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 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	O(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

```
{
    setlocale(LC_ALL, "RUS");
    List list = List();
    list.push_back(5);
    list.push_back(4);
    list.push_front(3);
    list.push_front(2);
    list.push_back(1);
    list.print_to_console();

    list.insert(222, 3);
    list.print_to_console();

    list.remove(3);
    list.print_to_console();

    list.pop_back();
    list.print_to_console();


    List list2;
    list2.push_front(10);
    list2.push_back(20);
    list2.print_to_console();
    list.push_front(list2);
    list.print_to_console();

    cout << "Размер списка: "
        << list.get_size() << endl;

    list.clear();
    list.print_to_console();
}
```

```
2 -> 3 -> 5 -> 4 -> 1 -> nullptr
3 5
2 -> 3 -> 5 -> 222 -> 4 -> 1 -> nullptr
2 -> 3 -> 5 -> 4 -> 1 -> nullptr
2 -> 3 -> 5 -> 4 -> nullptr
10 -> 20 -> nullptr
10 -> 20 -> 2 -> 3 -> 5 -> 4 -> nullptr
Размер списка: 6
Список пустой
```

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);
		}
	};
}
```