**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 3**

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

|  |  |  |
| --- | --- | --- |
| Студент гр. 9302 |  | Баязитов О.О. |
| Преподаватель |  | Тутуева.А.В |

Санкт-Петербург

2020

**Постановка задачи**

Необходимо реализовать односвязный 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 |
| insertlist | Вставка другого списка в список, начиная с индекса |

**Оценка временной сложности методов**

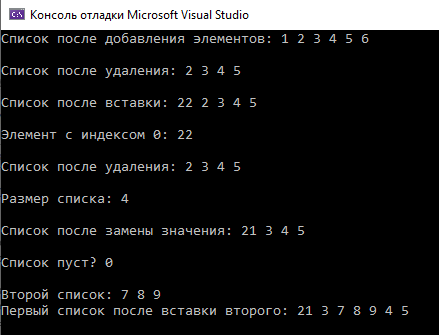
|  |  |
| --- | --- |
| Название метода | Оценка временной сложности |
| reset\_list | O(1) |
| add\_first | O(1) |
| push\_back | O(1) |
| push\_front | O(1) |
| pop\_back | O(1) |
| pop\_front | O(1) |
| insert | O(n) |
| at | O(n) |
| remove | O(n) |
| get\_size | O(1) |
| print\_to\_console | O(n) |
| clear | O(n) |
| set | O(n) |
| isEmpty | O(1) |
| insertlist | O(n^2) |

**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\_back\_1 |
| push\_back\_2 |
| push\_ front\_1 |
| push\_ front\_2 |
| pop\_back\_front |
| insert\_1 |
| insert\_2 |
| insert\_3 |
| insertlist\_1 |
| insertlist\_2 |
| insertlist\_3 |
| remove\_1 |
| remove\_2 |
| clear |
| set |
| isEmpty |

**Пример работы**



**Листинг**

**Main.cpp**

#include <iostream>

#include "LinkedList.h"

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

LinkedList lst;

lst.push\_back(3);

lst.push\_back(4);

lst.push\_back(5);

lst.push\_back(6);

lst.push\_front(2);

lst.push\_front(1);

cout << "Список после добавления элементов: ";

lst.print\_to\_console();

cout << endl << endl;

lst.pop\_back();

lst.pop\_front();

cout << "Список после удаления: ";

lst.print\_to\_console();

cout << endl << endl;

lst.insert(22, 0);

cout << "Список после вставки: ";

lst.print\_to\_console();

cout << endl << endl;

cout << "Элемент с индексом 0: ";

cout << lst.at(0) << endl << endl;

lst.remove(0);

cout << "Список после удаления: ";

lst.print\_to\_console();

cout << endl << endl;

cout << "Размер списка: " << lst.get\_size() << endl << endl;

lst.set(0, 21);

cout << "Список после замены значения: ";

lst.print\_to\_console();

cout << endl << endl;

bool empty = lst.isEmpty();

cout << "Список пуст? " << empty << endl << endl;

LinkedList Slist;

Slist.push\_back(7);

Slist.push\_back(8);

Slist.push\_back(9);

cout << "Второй список: ";

Slist.print\_to\_console();

cout << endl;

lst.insertlist(Slist, 2);

cout << "Первый список после вставки второго: ";

lst.print\_to\_console();

cout << endl << endl;

return 0;

}

**LinkedList.h**

#pragma once

class LinkedList

{

private:

void add\_first(int newElem);

void reset\_list();

class Node

{

public:

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

this->data = data;

this->next = next;

}

~Node() {}

int data;

Node\* next;

};

Node\* head;

Node\* tail;

size\_t size;

public:

LinkedList();

~LinkedList();

void push\_back(int newElem); //add in the end

void push\_front(int newElem); //add in the origin

void pop\_back(); //delete last

void pop\_front(); //delete first

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

int at(size\_t index); //returns data of the index element

void remove(size\_t index); //delete index element

size\_t get\_size(); //returns size of the list

void print\_to\_console();

void clear(); //clear list

void set(size\_t, int);

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

void insertlist(LinkedList&, size\_t);//insert list into the list, starting at the index

//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()

{

if (!isEmpty())//if the list is already empty, nothing happens

{

clear();

}

}

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->next;

}

size++;

}

void LinkedList::push\_front(int newElem)

{

if (size == 0) {

add\_first(newElem);

}

else {

head = new Node(newElem, head);

}

size++;

}

void LinkedList::pop\_back()

{

if (size == 0) return;

if (size == 1) {

delete head;

reset\_list();

}

else {

Node\* current = head;

while (current->next != tail) {

current = current->next;

}

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;

delete head;

head = current;

}

size--;

}

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

{

if (index >= size) {

throw out\_of\_range("out of range");

}

else if (index == 0) {

push\_front(newElem);

}

else {

Node\* current = head;

Node\* previous = head;

size\_t counter = 0;

while (counter != index - 1) {

current = current->next;

counter++;

if (counter > 0)

{

previous = previous->next;

}

}

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

/\*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;

Node\* previous = head;

size\_t counter = 0;

while (counter != index) {

current = current->next;

if (counter > 0)

{

previous = previous->next;

}

counter++;

}

/\*current->prev->next = current->next;

current->next->prev = current->prev;\*/

previous->next = current->next;

delete current;

size--;

}

}

size\_t LinkedList::get\_size()

{

return size;

}

void LinkedList::print\_to\_console()

{

if (!isEmpty())

{

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::insertlist(LinkedList& SecList, size\_t index)//I didn't write a copy constructor, so I pass the list like this

{

Node\* curr = SecList.head;

while (curr->next != nullptr) {

insert(curr->data, index);

index++;

curr = curr->next;

}

insert(curr->data, index);

}