პროგრამირების აბსტრაქციები

სემინარის ამოცანები #19

deque კლასი

თქვენი ამოცანაა დაწეროთ სტეკის და რიგის ფუნქციონალობის გამაერთიანებელი კლასი ე.წ. "deque". ამ კლასის ობიექტებში უნდა შეგვეძლოს დასაწყისშიც და ბოლოშიც ელემენტის ჩამატებაც და ამოღებაც. ოთხივე ოპერაცია $\mathcal{O}(1)$ დროში უნდა მუშაობდეს. ამ სტრუქტურის გაკეთება შესაძლებელია უწყვეტი მასივითაც და ორმხრივად ბმული სიითაც. ჩვენ ორმხრივად ბმული სიით გავაკეთებთ. ასევე, დასაწყისშიც და ბოლოშიც გამოვიყენებთ ზედმეტ უჯრას რომელიც არცერთ ელემენტს არ შეესაბამება, უბრალოდ კოდს გვიმარტივებს იმითი რომ ჩამატების და ამოღების ოპერაციებს კერძო შემთხვევები არ ექნებათ როცა "deque" ცარიელი ან ერთ ელემენტიანია. "deque" კლასი უნდა მუშაობდეს `int` და `double` ტიპის ელემენტებისთვის.

header file

```
#ifndef DEQUE_H_
#define DEQUE_H_

template <typename T>
class Deque {
private:
    struct Node {
        T value;
        Node* next;
        Node* prev;
    };
    Node* head;
    Node* tail;
public:
    Deque();
    ~Deque();
    bool isEmpty();
    void pushFront(T value);
    void pushBack(T value);
    T popFront();
    T popBack();
};

#endif
```

source file (კონსტრუქტორი, დესტრუქტორი და `isEmpty()` ფუნქციები)

```cpp
#include "error.h"
#include "Deque.h"

template <typename T>
Deque<T>::Deque() {
    head = new Node;
    tail = new Node;
    head -> next = tail;
    head -> prev = NULL;
    tail -> prev = head;
    tail -> next = NULL;
}

template <typename T>
Deque<T>::~Deque() {
    Node* ptr = head;
    while (ptr != NULL) {
        Node* tmp = ptr;
        ptr = ptr -> next;
        delete tmp;
    }
}

template <typename T>
bool Deque<T>::isEmpty() {
    return (head -> next == tail);
}
```

source file (ელემენტის ჩამატების და ამოღების ოპერაციები)

```
template <typename T>
void Deque<T>::pushFront(T value) {
    Node* newNode = new Node;
    newNode -> value = value;
    newNode -> next = head -> next;
    newNode -> prev = head;
    head -> next = newNode;
    newNode -> next -> prev = newNode;
}

template <typename T>
void Deque<T>::pushBack(T value) {
    Node* newNode = new Node;
    newNode -> value = value;
    newNode -> next = tail;
    newNode -> prev = tail -> prev;
    tail -> prev = newNode;
    newNode -> prev -> next = newNode;
}

template <typename T>
T Deque<T>::popFront() {
    if (isEmpty())
        error("Attempted to pop empty deque");
    Node* tmp = head -> next;
    head -> next = tmp -> next;
    tmp -> next -> prev = head;
    T result = tmp -> value;
    delete tmp;
    return result;
}

template <typename T>
T Deque<T>::popBack() {
    if (isEmpty())
        error("Attempted to pop empty deque");
    Node* tmp = tail -> prev;
    tail -> prev = tmp -> prev;
    tmp -> prev -> next = tail;
    T result = tmp -> value;
    delete tmp;
    return result;
}

template class Deque<int>;
template class Deque<double>;
```