez parametrów – jego wywołanie ustawia pole `fNazwa` na `"Kwadrat"`, zaś oba element tablicy `Srodek` (współrzędne `X`, `Y` środka) na 1 (należy zaalokować dynamicznie pamięć tablicy na 2 elementy). Użycie konstruktora zwiększa pole statyczne o 1. W tym konstruktorze używamy listy inicjalizacyjnej.
- przyjmujący trzy parametry, ustawiające współrzędne środka `X`, `Y` (elementy tablicy `Srodek` – należy zaalokować dynamicznie pamięć tablicy na 2 elementy). W tym konstruktorze nie używamy listy inicjalizacyjnej. Użycie konstruktora zwiększa pole statyczne o 1.
- konstruktor kopiujący

destruktor:

- Destruktor zmniejsza ilośc figur o 1 oraz usuwa (zwalnia pamięć) wcześniej zaalokowaną tablicę `Srodek`

i metody „set” oraz „get” zmieniające i wypisujące wszystkie pola klasy (prócz statycznego; `SetX` oraz `GetX` ustawiają i zwracają element [0] tablicy `Srodek`, `SetY` oraz `GetY` element [1]). Pole statyczne powinno posiadać odpowiadającą metodę statyczną zwracającą to pole.
Klasa powinna mieć przeciążony `operator[]` zdefiniowany tak, by mógł stać po obu stronach znaku =. Wywołanie tego operatora na obiekcie typu `Figura` z wartością 1 element [0] tablicy `Srodek`, a z wartością 1 (i pozostałymi) element [1] tablicy `Srodek`.
Klasa powinna mieć również przeciążony `operator+`. Dodanie dwóch figur powinno zwrócić figurę gdzie współrzędne `X` oraz `Y` figury wynikowej są sumą dwóch dodawanych (czyli sumujemy elementy tablicy `Srodek`, odpowiednio [0] i [1]). Powinna mieć też nazwę ustawioną jako ciąg znaków będący połączeniem nazw dodawanych figur.
Należy również w programie stworzyć globalną funkcję (która nie jest zaprzyjaźniona z klasą) `void WypiszFigura(Figura& fig)`, która wypisuje na ekran wszystkie pola składowe figury.

Poniżej znajduje się funkcja `main`, której należy użyć:

```
int main()
{
  Figura *fig1 = new Figura();
  cout<<"Ilosc figur: "<<Figura::GetIloscFigur()<<endl;
  WypiszFigura(*fig1);
  cout<<"X srodka fig1: "<<(*fig1)[0]<<endl<<endl;

  Figura *fig2 = new Figura("Kolo",56,87);
  cout<<"Ilosc figur: "<<Figura::GetIloscFigur()<<endl;
  WypiszFigura(*fig2);
  cout<<"Y srodka fig2: "<<fig1->GetY()<<endl;

  Figura fig3 = (*fig1)+(*fig2);
  cout<<"Ilosc figur: "<<Figura::GetIloscFigur()<<endl;
  WypiszFigura(fig3);

  delete fig1;
  delete fig2;

  return 0;
}
```

Języki programowania, Wtorek, 5.12.2013, 16:15-17:45
Kolokwium Testowe, grupa B

Program wysyłamy najpóźniej w godzinie wyjścia z sali (17:45) na adres:
lgraczyk@if.pw.edu.pl
W temacie wiadomości należy napisać:
Kolokwium Testowe z JP, wtorek 5.12.2013, imię nazwisko, grupa (A lub B)
Do maila należy dołączyć wszystkie pliki potrzebne do skompilowania.
W trakcie kolokwium nie można korzystać z Internetu.

Tworzymy klasę `Point`, która jest punktem w przestrzeni dwuwymiarowej, dodatkowo przechowującym jakąś wartość (liczbę). Klasa ta będzie posiadała pola:
`int fIlosc;` - powinno być zdefiniowane jako pole statyczne i początkowo ustawione na 0
`double *Pozycja;` - dynamicznie tworzona tablica 2-elementowa (element [0] – współrzędna X, element [1] – współrzędna Y)
`string Note;`
oraz konstruktory:

- domyślny/bez parametrów – tworzy punkt mający wszystkie 2 parametry (pozycje X oraz Y) tablicy `Pozycja` ustawione na 1 (należy zaalokować dynamicznie pamięć tablicy na 2 elementy) oraz pole `Note` na `"Punkcik"`. W tym konstruktorze używamy listy inicjalizacyjnej (oczywiście na polach, dla których jest to możliwe).
- przyjmujący trzy parametry, ustawiające pola `X`, `Y` (elementy tablicy `Pozycja` – należy zaalokować dynamicznie pamięć tablicy na 2 elementy) oraz `Note`. W tym konstruktorze nie używamy listy inicjalizacyjnej. Użycie konstruktora zwiększa pole statyczne o 1.
- konstruktor kopiujący

destruktor:

- Destruktor zmniejsza ilość punktów o 1 oraz usuwa (zwalnia pamięć) wcześniej zaalokowaną tablicę `Srodek`

Należy również stworzyć metody „set” oraz „get” ustawiające i wypisujące współrzędne `X` oraz `Y` (elementy tablicy `Pozycja`; `SetX` oraz `GetX` ustawiają i zwracają element [0] tablicy `Srodek`, `SetY` oraz `GetY` element [1]) oraz pole `Note`. Pole statyczne powinno mieć swoją metodę statyczną zwracającą to pole.
Klasa powinna mieć przeciążony `operator[]` zdefiniowany tak, by mógł stać po obu stronach znaku =. Wywołanie tego operatora na obiekcie typu `Point` z wartością 0 powinno zwracać elementy tablicy `Pozycja`: pole `X`, z wartością 1 (oraz pozostałymi) pole `Y`.
Klasa powinna mieć również przeciążony `operator+`. Dodanie dwóch punktów powinni zwrócić punkt będący sumą współrzędnych obu punktów `X`, `Y` (czyli sumujemy elementy tablicy `Srodek`, odpowiednio [0] i [1]). Powinien mieć też notatkę ustawioną jako ciąg znaków będący połączeniem notatek dodawanych punktów.
W programie tworzymy ponadto funkcję zaprzyjaźnioną `void WypiszPunkt(Point& point)`, która wypisuje na ekran współrzędne punktu `X`, `Y` (czyli elementy [0] i [1] tablicy `Pozycja`) oraz `Note` danego punktu.

Poniżej znajduje się funkcja `main`, której należy użyć:

```
int main()
{

  Point *point1 = new Point();
  cout<<"Ilosc punktow: "<<Point::GetIloscPunktow()<<endl;
  WypiszPunkt(*point1);
  cout<<"punt1 x: "<<(*point1)[0]<<endl;

  Point *point2 = new Point(2,2,"Punkt");
  cout<<"Ilosc punktow: "<<Point::GetIloscPunktow()<<endl;
  WypiszPunkt(*point2);
  cout<<"point2 y: "<<point2->GetY()<<endl;

  Point point3 = (*point1)+(*point2);
  cout<<"Ilosc punktow: "<<Point::GetIloscPunktow()<<endl;
  WypiszPunkt(point3);

  delete point1;
  delete point2;

  return 0;
}
```