**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе №1**

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

## Вариант №15

Студент гр. 9309 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Юшин Е.В.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тутуева А.В.

Санкт-Петербург

2021

## Цель работы

Изучить способы представления линейных структур данных с помощью классов.

## Постановка задачи:

Реализовать класс связного списка с набором методов. Данные, хранящиеся в списке могут быть любого типа на ваш выбор.

## Описание алгоритмов:

## Описание классов:

**-List**

Класс списка. Состоит из указателей на первый и последний элементы, значения размера.

**-ListElem**   
Класс элемента списка. Состоит из информационного наполнения и указателя на следующий элемент.

 Описание методов:

* -push\_back  
  Вставляет новый элемент в конец списка. Работает за О(1).
* -push\_front  
  Вставляет новый элемент в начало списка. Работает за О(1).
* -pop\_back  
  Удаляет элемент с конца списка, меняя при этом значение указателя last. Меняет значение head, если список состоял из одного элемента. Выдаёт исключение, если список пуст. Работает за O(n).
* -pop\_front  
  Удаляет элемент с начала списка, перезаписывает указатель head. Меняет значение last, если список состоял из одного элемента. Выдаёт исключение, если список пуст. Работает за О(1).
* -insert  
  Вставляет элемент на позицию входящего индекса. Элемент, ранее стоявший на этой позиции, перемещается вправо. Если вставка на нулевой индекс , то используется push\_front. Выдаёт исключение, когда входящий индекс больше размера списка. Работает за O(n).
* -at  
  Возвращает значение элемента под входящим индексом. Выдаёт исключение, если индекс больше или равен размеру списка. Работает за O(n).
* -**remove**  
  Удаляет элемент с заданным индексом из списка. Работает через pop\_back(при удалении из конца) или pop\_front(для удаления из начала) Выдаёт исключение при выходе за пределы списка. Работает за O(n).
* -**get\_size**

Возвращает значение размера списка. Работает за O(1).

* -print\_to\_console  
  Выводит элементы списка на экран. Выдаёт исключение, если список пуст. Работает за O(n).
* -**isEmpty**  
  Возвращает true, если список пустой. Работает за O(1).
* -**clear**  
  Вызывает pop\_front до тех пор, пока список не окажется пуст. Работает за O(n).  
  -**set**
* Выдаёт исключение при выходе за пределы списка. Меняет элемент с входящим индексом на значение входящего элемента. Работает через последовательный вызов insert и remove. Работает за О(n).  
  -**swap**
* Выдаёт исключение если индекс любого из двух элементов выходит за границы списка. Меняет местами значения двух элементов списка по индексу. В лучшем случае работает за O(1), в худшем за О(n).

## Код юнит-теста:

#include "pch.h"

#include "CppUnitTest.h"

#include "../L1/L1.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1

{

TEST\_CLASS(UnitTest1)

{

public:

TEST\_METHOD(push\_back)

{

List L;

L.push\_back(1);

L.push\_back(2);

Assert::AreEqual(L.at(0), 1);

Assert::AreEqual(L.at(1), 2);

}

TEST\_METHOD(push\_front)

{

List L;

L.push\_front(1);

L.push\_front(2);

Assert::AreEqual(L.at(0), 2);

Assert::AreEqual(L.at(1), 1);

}

TEST\_METHOD(pop\_back)

{

List L;

L.push\_back(1);

L.push\_back(2);

L.pop\_back();

Assert::AreEqual(L.at(0), 1);

L.pop\_back();

Assert::IsTrue(L.isEmpty());

try

{

L.pop\_back();

Assert::IsTrue(0);

}

catch(exception&exception){}

}

TEST\_METHOD(pop\_front)

{

List L;

L.push\_back(1);

L.push\_back(2);

L.pop\_front();

Assert::AreEqual(L.at(0), 2);

L.pop\_front();

Assert::IsTrue(L.isEmpty());

try

{

L.pop\_front();

Assert::IsTrue(0);

}

catch (exception& exception) {}

}

TEST\_METHOD(insert)

{

List L;

L.push\_back(1);

L.push\_back(3);

L.insert(1, 2);

L.insert(0, 0);

L.insert(4, 4);

Assert::AreEqual(L.at(0), 0);

Assert::AreEqual(L.at(1), 1);

Assert::AreEqual(L.at(2), 2);

Assert::AreEqual(L.at(3), 3);

Assert::AreEqual(L.at(4), 4);

try

{

L.insert(10, 10);

Assert::IsTrue(0);

}

catch (exception& exception) {}

}

TEST\_METHOD(remove)

{

List L;

L.push\_back(1);

L.push\_back(2);

L.push\_back(3);

L.remove(1);

Assert::AreEqual(L.at(0), 1);

Assert::AreEqual(L.at(1), 3);

try

{

L.remove(10);

Assert::IsTrue(0);

}

catch (exception& exception) {}

}

TEST\_METHOD(clear)

{

List L;

L.push\_back(1);

L.push\_back(2);

L.push\_back(3);

L.push\_back(4);

L.clear();

Assert::IsTrue(L.isEmpty());

}

TEST\_METHOD(set)

{

List L;

L.push\_back(1);

L.push\_back(3);

L.insert(1, 2);

L.set(1, 4);

Assert::AreEqual(L.at(0), 1);

Assert::AreEqual(L.at(1), 4);

Assert::AreEqual(L.at(2), 3);

try

{

L.insert(10, 10);

Assert::IsTrue(0);

}

catch (exception& exception) {}

}

TEST\_METHOD(swap)

{

List L;

L.push\_back(1);

L.push\_back(2);

L.push\_back(3);

L.push\_back(4);

L.swap(1, 3);

Assert::AreEqual(L.at(0), 1);

Assert::AreEqual(L.at(1), 4);

Assert::AreEqual(L.at(2), 3);

Assert::AreEqual(L.at(3), 2);

try

{

L.swap(10, 11);

Assert::IsTrue(0);

}

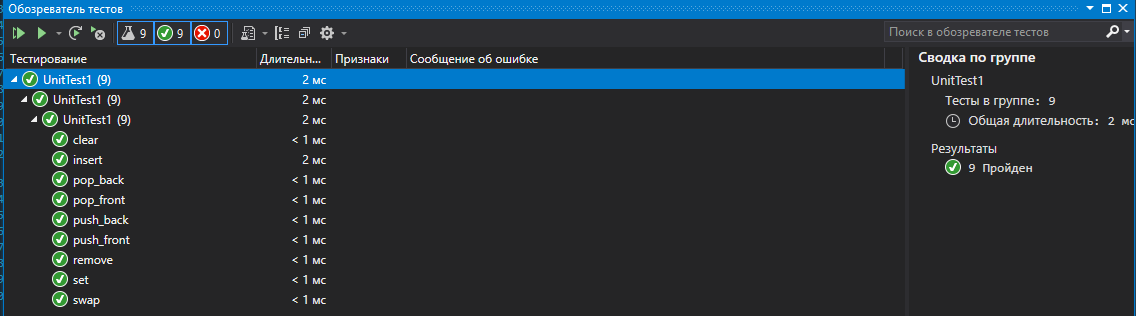
catch (exception& exception) {}

}

};

}

## Результат выполнения unit-тестов:



## Код программы:

#include <iostream>

#include <stdexcept>

#include<clocale>

using namespace std;

class ListElem

{

public:

int inf;

ListElem\* next;

ListElem(int inform)

{

inf = inform;

next = NULL;

}

};

class List

{

ListElem\* first;

ListElem\* last;

size\_t size;

ListElem\* find(size\_t index)

{

if (index == 0)

{

return first;

}

else if (index == (size - 1))

{

return last;

}

else

{

ListElem\* Search = first;

for (int i = 0;i < index;i++)

{

Search = Search->next;

}

return Search;

}

}

public:

List()

{

first = NULL;

last = first;

size = 0;

}

void push\_back(int inf)

{

ListElem\* New = new ListElem(inf);

if (last == NULL)

{

last = New;

first = last;

}

else

{

last->next = New;

last = last->next;

}

size++;

}

void push\_front(int inf)

{

ListElem\* New = new ListElem(inf);

if (last == NULL)

{

last = New;

first = last;

}

else

{

New->next = first;

first = New;

}

size++;

}

void pop\_back()

{

if (size != 0)

{

if (size == 1)

{

delete last;

first = NULL;

last = first;

}

else

{

ListElem\* newlast = first;

while (newlast->next != last)

newlast = newlast->next;

delete last;

last = newlast;

}

size--;

}

else

{

throw runtime\_error("Ошибка: список пуст");

}

}

void pop\_front()

{

if (size != 0)

{

if (size == 1)

{

delete last;

first = NULL;

last = first;

}

else

{

ListElem\* newfirst = first->next;

delete first;

first = newfirst;

}

size--;

}

else

{

throw runtime\_error("Ошибка: список пуст");

}

}

void insert(size\_t index, int inf)

{

if (index == 0)

{

push\_front(inf);

}

else if (index == size)

{

push\_back(inf);

}

else if (index < size)

{

ListElem\* New = new ListElem(inf);

ListElem\* Prev = first;

ListElem\* Next = Prev->next;

for (int i = 0;i < index-1;i++)

{

Prev = Prev->next;

Next = Prev->next;

}

Prev->next = New;

New->next = Next;

size++;

}

else

{

throw out\_of\_range("Ошибка: элемента с таким индексом не существует");

}

}

int at(size\_t index)

{

if (index < size)

{

return find(index)->inf;

}

else

{

throw out\_of\_range("Ошибка: элемента с таким индексом не существует");

}

}

void remove(size\_t index)

{

if (index >= size)

{

throw out\_of\_range("Ошибка: элемента с таким индексом не существует");

}

else if (index == (size - 1))

{

pop\_back();

}

else if (index == 0)

{

pop\_front();

}

else

{

ListElem\* Prev = find(index - 1);

ListElem\* Remove = Prev->next;

ListElem\* Next = Remove->next;

Prev->next = Next;

delete Remove;

size--;

}

}

size\_t get\_size()

{

return size;

}

void print\_to\_console()

{

ListElem\* Cur = first;

while (Cur != NULL)

{

cout << Cur->inf << ' ';

Cur = Cur->next;

}

cout << endl;

}

bool isEmpty()

{

if (first == NULL)

return true;

else

return false;

}

void clear()

{

while (!isEmpty())

pop\_front();

}

void set(size\_t index, int inf)

{

if (index >= size)

{

throw out\_of\_range("Ошибка: элемента с таким индексом не существует");

}

else

{

find(index)->inf = inf;

}

}

void swap(size\_t index1, size\_t index2)

{

if ((index1 >= size) || (index2 >= size))

{

throw out\_of\_range("Ошибка: элемента с таким индексом не существует");

}

else

{

int temp;

ListElem\* E1 = find(index1);

ListElem\* E2 = find(index2);

temp = E1->inf;

E1->inf = E2->inf;

E2->inf = temp;

}

}

~List()

{

clear();

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

List L;

L.push\_back(1);

L.push\_back(2);

L.push\_back(3);

L.push\_back(4);

L.push\_back(5);

L.print\_to\_console();

L.swap(2, 4);

L.print\_to\_console();

try

{

L.swap(10, 11);

}

catch (exception& exception)

{

cout << exception.what() << endl;

}

}

## Результат работы программы:

