**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 8**

**Тема: Создание списка. Работа с UNIT-Тестированием**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9302 |  | Точилин А.Е. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2020

### Постановка задачи

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

### Описание реализуемого класса и методов. Оценка временной сложности методов.

Для реализации было создано два класса. Class Node, для хранения узла, Код данного класса представлен в приложении А. Class List, для хранения всего списка, описание методов представлено в табл. 1. Код данного класса представлен в приложении Б.

Таблица 1 – Описание и оценка сложности методов класса List

|  |  |  |
| --- | --- | --- |
| Название метода | Описание | Оценка временной сложности |
| void push\_back(int elem); | Добавление элемента в конец списка | О(1) |
| void push\_front(int elem); | Добавление элемента в начало списка | O(1) |
| void pop\_back(); | Удаление элемента из конца списка | O(1) |
| void pop\_front(); | Удаление элемента из начала списка | O(1) |
| void insert(int, size\_t); | Вставка элемента по индексу | O(n) |
| int at(size\_t); | Вывод элемента по индексу | O(n) |
| void remove(size\_t); | Удаление элемента по индексу | O(n) |
| size\_t get\_size(); | Получение размера списка | O(1) |
| void print\_to\_console(); | Вывод списка в консоль | O(n) |
| void clear(); | Очистка списка | O(n) |
| void set(size\_t, int); | Вставка элемента по индексу | O(n) |
| bool isEmpty(); | Проверка списка на пустоту | O(1) |
| void push\_front(List); | Вставка списка в начало | O(1) |

**Описание реализованных unit-тестов**

Для проверки работоспособности программы были реализованы unit-тесты. Бля этого был создан новый проект в Visual Studio, в котором написан класс LR1test, методы класса и их описание представлены в табл. 2, код в приложении В.

Таблица 2 – Описание методов класса LR1test

|  |  |
| --- | --- |
| Метод | Описание метода |
| init | Метод инициализирующий список  Добавляется 5 элементов с помощью метода push\_back() |
| testAt | Проверяется, что инициализация прошла правильно и все элементы на своих позициях. |
| testPushBack | С помощью метода push\_back(), добавляется элемент и проверяется, что он попал в конец |
| testPushFront | С помощью метода push\_back(), добавляется элемент и проверяется, что он попал в начало. |
| testGetSize | Проверяется, что размер инициализированного списка равен 5. Проверяется, что если из списка убрать один элемент, то размер его уменьшится на 1, а также, что у пустого списка размер 0. |
| testPopBack | Проверяется, что после вызова функции pop\_back(), размер списка уменьшится на 1, а так же последний элемент равен 2. |
| testPopFront | Проверяется, что после вызова функции pop\_front(), размер списка уменьшится на 1, а так же первый элемент равен 4. |
| testInsert | Проверяется, что после вызова функции insert(), элемент встанет на нужную позицию, и не удалит другой элемент по индексу вставки, а размер увеличится на 1. |
| testRemove | Проверяется, что после вызова функции remove(), удаляется элемент по нужному индексу, а размер уменьшится на 1. |
| testClear | Проверяется, что после вызова функции clear() список пуст и размер равен 0. |
| testIsEmpty | Проверяется, что до вызова функции clear() список не пуст, а после пуст. |
| testPushFrontList | Создается новый список, и с помощью функции push\_front(List), добавляется в начало существующего. Затем проверяется, что первый элемент, нового списка равен первому элементу второго. а 3 элемент, нового списка равен первому старого. |

### Пример работы

|  |  |
| --- | --- |
|  | Вывод |

**Приложение А**

**листинг файла node.h**

//Class node used for each element of list

class Node

{

public:

Node(int data) {

this->data = data;

this->next = nullptr;

this->prev = nullptr;

}

~Node() {

this->data = NULL;

this->next = nullptr;

this->prev = nullptr;

}

int data; //data into the node

Node\* next; //pointer to the next node

Node\* prev; //pointer to the previous element

};

**Приложение В**

**листинг файлов list.cpp и list.h**

**list.h**

#include <cstring>

#include <iostream>

#include "node.h"

//class List

class List

{

private:

Node\* head; //first node of list

Node\* end; //last node of list

size\_t size; //size of list

bool isEmptyEnd();

public:

List();

~List();

void push\_back(int elem);

void push\_front(int elem);

void pop\_back();

void pop\_front();

void insert(int, size\_t);

int at(size\_t);

void remove(size\_t);

size\_t get\_size();

void print\_to\_console();

void clear();

void set(size\_t, int);

bool isEmpty();

void push\_front(List);

};

**list.cpp**

#include "list.h"

#include <iostream>

//check if end is empty

bool List::isEmptyEnd()

{

return this->end == nullptr;

}

//constructor

List::List()

{

this->head = nullptr;

this->end = nullptr;

this->size = 0;

}

//destructor del all elems from list

List::~List()

{

clear();

}

//check if head is empty

bool List::isEmpty()

{

return this->head == nullptr;

}

//add new list befind current list

//set head to new list head

//connect two lists

//and sum sizes

void List::push\_front(List prevList)

{

prevList.end->next = this->head;

this->head->prev = prevList.end;

this->head = prevList.head;

size += prevList.size;

}

//retrun list size

size\_t List::get\_size()

{

return this->size;

}

//add new elem to list head

//if list empty, set elem to head

//if end empty, set elem to end

//else set new end, connect to previous elem

void List::push\_back(int elem)

{

this->size++;

Node\* newEnd= new Node(elem);

if (isEmpty()) {

head = newEnd;

}

else if (isEmptyEnd()) {

this->end = newEnd;

this->head->next = this->end;

this->end->prev = this->head;

}

else {

this->end->next = newEnd;

newEnd->prev = this->end;

this->end = newEnd;

}

}

//add new elem to list end

//if list empty, set elem to head

//if end empty, set elem to end

//else add set new, end connect to previous elem

void List::push\_front(int elem)

{

this->size++;

Node\* newHead = new Node(elem);

if (isEmpty()) {

head = newHead;

}

else if (isEmptyEnd()){

this->end = this->head;

this->head = newHead;

this->head->next = this->end;

this->end->prev = this->head;

}

else {

newHead->next = this->head;

this->head->prev = newHead;

this->head = newHead;

}

}

//pop elem from back of list

//if list emptu return

//if end is empty pop head

//else set new end and del previous

void List::pop\_back()

{

if (isEmpty()) {

return;

}

else if (isEmptyEnd()) {

size--;

this->head = nullptr;

}

else {

size--;

this->end = this->end->prev;

delete this->end->next;

this->end->next = nullptr;

}

}

//pop elem from head of list

//if list empty return

//if size = 1, delete head

//else set new head, and del previous

//if after pop head = end delete end

void List::pop\_front()

{

if (isEmpty()) {

return;

}

else if (size == 1) {

size--;

delete this->head;

this->head = nullptr;

}

else {

this->size--;

this->head = this->head->next;

this->head->prev = nullptr;

if (this->head == this->end) {

this->head = new Node(this->head->data);

this->end = nullptr;

this->head->next = nullptr;

}

}

}

//insert elem into list by index

//if index go over list return

//else find node with required index and insert new node before

void List::insert(int elem, size\_t index)

{

if (index > ((this->size) - 1)) {

return;

}

else if (index == 0) {

push\_front(elem);

}

else

{

this->size++;

Node\* newNode = new Node(elem);

Node\* curNode = this->head;

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

curNode = curNode->next;

}

newNode->next = curNode;

newNode->prev = curNode->prev;

curNode->prev->next = newNode;

curNode->prev = newNode;

}

}

int List::at(size\_t index)

{

if (index == 0) {

return head->data;

}

else if (index == size - 1) {

return end->data;

}

else {

Node\* curNode = this->head;

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

curNode = curNode->next;

}

return curNode->data;

}

}

void List::remove(size\_t index)

{

if (index > (this->size - 1)) {

std::cout << "Индекс больше размера списка" << std::endl;

}

else if (index == this->size - 1) {

pop\_back();

}

else if (index == 0) {

pop\_front();

} {

size--;

Node\* curNode = this->head;

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

curNode = curNode->next;

}

curNode->prev->next = curNode->next;

curNode->next->prev = curNode->prev;

delete curNode;

}

}

//print all nodes to consloe one by one

//iterate in list and print each data in it and separate nodes by " -> "

void List::print\_to\_console()

{

if (isEmpty()) {

std::cout << "Список пустой" << std::endl;

}

else {

Node\* curNode = this->head;

do {

std::cout << curNode->data << " -> ";

curNode = curNode->next;

} while (curNode != nullptr);

std::cout << "nullptr" << std::endl;

}

}

//clear the list by pop first node while list is not empty

void List::clear(){

while (!isEmpty()) {

pop\_front();

}

}

//set new data into node by index

//if index higher last node index return

//if index = index of the end, set data to end

//else iterete in list and find required node and set data to it

void List::set(size\_t index, int data)

{

if (index > (this->size - 1)) {

return;

}

else if (index == this->size - 1) {

this->end->data = data;

}

else {

Node\* curNode = this->head;

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

curNode = curNode->next;

}

curNode->data = data;

}

}

**Приложение В**

**листинг файла LR1\_test.cpp**

#include "pch.h"

#include "CppUnitTest.h"

#include "../lr1\_aisd/list.h"

#include "../lr1\_aisd/list.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace LR1test

{

TEST\_CLASS(LR1test)

{

private:

List list = List();

public:

TEST\_METHOD\_INITIALIZE(init)

{

list.push\_back(5);

list.push\_back(4);

list.push\_back(3);

list.push\_back(2);

list.push\_back(1);

}

TEST\_METHOD(testAt)

{

Assert::AreEqual(list.at(0), 5);

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

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

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

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

}

TEST\_METHOD(testPushBack)

{

list.push\_back(0);

Assert::AreEqual(list.at(5), 0);

}

TEST\_METHOD(testPushFront)

{

list.push\_front(6);

Assert::AreEqual(list.at(0), 6);

}

TEST\_METHOD(testGetSize)

{

Assert::AreEqual(list.get\_size(), (size\_t)5);

list.pop\_back();

Assert::AreEqual(list.get\_size(), (size\_t)4);

List list2 = List();

Assert::AreEqual(list2.get\_size(), (size\_t)0);

}

TEST\_METHOD(testPopBack)

{

list.pop\_back();

Assert::AreEqual(list.get\_size(), (size\_t)4);

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

}

TEST\_METHOD(testPopFront)

{

list.pop\_front();

Assert::AreEqual(list.get\_size(), (size\_t)4);

Assert::AreEqual(list.at(0), 4);

}

TEST\_METHOD(testInsert)

{

list.insert(222, 3);

Assert::AreEqual(list.at(3), 222);

Assert::AreEqual(list.at(4), 2);

Assert::AreEqual(list.get\_size(), (size\_t)6);

}

TEST\_METHOD(testRemove)

{

list.remove(3);

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

Assert::AreEqual(list.get\_size(), (size\_t)4);

}

TEST\_METHOD(testClear)

{

list.clear();

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

Assert::AreEqual(list.get\_size(), (size\_t)0);

}

TEST\_METHOD(testIsEmpty)

{

Assert::IsFalse(list.isEmpty());

list.clear();

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

}

TEST\_METHOD(testPushFrontList)

{

List list2;

list2.push\_front(10);

list2.push\_back(20);

list.push\_front(list2);

Assert::AreEqual(list.at(0), 10);

Assert::AreEqual(list.at(2), 5);

}

};

}