Zadanie U
Stos i kolejka

Celem zadania jest zaimplementowanie trzech, niżej opisanych klas: `point`, `queue`, `stack`. Definicję klas wraz z definicjami wszystkich metod należy umieścić w pliku z rozszerzeniem `.h`. Tak przygotowany plik należy wysłać na Satori. Zostanie on skompilowany wraz z plikiem zawierającym funkcję `main`.

Do realizacji zadania, nie należy wykorzystywać szablonów z biblioteki STL (m.in. `vector`, `stack`, `queue`).

Klasa `point`

- Klasa posiada dwa pola prywatne: `int x` oraz `int y`.
- Klasa udostępnia dwa konstruktory: domyślny ustawiający wartości pól na 0 oraz konstruktor z dwoma parametrami typu `int` inicjalizującymi pola `x` i `y`.
- Klasa udostępnia trzy metody publiczne:
  - `getX()`, zwraca wartości `x`.
  - `getY()`, zwraca wartości `y`.
  - `print()`; wypisuje pole `x`, spację, pole `y` oraz znak końca linii

Klasa `queue`

Klasa `queue` jest tablicową implementacją kolejki przechowującej obiekty typu `point`. Liczba elementów, jakie kolejka może przechowywać, zależy od parametru przekazanemu konstruktorowi. Domyślny konstruktor tworzy kolejkę na maksymalnie 15 elementów.

Klasa `queue` udostępnia:

- Dwa konstruktory: domyślny oraz konstruktor z parametrem typu `int` (oznaczającym maksymalny rozmiar kolejki).
- Metodę `enqueue(point& x)` – dodaje obiekt x do kolejki. Jeśli kolejka jest pełna, funkcja nie robi nic.
- Metodę `dequeue()` – wyjmuje pierwszy obiekt x z kolejki i zwraca jego wartość. Jeśli kolejka jest pusta, funkcja zwraca punkt $(0, 0)$.
- Metodę `front()` – zwraca wartość elementu pierwszego w kolejce. Jeśli kolejka jest pusta, funkcja zwraca punkt $(0, 0)$.
- Metodę `empty()` – zwraca `true`, jeśli kolejka jest pusta.
- Metodę `full()` – zwraca `true`, jeśli kolejka jest pełna.

- Metodę `clear()` – czyści kolejkę.
- Metodę `resize(int n)` – zwiększa liczbę elementów jakie kolejka może przechować. Jeśli `n` jest mniejsze od dotychczasowego rozmiaru kolejki, funkcja nie robi nic.
- Destruktor.

Klasa `stack`

Klasa `stack` implementuje stos przechowujący duże liczby całkowite (typu `long long`) wykorzystując do tego tablicę. Liczba elementów, jakie stos może przechowywać, zależy od parametru przekazanemu konstruktorowi. Domyślny konstruktor tworzy stos na maksymalnie 15 elementów.

Klasa `stack` udostępnia:

- Dwa konstruktory: domyślny oraz konstruktor z parametrem typu `int` (oznaczającym maksymalny rozmiar stosu).
- Metodę `push(long long &x)` – wkłada element x stos. Jeśli stos jest pełny, funkcja nie robi nic.
- Metodę `pop()` – wyjmuje element x ze szczytu stosu i zwraca jego wartość. Jeśli stos jest pusty, funkcja zwraca wartość 0.
- Metodę `top()` – zwraca wartość elementu ze szczytu stosu. Jeśli stos jest pusty, funkcja zwraca wartość 0.
- Metodę `empty()` – zwraca `true`, jeśli stos jest pusty.
- Metodę `full()` – zwraca `true`, jeśli stos jest pełny.
- Metodę `clear()` – czyści stos.
- Metodę `resize(int n)` – zwiększa liczbę elementów jakie stos może przechować. Jeśli `n` jest mniejsze od dotychczasowego rozmiaru stosu, funkcja nie robi nic.
- Destruktor.

Przykładowy plik z funkcją `main`:

```
#include<iostream>
using namespace std;
#include "solution.h"

int main()
{
```

```
ios_base::sync_with_stdio(false);

point A;
point B(5,8);
cout << A.getX() << " " << A.getY() << endl;
B.print();

stack S1;
for (long long  i=0; i<20; ++i) S1.push(i);
cout << S1.top() << " ";
if (S1.full()) cout << "tak" << endl; else cout << "nie" << endl;

S1.resize(50);
cout << S1.pop() << " ";
S1.clear();
if (S1.empty()) cout << "tak" << endl; else cout << "nie" << endl;
for (long long  i=20; i<30; ++i) S1.push(i);
cout << S1.top() << " ";
if (S1.full()) cout << "tak" << endl; else cout << "nie" << endl;

stack S2(100);
for (long long  i=0; i<30; ++i) S2.push(i);
for (int i=0; i<10; ++i) cout << S2.pop() << " " ;
cout << endl;

queue Q1;
Q1.dequeue();
Q1.front();
for (int i=1; i<20; ++i) { point p(i,i); Q1.enqueue( p ); };

cout << Q1.front().getX() << " ";
if (Q1.full()) cout << "tak" << endl; else cout << "nie" << endl;

for (int i=0; i<10; ++i) {
   point a = Q1.dequeue(); cout << a.getX()<< " " << a.getY()<< " ";
}
for (int i=30; i<39; ++i) { point p(i,i); Q1.enqueue( p ); };
if (Q1.full()) cout << "tak" << endl; else cout << "nie" << endl;
cout << Q1.front().getY() << " " << endl;

queue Q2(100);
if (Q2.empty()) cout << "tak" << endl; else cout << "nie" << endl;
for (int i=1; i<51; ++i) { point p(i,i); Q2.enqueue( p ); };
```

```
   for (int i=0; i<10; ++i) {
      point a = Q2.dequeue(); cout << a.getX()<< " " << a.getY()<< " ";
   }
   return 0;
}
```

Wynik działania powyższej funkcji `main`:

```
0 0
5 8
14 tak
14 tak
29 nie
29 28 27 26 25 24 23 22 21 20
1 tak
1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 10 nie
11
tak
1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 10
```