«Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Алтайский государственный технический университет им. И.И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчёт защищён с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель Бубнова Н.Д.

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2021 г.

Отчёт

Лабораторной работе №2

**«**Связные списки**»**

Студент группы ПИ 92 В.М. Шульпов

Старший преподаватель Бубнова Н.Д.

Барнаул 2021

**Задание:**

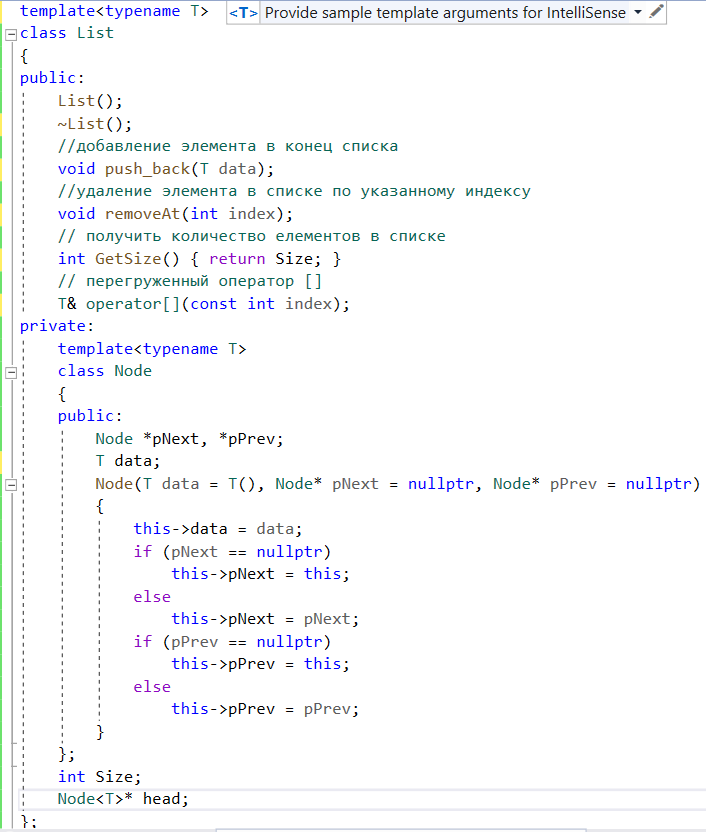
1. Ознакомиться с постановкой задачи
2. Написать программу для ее реализации
3. Разработать тесты
4. Исходные данные поместить в файл input.dat. Если вид исходных данных не указан в варианте, выбираете любой подходящий.
5. Исходные данные и результаты вывести на экран. Исходные данные и результаты вывести также в вы

**Вариант 27**

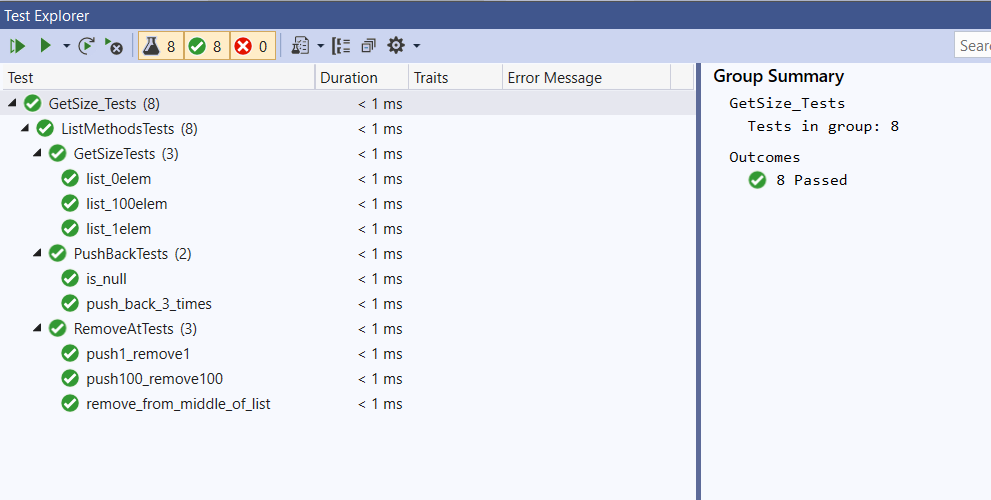
Задан двусвязный циклический список с головой. Напечатать его. Удалить первый отрицательный элемент этого списка. Напечатать результирующий список.

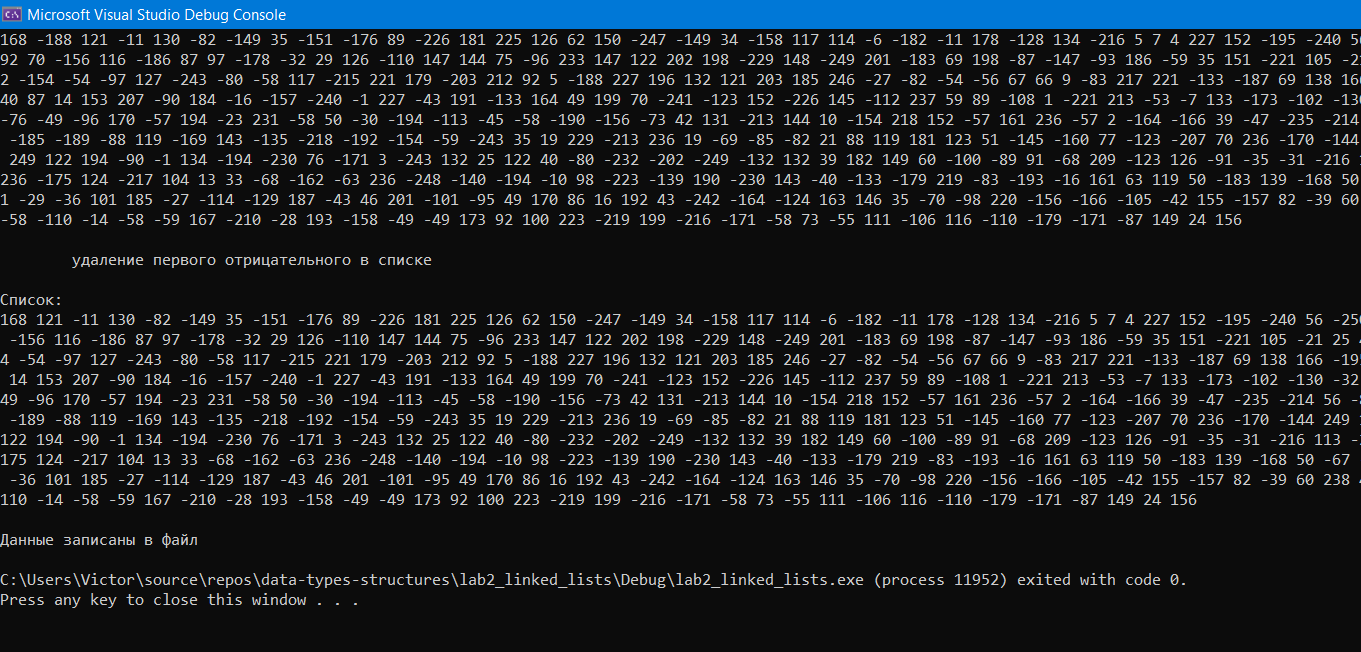
Структура списка



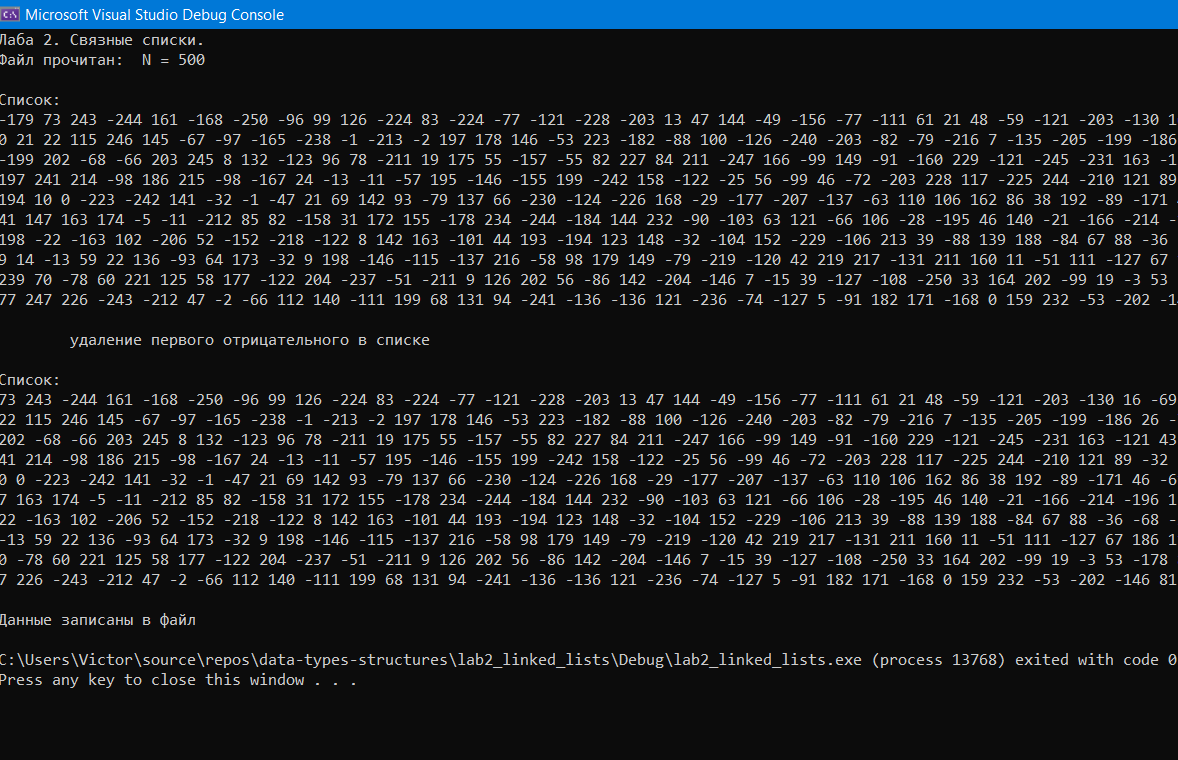


Тесты

первый элемент положительный:



Первый элемент отрицательный:



Код

**List.h**

#pragma once

#include <fstream>

#include <iostream>

#include <string>

#include <ctime>

using namespace std;

/\*

\* циклический двусвязный список с головой

\* реализация

\*/

template<typename T>

class List

{

public:

List();

~List();

//добавление элемента в конец списка

void push\_back(T data);

//удаление элемента в списке по указанному индексу

void removeAt(int index);

// получить количество елементов в списке

int GetSize() { return Size; }

// перегруженный оператор []

T& operator[](const int index);

private:

template<typename T>

class Node

{

public:

Node \*pNext, \*pPrev;

T data;

Node(T data = T(), Node\* pNext = nullptr, Node\* pPrev = nullptr)

{

this->data = data;

if (pNext == nullptr)

this->pNext = this;

else

this->pNext = pNext;

if (pPrev == nullptr)

this->pPrev = this;

else

this->pPrev = pPrev;

}

};

int Size;

Node<T>\* head;

};

template<typename T>

List<T>::List()

{

Size = 0;

head = nullptr;

}

template<typename T>

List<T>::~List()

{

}

template<typename T>

void List<T>::push\_back(T data)

{

if (head == nullptr)

{

head = new Node<T>(data);//создание первого элемента

}

else

{

//добавление элемента в конец списка

Node<T>\* before\_last = this->head->pPrev;//получаем временно последний Node (далее - предпоследний)

Node<T>\* last = new Node<T>(data);//создаем новый последний Node

last->pNext = this->head;//следующий за последним - голова списка

before\_last->pNext = last;//следующий за предпоследним - последний

last->pPrev = before\_last;//предыдущий перед последним - бывший предпоследний

head->pPrev = last;//предыдущий перед головой - последний

}

Size++;

}

template<typename T>

void List<T>::removeAt(int index)

{

if (this->GetSize() < index || index < 0) {

throw exception("Wrong function using");

}

else {

if (index == 0)

{

Node<T>\* previous = head->pPrev;

//cout << "prev = " << previous->data << endl;

Node<T>\* next = head->pNext;

previous->pNext = next;

next->pPrev = previous;

if (Size > 0) this->head = this->head->pNext;

else head = nullptr;//если список стал пустым

Size--;

}

else

{

Node<T>\* previous = this->head;

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

{

previous = previous->pNext;

}

Node<T>\* toDelete = previous->pNext;

Node<T>\* next = toDelete->pNext;

previous->pNext = next;

next->pPrev = previous;

delete toDelete;

Size--;

}

}

}

template<typename T>

T& List<T>::operator[](const int index)

{

int counter = 0;

Node<T>\* current = this->head;

while (current != nullptr)

{

if (counter == index)

{

return current->data;

}

current = current->pNext;

counter++;

}

}

**Lab2\_linked\_lists.cpp**

#include <fstream>

#include <iostream>

#include <string>

#include <ctime>

#include "lab2\_linked\_lists.h"

#include "List.h"

using namespace std;

int main()

{

srand(time(NULL));

setlocale(LC\_ALL, "Rus");

cout << "Лаба 2. Связные списки." << endl;

string in\_filneame = "input.dat";

string out\_filneame = "output.dat";

int i, n;

//чтения входных данных из файла (количества элементов)

try {

n = ReadArrayN(in\_filneame);

cout << "Файл прочитан:\tN = " << n << endl;

}

catch (exception exc) {

cout << "\n" << exc.what() << endl << "n = 0" << endl;

n = 0;

}

//заполнение списка рандомными числами

List<int> lst;

int value;

for (int i = 0; i < n; i++) {

value = rand() % 500 - 250;

lst.push\_back(value);

}

DisplayList(lst);

cout << "\n\n\tудаление первого отрицательного в списке" << endl;

for (int i = 0; i < n; i++) {

if (lst[i] < 0) {

lst.removeAt(i);

break;

}

}

DisplayList(lst);

try {

SaveListInFile(lst, out\_filneame);//запись выходных данных

cout << "\n\nДанные записаны в файл" << endl;

}

catch (exception exc) {

cout << "\n" << exc.what() << endl;

}

}

//вывод списка

void DisplayList(List<int> &list) {

if (list.GetSize()) {

cout << "\nСписок: " << endl; //вывод списка

for (int i = 0; i < list.GetSize(); i++)

cout << list[i] << " ";

}

else {

cout << "\nСписок пуст" << endl;

}

}

//чтения количества элементов массива из файла

int ReadArrayN(const std::string& filename) {

ifstream in;

string str\_n;

int n = 0;

// Open the file.

in.open(filename, std::ios::in | std::ios::binary);

if (in.is\_open()) {

getline(in, str\_n);

}

else

{

throw exception("Exception: file did't open");

}

try {

n = stoi(str\_n);

}

catch (exception exc) {

throw exception("Exception: convert failure");

}

return n;

}

//сохранения массива в файл

bool SaveListInFile(List<int>& list, const std::string& filename)

{

/\*

Первое число в файле будет N, остальные - элементы массива

\*/

ofstream out;

// Open the file.

out.open(filename, std::ios::out | std::ios::binary);

// Validate that the file is open.

if (!out.is\_open())

return false;

out << list.GetSize() << "\n\n";

for (int i = 0; i <= list.GetSize(); i++) {

out << list[i] << endl;

}

out.close();

return true;

}

**Lab2\_linked\_lists.h**

#pragma once

#include "List.h"

int ReadArrayN(const std::string& filename);//чтения количества элементов массива из файла

bool SaveListInFile(List<int>& list, const std::string& filename);//сохранения массива в файл

void DisplayList(List<int>& list);

**GetSize\_Tests.cpp**

#include "pch.h"

#include "CppUnitTest.h"

#include "..\lab2\_linked\_lists\lab2\_linked\_lists.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace ListMethodsTests

{

TEST\_CLASS(GetSizeTests)

{

public:

TEST\_METHOD(list\_0elem)

{

List<int> lst;

Assert::AreEqual(0, lst.GetSize());

}

TEST\_METHOD(list\_100elem)

{

List<int> lst;

for (int i = 0; i < 100; i++) {

lst.push\_back(rand() % 500 - 250);

}

Assert::AreEqual(100, lst.GetSize());

}

TEST\_METHOD(list\_1elem)

{

List<int> lst;

lst.push\_back(44);

Assert::AreEqual(1, lst.GetSize());

}

};

TEST\_CLASS(RemoveAtTests)

{

public:

TEST\_METHOD(push1\_remove1)

{

List<int> lst;

lst.push\_back(44);

lst.removeAt(1);

Assert::AreEqual(0, lst.GetSize());

}

TEST\_METHOD(push100\_remove100)

{

List<int> lst;

for (int i = 0; i < 100; i++) {

lst.push\_back(rand() % 500 - 250);

}

for (int i = 0; i < 100; i++) {

lst.removeAt(0);

}

Assert::AreEqual(0, lst.GetSize());

}

TEST\_METHOD(remove\_from\_middle\_of\_list)

{

List<int> lst;

for (int i = 0; i < 5; i++) {

lst.push\_back(i);

}

for (int i = 0; i < 3; i++) {

lst.removeAt(1);

}

Assert::AreEqual(4, lst[1]);

}

};

TEST\_CLASS(PushBackTests)

{

public:

TEST\_METHOD(push\_back\_3\_times)

{

List<int> lst;

lst.push\_back(11);

lst.push\_back(22);

lst.push\_back(33);

Assert::AreEqual(lst[0], 11);

Assert::AreEqual(lst[1], 22);

Assert::AreEqual(lst[2], 33);

}

TEST\_METHOD(is\_null)

{

List<int> lst;

lst.push\_back(11);

lst.removeAt(0);

Assert::AreEqual(0, lst.GetSize());

lst.push\_back(333);

Assert::AreEqual(1, lst.GetSize());

}

};

}