Programowanie II R

Zadania – seria 8.

Programowanie obiektowe – podstawy.

Zadanie 1. `circles` – Okręgi.

Niech będzie dany pewien kartezjański układ współrzędnych na płaszczyźnie. Napisz klasę `Circle` reprezentującą okrąg w tym układzie.

Zaimplementuj:

- prywatne pole `x`, przechowujące odciętą środka okręgu w danym układzie współrzędnych,
- prywatne pole `y`, przechowujące rzędną środka okręgu w danym układzie współrzędnych,
- prywatne pole `r`, przechowujące promień okręgu,
- konstruktor, przyjmujący jako argumenty odciętą i rzędną środka okręgu oraz jego promień,
- publiczną metodę `Circumference`, zwracającą obwód okręgu,
- publiczną metodę `Scale`, zwiększającą promień okręgu k-krotnie, gdzie wartość k ma być przekazywana metodzie jako argument,
- publiczną metodę `Concentric`, zwracającą wartość `true`, gdy okrąg jest współśrodkowy z innym okręgiem, reprezentowanym przez inną instancję klasy `Circle`, przekazaną metodzie `Concentric` jako argument, lub wartość `false` w przciwnym przypadku.

Korzystając z tej klasy napisz program `circles`. Program ten powinien wczytywać ze standardowego wejścia sześć liczb zmiennoprzecinkowych opisujących dwa okręgi, oznaczających kolejno: odciętą środka pierwszego okręgu, rzędną środka pierwszego okręgu, promień pierwszego okręgu, odciętą środka drugiego okręgu, rzędną środka drugiego okręgu i promień drugiego okręgu, a następnie wypisywać na standardowe wyjście obwody tych okręgów, informację o tym, czy okręgi te są współśrodkowe, oraz obwody okręgów o trzykrotnie większych promieniach.

Zadanie 2. `resistors` – Oporniki.

Napisz klasę `Resistor` reprezentującą opornik.

Zaimplementuj:

- prywatne pole `R`, przechowujące opór opornika wyrażony w omach,
- konstruktor, przyjmujący jako argument opór opornika,
- publiczną metodę `GetResistance`, zwracającą opór opornika,
- publiczną metodę `SetResistance`, zmieniającą wartość oporu opornika na przekazaną funkcji jako argument,
- funkcję zaprzyjaźnioną

```
Resistor series (const Resistor &a, const Resistor &b)
```

zwracającą nową instancję klasy `Resistor`, reprezentującą opornik o oporze równym oporowi zastępczemu szeregowego połączenia oporników reprezentowanych przez argumenty tej funkcji,

- funkcję zaprzyjaźnioną

```
Resistor parallel (const Resistor &a, const Resistor &b)
```

zwracającą nową instancję klasy `Resistor`, reprezentującą opornik o oporze równym oporowi zastępczemu równoległego połączenia oporników reprezentowanych przez argumenty tej funkcji.

Korzystając z tej klasy napisz program `resistors`. Program ten powinien wczytywać ze standardowego wejścia dwie liczby zmiennoprzecinkowe opisujące opory dwóch oporników, a następnie wypisywać na standardowe wyjście opory zastępcze szeregowego i równoległego połączenia tych oporników.

Zadanie 3. `stack` – Stos.

Napisz własną implementację stosu, czyli prostej struktury danych dopuszczającej dwie operacje: odłożenie elementu na stos oraz zdjęcie ze stosu ostatnio odłożonego elementu. Twój stos powinien przechowywać obiekty typu `int`.

W tym celu napisz strukturę `StackItem` reprezentującą element stosu. Zaimplementuj:

- pole `value` typu `int`, przechowujące wartość elementu,
- pole `lower` typu `StackItem *`, przechowujące wskaźnik na element odłożony na stos bezpośrednio wcześniej,

Napisz ponadto klasę `Stack` reprezentującą stos. Zaimplementuj:

- prywatne pole `top` typu `StackItem *`, przechowujące wskaźnik na element odłożony na stos jako ostatni,
- konstruktor bezargumentowy, przyjmujący polu `top` wartość `nullptr`,
- publiczną metodę `pop`, usuwającą ze stosu element odłożony nań jako ostatni oraz zwracającą wartość tego elementu,
- publiczną metodę `push`, odkładającą na stos element o wartości przekazanej jej jako argument.

Korzystając z tej klasy napisz program `stack`. Program ten powinien wczytywać ze standardowego wejścia liczby całkowite aż do napotkania znaku końca pliku, a następnie wypisywać na standardowe wyjście te liczby w odwrotnej kolejności.

Opracowanie: Bartłomiej Zglinicki.