**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 4**

Тема: Двусвязный Список

|  |  |  |
| --- | --- | --- |
| Студент гр. 0309 |  | Головатюк К.А. |
| Преподаватель |  | Тутуева А.В |

Санкт-Петербург

2021

### Постановка задачи

Реализовать класс связного списка с набором методов. Реализовать unit-тесты ко всем реализуемым методам. Двусвязный список.

### Цель работы

Научиться создавать и оптимизировать алгоритмы для работы со списками.

### Описание реализуемого класса и методов.

Среда разработки – Visual Studio

Язык программирования – C++

В ходе выполнения лабораторной работы было создано два класса: для хранения типа данных элемента и адреса на следующий и предыдущий элементы списка, и класс списка.

Таблица 1 – описание класса Node

|  |  |  |
| --- | --- | --- |
| Название метода/поля | Описание | Оценка временной сложности  (Для методов) |
| Public int value | Поле данных |  |
| public Node next; | Указатель на следующий элемент списка | O(1) |
| public Node prev; | Указатель на предыдущий элемент списка | O(1) |

Таблица 2 – описание класса List

|  |  |  |
| --- | --- | --- |
| Название метода/поля | Описание | Оценка временной сложности  (Для методов) |
| Node \*head | Указатель на первый элемент списка |  |
| Size\_t size | Размер списка |  |
| public void push\_back(int) | Добавление в конец списка | О(1) |
| public void push\_front(int) | Добавление в начало списка | O(1) |
| public void pop\_back() | Удаление последнего элемента | O(1) |
| public void pop\_front() | Удаление первого элемента | O(1) |
| public int at(size\_t) | получение элемента по индексу | O(n) |
| public void insert(int, size\_t) | добавление элемента по индексу | O(n) |
| public void remove(int size\_t) | удаление элемента по индексу | O(n) |
| public int getSize() | получение размера списка | O(1) |
| public void set(int, size\_t) | замена элемента по индексу на передаваемый элемент | O(n) |
| public void insert(List \*, size\_t) | Вставляет список в список, после элемента с заданным индексом | O(n) |
| friend ostream &operator<<(ostream &out, const List \*list) | вывод элементов списка через оператор вывода. | O(n) |
| public void clear() | удаление всех элементов списка | O(n) |
| public bool isEmpty() | проверка на пустоту списка | O(1) |

### Описание реализованных unit-тестов

Для проверки реализованных методов были написаны unit-тесты. Их названия представлены ниже.

Изображение выглядит как текст

Автоматически созданное описание

Unit-тесты разделены на классы:

DeleteTest – проверка методов связанных с удалением.

InputTest – проверка методов связанных с вводом.

OutputTest – проверка методов связанных с выводом.

### Пример работы :

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

### Листинг

**List.h**

#pragma once

#include <iostream>

using namespace std;

class Node {

public:

int value;

Node\* next = nullptr;

Node\* prev = nullptr;

~Node() {

next = nullptr;

prev = nullptr;

delete(next);

delete(prev);

};

Node() {

value = NULL;

next = nullptr;

prev = nullptr;

};

};

class List {

private:

Node\* head = nullptr;

size\_t size = 0;

public:

List();

List(int);

~List();

void push\_back(int);

void push\_front(int);

void pop\_back();

void pop\_front();

void insert(int, size\_t);

int at(size\_t);

void remove(size\_t);

size\_t get\_size();

void clear();

void set(size\_t, int);

bool isEmpty();

void insert(List\*, size\_t);

friend ostream& operator<< (ostream& out, const List\* list);

};

**List.cpp**

#include "List.h"

using namespace std;

List::List(int value) {

head = new Node();

size = 0;

this->push\_back(value);

}

List::List() {

head = nullptr;

size = 0;

}

List::~List() {

if (this->head == nullptr) {

delete(this->head);

return;

}

while (this->head->next)

{

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

delete(this->head->prev);

this->head->prev = nullptr;

}

delete(this->head);

}

void List::push\_back(int value) {

size++;

if (this->head == nullptr) {

this->head = new Node();

this->head->value = value;

this->head->next = nullptr;

this->head->prev = nullptr;

return;

}

Node\* add = new Node();

add->value = value;

Node\* iter = this->head;

while (iter->next) {

iter = iter->next;

}

iter->next = add;

add->prev = iter;

}

void List::push\_front(int value) {

size++;

if (this->head == nullptr) {

this->head = new Node();

this->head->value = value;

this->head->next = nullptr;

this->head->prev = nullptr;

return;

}

Node\* add = new Node();

add->value = value;

add->next = this->head;

this->head->prev = add;

this->head = add;

}

void List::pop\_back() {

if (this->head == nullptr) {

return;

}

this->size--;

if (this->head->next == nullptr) {

delete(this->head);

this->head = nullptr;

return;

}

Node\* iter = this->head;

while (iter->next->next != nullptr) {

iter = iter->next;

}

delete(iter->next);

iter->next = nullptr;

}

void List::pop\_front() {

if (this->head == nullptr) {

return;

}

this->size--;

if (this->head->next == nullptr) {

delete(this->head);

this->head = nullptr;

return;

}

Node\* del = this->head;

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

this->head->prev = nullptr;

delete(del);

}

void List::insert(int value, size\_t pos) {

if (size == 0)

return;

if (pos > this->size) {

return;

}

if (pos == this->size) {

this->push\_back(value);

return;

}

size++;

Node\* iter = this->head;

while (pos > 0)

{

iter = iter->next;

pos--;

}

Node\* add = new Node();

add->value = value;

add->prev = iter->prev;

add->next = iter;

iter->prev = add;

if (add->prev) {

add->prev->next = add;

}

else {

this->head = add;

}

}

int List::at(size\_t pos) {

if (pos < 0)

return NULL;

if (pos >= this->size) {

return NULL;

}

Node\* iter = this->head;

while (pos != 0)

{

iter = iter->next;

pos--;

}

return iter->value;

}

void List::remove(size\_t pos) {

if (pos >= this->size) {

return;

}

this->size--;

Node\* iter = this->head;

while (pos != 0)

{

iter = iter->next;

pos--;

}

if (iter->prev) {

iter->prev->next = iter->next;

}

else if (iter->next) {

this->head = iter->next;

this->head->prev = nullptr;

delete(iter);

return;

}

else {

delete(this->head);

this->head = nullptr;

return;

}

if (iter->next)

iter->next->prev = iter->prev;

delete(iter);

}

size\_t List::get\_size() {

return this->size;

}

void List::clear() {

while (this->head->next)

{

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

delete(this->head->prev);

this->head->prev = nullptr;

}

delete(this->head);

this->head = nullptr;

}

void List::set(size\_t pos, int value) {

if (pos >= this->size) {

return;

}

Node\* iter = this->head;

while (pos != 0)

{

iter = iter->next;

pos--;

}

iter->value = value;

}

bool List::isEmpty() {

if (this->head == nullptr) {

return true;

}

return false;

}

void List::insert(List\* in, size\_t pos) {

Node\* iter = in->head;

while (iter)

{

this->insert(iter->value, pos);

pos++;

iter = iter->next;

}

}

ostream& operator<<(ostream& out, const List\* list) {

Node\* iter = list->head;

if (iter == nullptr) {

out << "Empty list";

return out;

}

out << "[";

while (iter)

{

if (iter->next == nullptr) {

out << iter->value;

break;

}

out << iter->value << ", ";

iter = iter->next;

}

out << "]";

return out;

}

**main.cpp**

#include "List.h"

#include <iostream>

using namespace std;

int main()

{

List \*test = new List();

List \*testIn = new List();

cout << "list test: " << test << endl;

cout << endl;

cout << "list test push back [1,2,3]" << endl;

test->push\_back(1);

test->push\_back(2);

test->push\_back(3);

cout << "list test after push back: " << test << endl;

cout << endl;

cout << "list test push front [1,2,3]" << endl;

test->push\_front(1);

test->push\_front(2);

test->push\_front(3);

cout << "list test after push front: " << test << endl;

cout << endl;

cout << "list test pop back" << endl;

test->pop\_back();

cout << "list test after pop back: " << test << endl;

cout << endl;

cout << "list test pop front" << endl;

test->pop\_front();

cout << "list test after pop front: " << test << endl;

cout << endl;

cout << "list test insert pos = 2, value = 3" << endl;

test->insert(3, 2);

cout << "list test after insert: " << test << endl;

cout << endl;

cout << "list test at pos = 2: " << test->at(2) << endl;

cout << "list test set pos = 2, value = 4" << endl;

test->set(2, 4);

cout << "list test after set: " << test << endl;

cout << endl;

cout << "list test remove pos = 2 " << endl;

test->remove(2);

cout << "list test after remove: " << test << endl;

cout << endl;

cout << "list test size: " << test->get\_size() << endl;

cout << endl;

cout << "list test insert list (variant 4 function 14) pos = 2" << endl;

testIn->push\_back(9);

testIn->push\_back(5);

testIn->push\_back(9);

cout << "list testIn: " << testIn << endl;

test->insert(testIn, 2);

cout << "list test after insert: " << test << endl;

cout << endl;

cout << "list test is empty: " << test->isEmpty() << endl;

test->clear();

cout << "list test is empty after clear: " << test->isEmpty() << endl;

delete(test);

delete(testIn);

return 0;

}