Kolejki

[std::queue - reference.com]

Kolejka jest to struktura danych, która działa według zasady "first in, first out" (FIFO), czyli pierwszy element dodany do kolejki będzie również pierwszy do wyjścia.

Implementacja Kolejki za Pomocą Klasy queue z Biblioteki Standardowej

1. Inkludowanie Biblioteki:

```
include <iostream>
include <queue> // Do korzystania z klasy queue
```

2. Deklaracja Kolejki:

```
std::queue<int> mojaKolejka;
```

3. Dodawanie Elementów do Kolejki:

```
mojaKolejka.push(1); // Dodanie elementu na koniec kolejki
mojaKolejka.push(2);
mojaKolejka.push(3);
```

4. Usuwanie Elementów z Kolejki:

```
mojaKolejka.pop(); // Usunięcie pierwszego elementu z kolejki
```

5. Sprawdzanie Rozmiaru Kolejki:

```
int rozmiar = mojaKolejka.size(); // Zwraca liczbę elementów w kolejce
```

6. Sprawdzanie, Czy Kolejka Jest Pusta:

```
bool czyPusta = mojaKolejka.empty(); // Zwraca true, jeśli kolejka jest pusta,
a false w przeciwnym przypadku
```

7. Odczytywanie Elementu Z Góry Kolejki (Bez Usuwania):

```
int pierwszyElement = mojaKolejka.front(); // Zwraca wartość pierwszego
elementu w kolejce
```

8. Odczytywanie Elementu Z Końca Kolejki (Bez Usuwania):

```
int ostatniElement = mojaKolejka.back(); // Zwraca wartość ostatniego
elementu w kolejce
```

Implementacja obiektowa

```
include <iostream>

// Definicja struktury węzła kolejki
struct Node {
    int data;
    Node* next;

    Node(int d) : data(d), next(nullptr) {} // Konstruktor węzła
};

// Definicja klasy Kolejki
class Queue {
private:
    Node* front; // Wskaźnik na początek kolejki
    Node* rear;  // Wskaźnik na koniec kolejki

public:
    Queue(); // Konstruktor kolejki
    ~Queue(); // Destruktor kolejki
    void enqueue(int data); // Dodawanie elementu do kolejki
    void dequeue(); // Usuwanie elementu z kolejki
    int peek(); // Podglądanie pierwszego elementu kolejki
    bool isEmpty(); // Sprawdzanie, czy kolejka jest pusta
};

// Konstruktor kolejki
Queue::Queue() : front(nullptr), rear(nullptr) {}

// Destruktor kolejki
Queue::~Queue() {
    dequeue(); // Usuwamy wszystkie elementy kolejki

}

// Dodawanie elementu do kolejki
void Queue::enqueue(int data) {
    Node* newNode = new Node(data); // Tworzymy nowy węzeł

    if (rear == nullptr) { // Jeśli kolejka jest pusta
        front = rear = newNode; // Nowy węzeł staje się zarówno
początkiem, jak i końcem kolejki
    } else {
        rear->next = newNode; // Ustawiamy następnik ostatniego
elementu kolejki na nowy węzeł
        rear = newNode; // Nowy węzeł staje się nowym końcem kolejki
    }
}

// Usuwanie elementu z kolejki
void Queue::dequeue() {
    Node* temp = front; // Zapamiętujemy wskaźnik na pierwszy
element
    front = front->next; // Przesuwamy wskaźnik na początek kolejki
    delete temp; // Usuwamy poprzedni pierwszy element
```

```
    if (front == nullptr) { // Jeśli kolejka jest teraz pusta
        rear = nullptr; // Zerujemy również wskaźnik na koniec
kolejki
    }
}

int main() {
    Queue myQueue;

    myQueue.enqueue(10);
    myQueue.enqueue(20);
    myQueue.enqueue(30);

 //   std::cout << "Pierwszy element kolejki: " << myQueue.peek() <<
std::endl;

    myQueue.dequeue();
//    std::cout << "Pierwszy element kolejki po usunięciu: " <<
myQueue.peek() << std::endl;

    myQueue.dequeue();
    myQueue.dequeue();

 //   if (myQueue.isEmpty()) {
 //       std::cout << "Kolejka jest pusta.\n";
 //   } else {
//        std::cout << "Kolejka nie jest pusta.\n";
 //   }

    return 0;
}
```

Zadania:

Do przykładu implementacja obiektowa wykonaj:

1. Uzupełnij metodę peek():
    a. Napisz metodę peek(), która będzie zwracała wartość pola data pierwszego elementu kolejki, bez usuwania tego elementu.
    b. Sprawdź, czy wskaźnik front nie jest równy nullptr, jeśli tak, zwróć wartość pola data pierwszego elementu.
    c. W przeciwnym razie, wypisz komunikat informujący o tym, że kolejka jest pusta.
2. Uzupełnij metodę isEmpty():
    a. Napisz metodę isEmpty(), która będzie sprawdzała, czy kolejka jest pusta.
    b. Jeśli wskaźnik front jest równy nullptr, to kolejka jest pusta, więc zwróć true.
    c. W przeciwnym razie, kolejka nie jest pusta, więc zwróć false.
3. Dodaj sprawdzanie czy kolejka jest pusta w metodzie dequeue():
    a. Przed usunięciem pierwszego elementu, sprawdź, czy kolejka nie jest pusta.
    b. Jeśli kolejka jest pusta (czyli front jest równy nullptr), wypisz komunikat informujący o tym, że kolejka jest pusta i nie można usunąć elementu.
    c. W przeciwnym razie, usuń pierwszy element z kolejki i przesuń wskaźnik front na następny element.
4. Usuń za komentowane linie w funkcji main():

a. Odkomentuj linie w funkcji main(), które wywołują metody peek() oraz sprawdzają, czy kolejka jest pusta.