**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра CАПР**

отчет

**по лабораторной работе №1**

**по дисциплине «Алгоритмы и структуры данных»**

**Вариант 2**

**Тема: Списки. Работа с unit-тестами**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9302 |  | Бабкин И. А. |
| Преподаватель |  | Тутуева.А.В |

Санкт-Петербург

2020

**Постановка задачи**

Необходимо реализовать двусвязный список при помощи классов с нужным набором методов. Реализовать unit-тесты ко всем реализуемым методам

**Описание реализуемого класса и методов**

Было создано два класса: для хранения элемента и адреса на следующий и предыдущий элементы (class Node) и класс списка (class LinkedList)

|  |  |
| --- | --- |
| Название метода | Описание |
| reset\_list | Обнуляет список |
| add\_first | Добавляет первый элемент списка |
| push\_back | Вставляет элемент в конец списка |
| push\_front | Вставляет элемент в начало списка |
| pop\_back | Удаляет последний элемент списка |
| pop\_front | Удаляет первый элемент списка |
| insert | Вставляет элемент перед элементом с заданным индексом |
| at | Возвращает значение элемента с заданным индексом |
| remove | Удаляет элемент с заданным индексом |
| get\_size | Возвращает размер списка (количество элементов) |
| print\_to\_console | Выводит список в консоль |
| clear | Удаляет все элементы списка |
| set | Заменяет элемент по индексу на передаваемый элемент |
| isEmpty | Проверяет список на наличие пустоты |
| reverse | Переворачивает список |

**Оценка временной сложности методов**

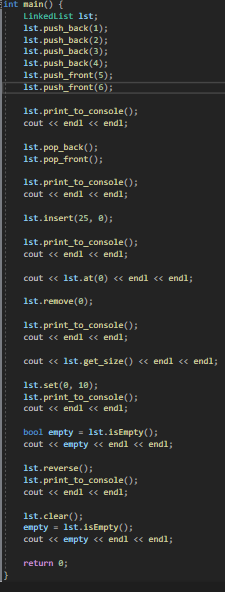
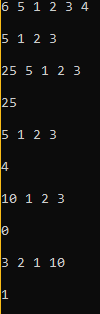
|  |  |
| --- | --- |
| Название метода | Оценка временной сложности |
| reset\_list | O(1) |
| add\_first | O(1) |
| push\_back | O(1) |
| push\_front | O(1) |
| pop\_back | O(1) |
| pop\_front | O(1) |
| insert | O(n) |
| at | O(n) |
| remove | O(n) |
| get\_size | O(1) |
| print\_to\_console | O(n) |
| clear | O(n) |
| set | O(n) |
| isEmpty | O(1) |
| reverse | O(n) |

**Описание реализованных unit-тестов**

Для проверки реализованных методов были написаны unit-тесты. В таблице ниже представлены их названия. Названия совпадают с названиями проверяемых методов.

|  |
| --- |
| Название теста |
| push\_back1 |
| push\_back2 |
| push\_ front1 |
| push\_ front2 |
| pop\_back\_front |
| insert1 |
| insert2 |
| insert3 |
| remove1 |
| remove2 |
| clear |
| set |
| isEmpty |
| reverse |

**Пример работы**



**Листинг**

**LinkedList.h**

#pragma once

class LinkedList

{

private:

void add\_first(int newElem); //

void reset\_list(); //

class Node

{

public:

Node(int data, Node\* next = nullptr, Node\* prev = nullptr) {

this->data = data;

this->next = next;

this->prev = prev;

}

~Node() {}

int data;

Node\* next;

Node\* prev;

};

Node\* head;

Node\* tail;

size\_t size;

public:

LinkedList();

~LinkedList();

void push\_back(int newElem); //

void push\_front(int newElem); //

void pop\_back(); //

void pop\_front(); //

void insert(int newElem, size\_t index); //

int at(size\_t index); //

void remove(size\_t index); //

size\_t get\_size(); //

void print\_to\_console(); //

void clear(); //

void set(size\_t, int); //

bool isEmpty(); //

void reverse(); //

};

**LinkedList.cpp**

#include <stdexcept>

#include "iostream"

#include "LinkedList.h"

using namespace std;

void LinkedList::reset\_list()

{

head = nullptr;

tail = nullptr;

}

LinkedList::LinkedList()

{

reset\_list();

size = 0;

}

LinkedList::~LinkedList()

{

}

void LinkedList::add\_first(int newElem)

{

head = new Node(newElem);

tail = head;

}

void LinkedList::push\_back(int newElem)

{

if (size == 0) {

add\_first(newElem);

}

else {

tail->next = new Node(newElem, nullptr, tail);

tail = tail->next;

}

size++;

}

void LinkedList::push\_front(int newElem)

{

if (size == 0) {

add\_first(newElem);

}

else {

head = new Node(newElem, head);

}

head->next->prev = head;

size++;

}

void LinkedList::pop\_back()

{

if (size == 0) return;

if (size == 1) {

delete head;

reset\_list();

}

else {

Node\* current = tail->prev;

current->next = nullptr;

delete tail;

tail = current;

}

size--;

}

void LinkedList::pop\_front()

{

if (size == 0) return;

if (size == 1) {

delete head;

reset\_list();

}

else {

Node\* current = head->next;

current->prev = nullptr;

delete head;

head = current;

}

size--;

}

void LinkedList::insert(int newElem, size\_t index)

{

if (index >= size) {

throw out\_of\_range("Index is greater than list size");

}

else if (index == 0) {

push\_front(newElem);

}

else {

Node\* current = head;

size\_t counter = 0;

while (counter != index - 1) {

current = current->next;

counter++;

}

current->next = new Node(newElem, current->next, current);

current = current->next;

current->next->prev = current;

current->prev->next = current;

size++;

}

}

int LinkedList::at(size\_t index)

{

if (index >= size) {

throw out\_of\_range("Index is greater than list size");

}

else {

Node\* current = head;

size\_t counter = 0;

while (counter != index) {

current = current->next;

counter++;

}

return current->data;

}

}

void LinkedList::remove(size\_t index)

{

if (index >= size) {

throw out\_of\_range("Index is greater than list size");

}

else if (index == 0) {

pop\_front();

}

else if (index == size - 1) {

pop\_back();

}

else {

Node\* current = head;

size\_t counter = 0;

while (counter != index) {

current = current->next;

counter++;

}

current->prev->next = current->next;

current->next->prev = current->prev;

delete current;

size--;

}

}

size\_t LinkedList::get\_size()

{

return size;

}

void LinkedList::print\_to\_console()

{

Node\* current = head;

while (current->next != nullptr) {

cout << current->data << " ";

current = current->next;

}

cout << current->data << " ";

}

void LinkedList::clear()

{

Node\* current = head->next;

while (current->next != nullptr) {

delete head;

head = current;

current = head->next;

}

delete head;

delete tail;

size = 0;

}

void LinkedList::set(size\_t index, int newElem)

{

if (index >= size) {

throw out\_of\_range("Index is greater than list size");

}

Node\* current = head;

size\_t counter = 0;

while (counter != index) {

current = current->next;

counter++;

}

current->data = newElem;

}

bool LinkedList::isEmpty() // if empty - true, else false

{

return !size;

}

void LinkedList::reverse()

{

Node\* current = head;

Node\* temp;

for (size\_t i = 0; i < size; i++) {

temp = current->prev;

current->prev = current->next;

current->next = temp;

current = current->prev;

}

temp = head;

head = tail;

tail = temp;

}