Министерство образования Республики Беларусь

Учреждение образования

Брестский государственный технический университет

Кафедра ИИТ

Отчет №14

По лабораторной работе

Тема:«Динамические структуры данных: списки и деревья»

Выполнил:

Студ. гр.ИИ-23

Романюк А.П.

Проверил:

Брест 2023

Вариант №8

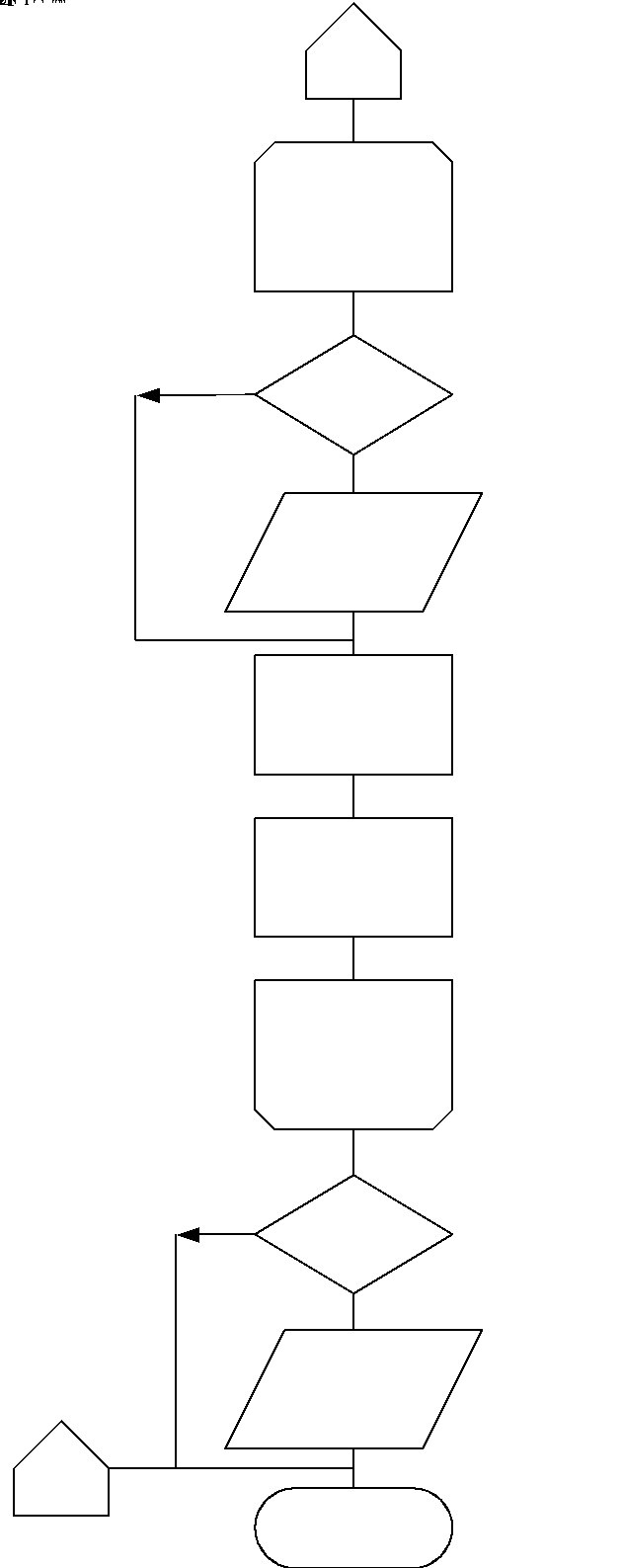
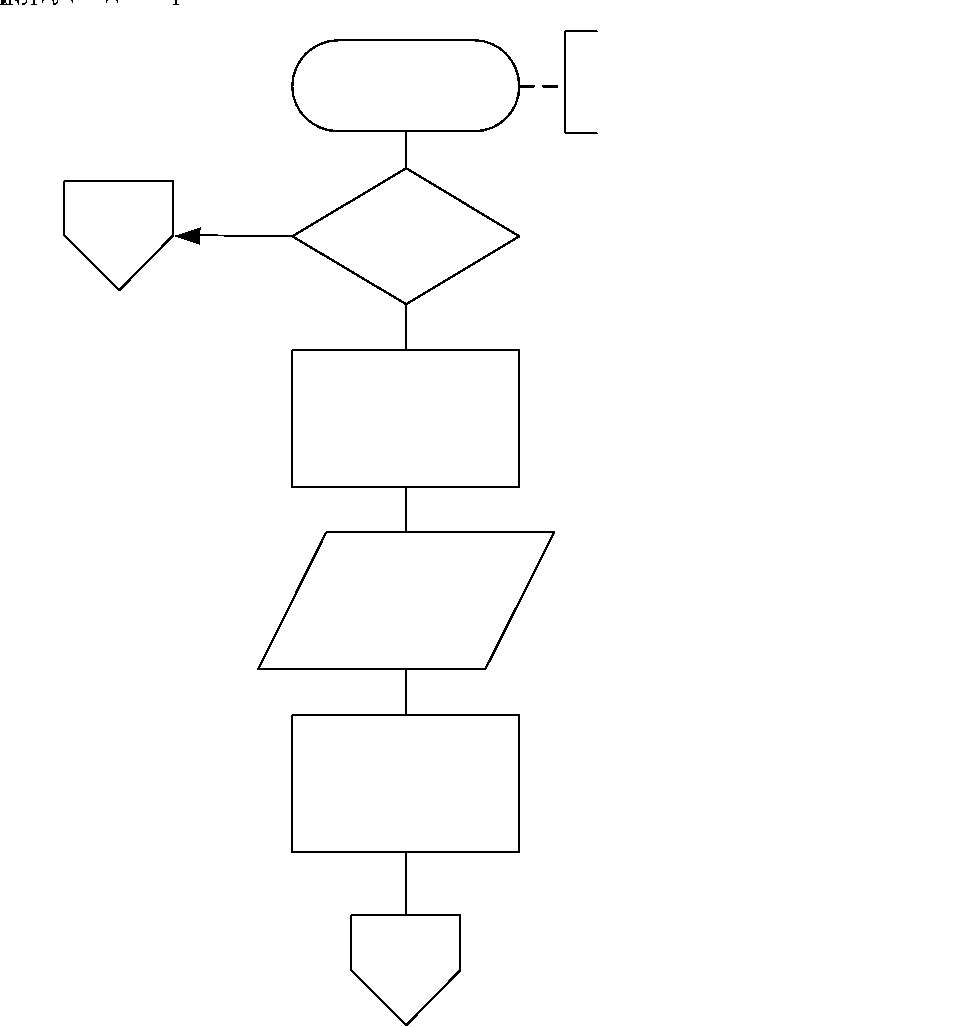
Цель работы: приобретение навыков работы с динамической памятью и указателями на С/C++; изучение принципов работы с динамическими структурами данных: списками и деревьями.

Текст задания:

Общие требования: в начале программы вывести задание; в процессе работы выводить подсказки пользователю (что ему нужно ввести, чтобы продолжить выполнение программы). Основные алгоритмы, ввод/вывод списков реализовать в виде функций с необходимыми параметрами. После работы программы вся динамически выделенная память должна быть освобождена. Взаимодействие с пользователем организовать в виде простого меню, обеспечивающего возможность переопределения исходных данных и завершение работы программы.

8. Написать программу, обеспечивающую работу с однонаправленным нециклическим списком: добавление/удаление элементов в голову, просмотр списка, инверсию списка

Блок-схема:

****

Текст программы:

**#include <iostream>**

**using namespace std;**

**struct list {**

**int data;**

**list\* next;**

**};**

**void menu(list\* head);**

**void create\_list(list\*& head);**

**void create\_node(list\*& head);**

**void add(list\*& head);**

**void del(list\*& head);**

**void output(list\* head);**

**void reverce(list\*& head);**

**bool empty(list\* head);**

**int main() {**

**list\* head;**

**head = NULL;**

**menu(head);**

**}**

**void menu(list \*head) {**

**int choise;**

**while (true) {**

**cout << "выберите действие:\n";**

**cout << "1.создание списка\n";**

**cout << "2.добавление элемента\n";**

**cout << "3.удаление элемента\n";**

**cout << "4.просмотр списка\n";**

**cout << "5.инверсия списка\n";**

**cout << "6.выход\n";**

**cin >> choise;**

**switch (choise) {**

**case 1:**

**system("cls");**

**create\_list(head);**

**break;**

**case 2:**

**system("cls");**

**add(head);**

**break;**

**case 3:**

**system("cls");**

**del(head);**

**break;**

**case 4:**

**system("cls");**

**output(head);**

**break;**

**case 5:**

**system("cls");**

**reverce(head);**

**break;**

**case 6:**

**return;**

**break;**

**}**

**}**

**}**

**void create\_list(list\*&head) {**

**head = NULL;**

**cout << "введите колво элементов списка\n";**

**int n;**

**cin >> n;**

**cout << "введите "<<n<<" элементов списка\n";**

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

**create\_node(head);**

**}**

**if (n > 0) {**

**output(head);**

**}**

**}**

**void create\_node(list\*& head) {**

**list \*new\_node = new list;**

**cin >> new\_node->data;**

**new\_node->next = NULL;**

**list\* q = head;**

**if (q == NULL) {**

**new\_node->next = head;**

**head = new\_node;**

**}**

**else {**

**while (q->next) {**

**q = q->next;**

**}**

**new\_node->next = q->next;**

**q->next = new\_node;**

**}**

**}**

**void del(list\*& head) {**

**if (empty(head) == 1) {**

**output(head);**

**cout << "введите элемент для удаления";**

**int d;**

**cin >> d;**

**list\* q = head;**

**bool find = false;**

**while (q) {**

**if (d == q->data) {**

**find = true;**

**if (q != NULL) {**

**if (q == head) {**

**head = head->next;**

**delete q;**

**q = head;**

**}**

**else {**

**list\* temp = head;**

**while (temp->next != q) {**

**temp = temp->next;**

**}**

**temp->next = q->next;**

**delete q;**

**q = temp;**

**}**

**}**

**}**

**else q = q->next;**

**}**

**if (find == false) {**

**cout << "Данный элемент не найден.\n";**

**}**

**else {**

**output(head);**

**cin.ignore();**

**}**

**}**

**}**

**void output(list\* head) {**

**if (empty(head) == 1) {**

**list\* q = head;**

**cout << "список\n";**

**while (q) {**

**cout << q->data<<" ";**

**q = q->next;**

**}**

**cout << endl;**

**}**

**}**

**void reverce(list\*& head) {**

**if (empty(head) == 1) {**

**list\* reversed\_list=NULL;**

**for (list\* node = head, \*next\_node; node != NULL; node = next\_node) {**

**next\_node = node->next;**

**node->next = reversed\_list;**

**reversed\_list = node;**

**}**

**head = reversed\_list;**

**output(head);**

**}**

**}**

**bool empty(list\* head) {**

**if (head != NULL) return 1; else {**

**cout << "список пуст\n";**

**return 0;**

**}**

**}**

**void add(list\*& head) {**

**int choise;**

**do {**

**cout << "введите новый элемент списка: ";**

**create\_node(head);**

**cout << "хотите добавить еще элемент? (1 - да, 2 - нет)\n";**

**cin >> choise;**

**if (choise != 1 && choise != 2) {**

**cout << "Вы ввели неправильный номер.\nВвод будет прекращен.\n";**

**cin.ignore();**

**cin.ignore();**

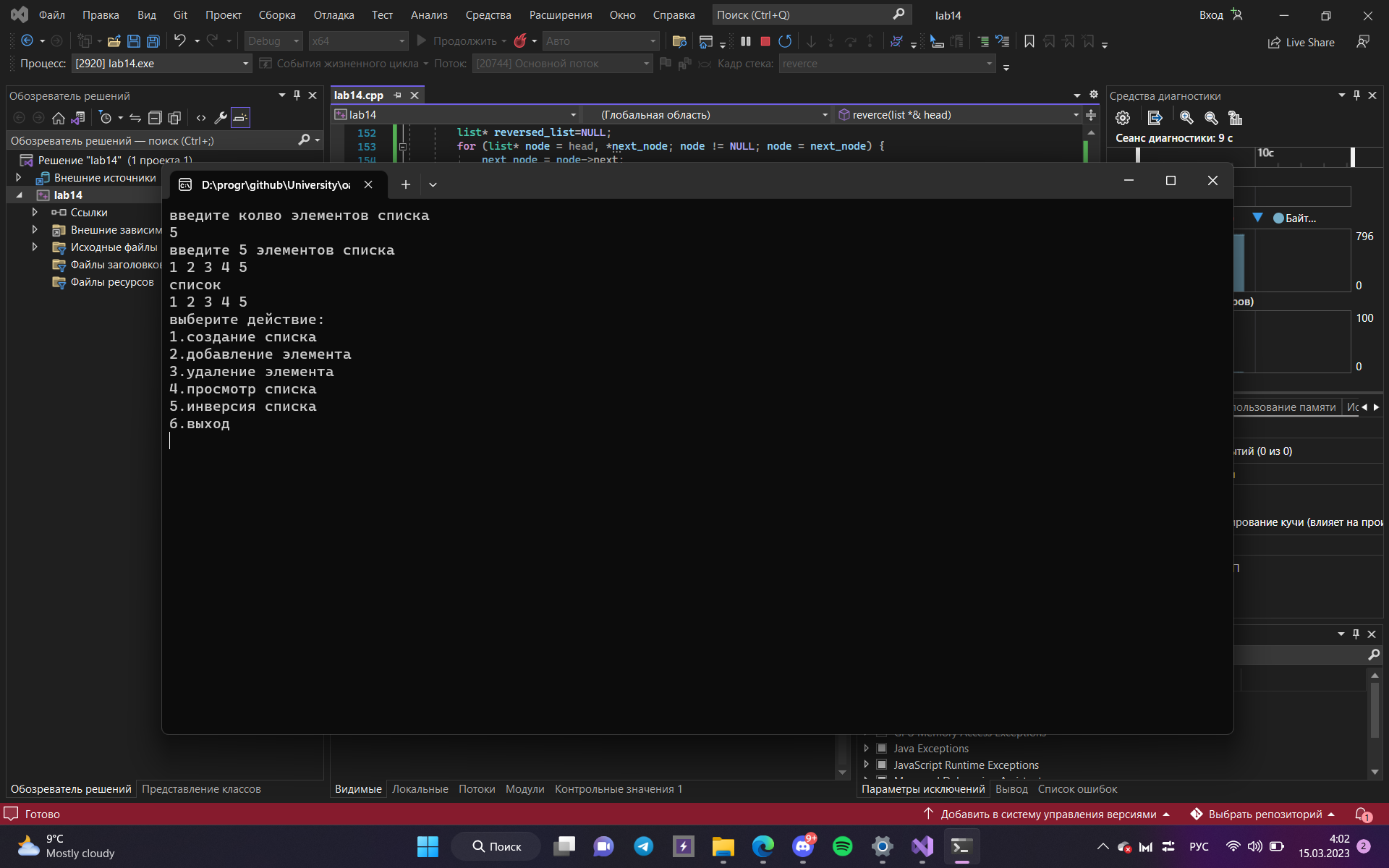
**}**

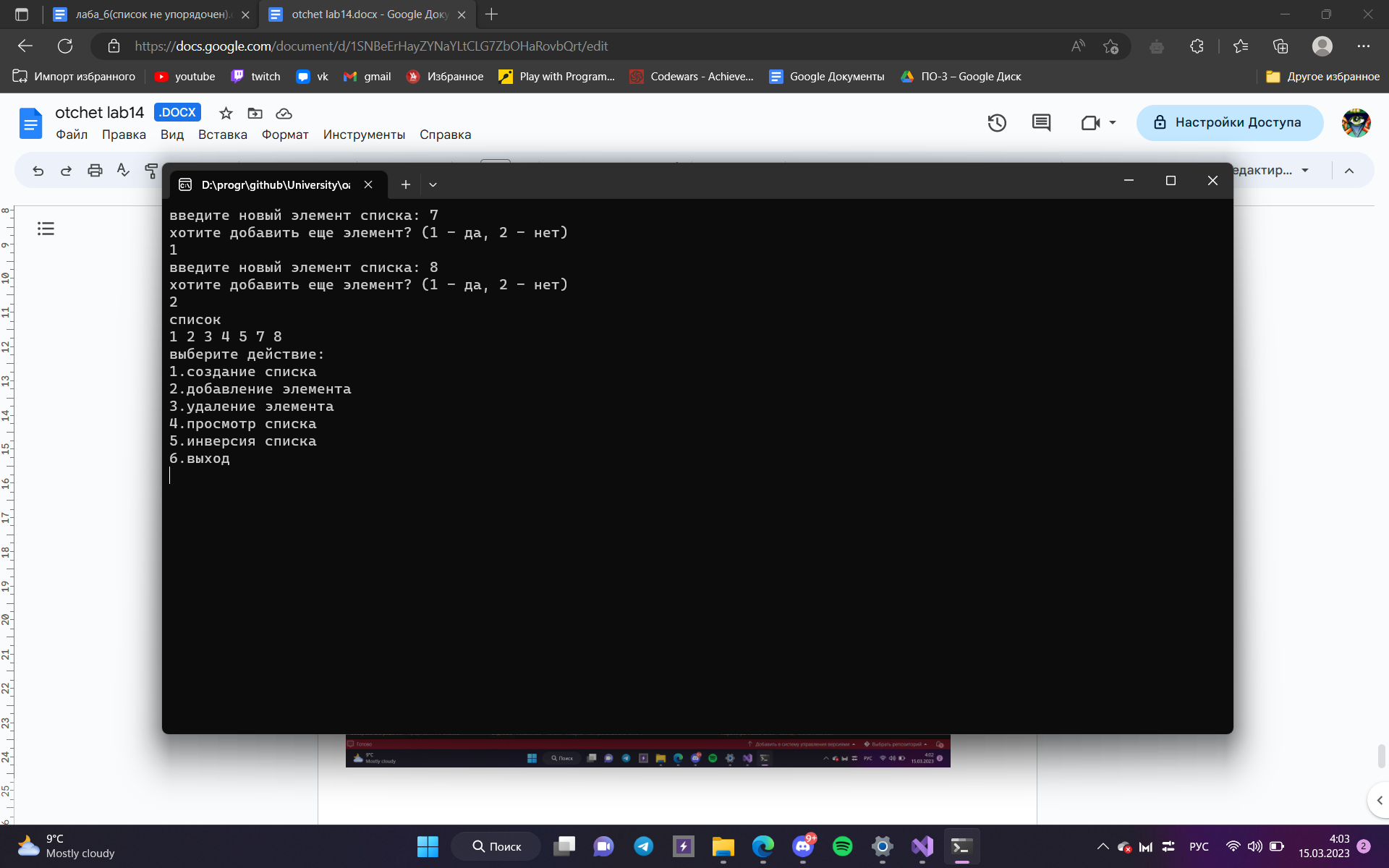
**} while (choise != 2);**

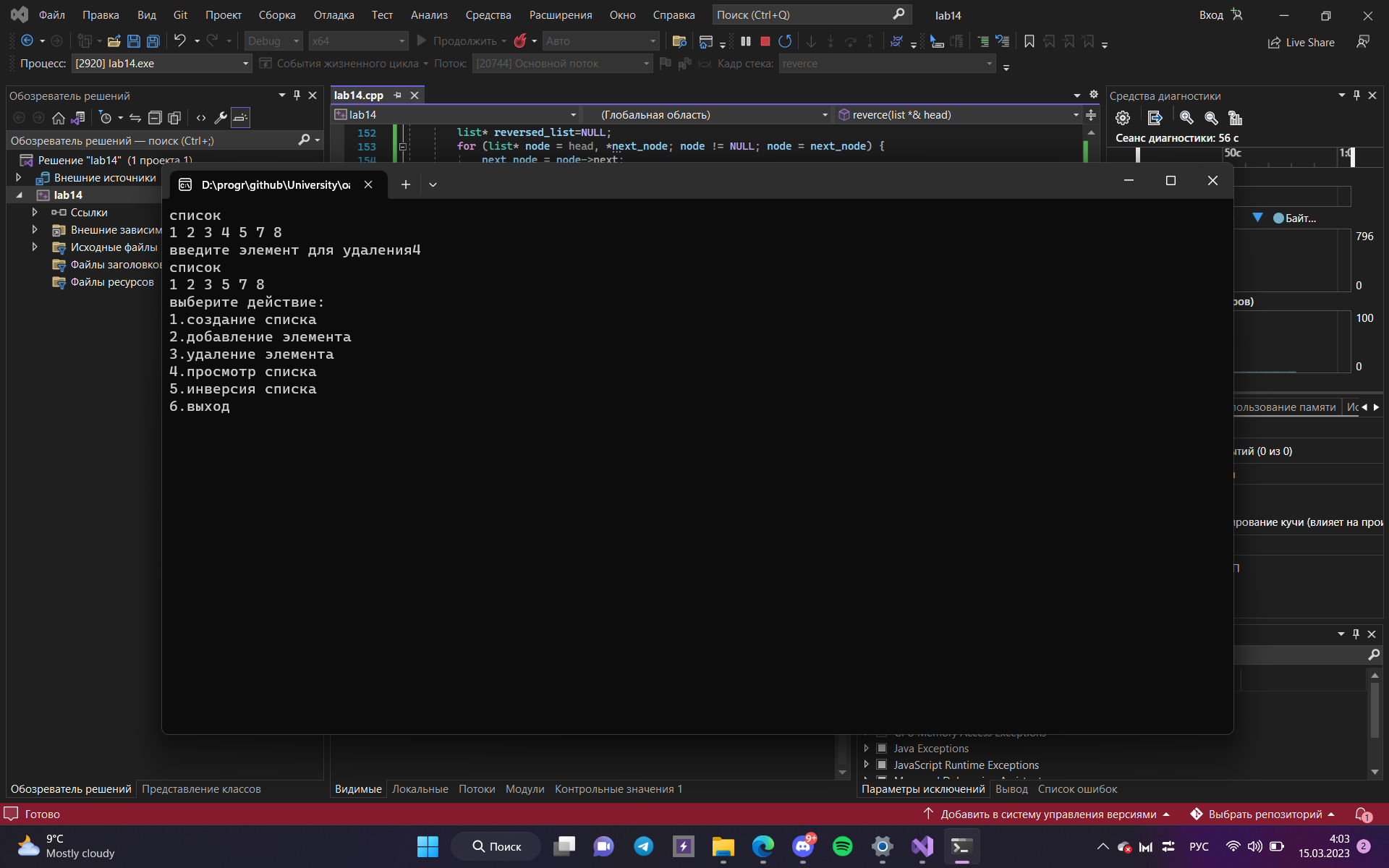
**output(head);**

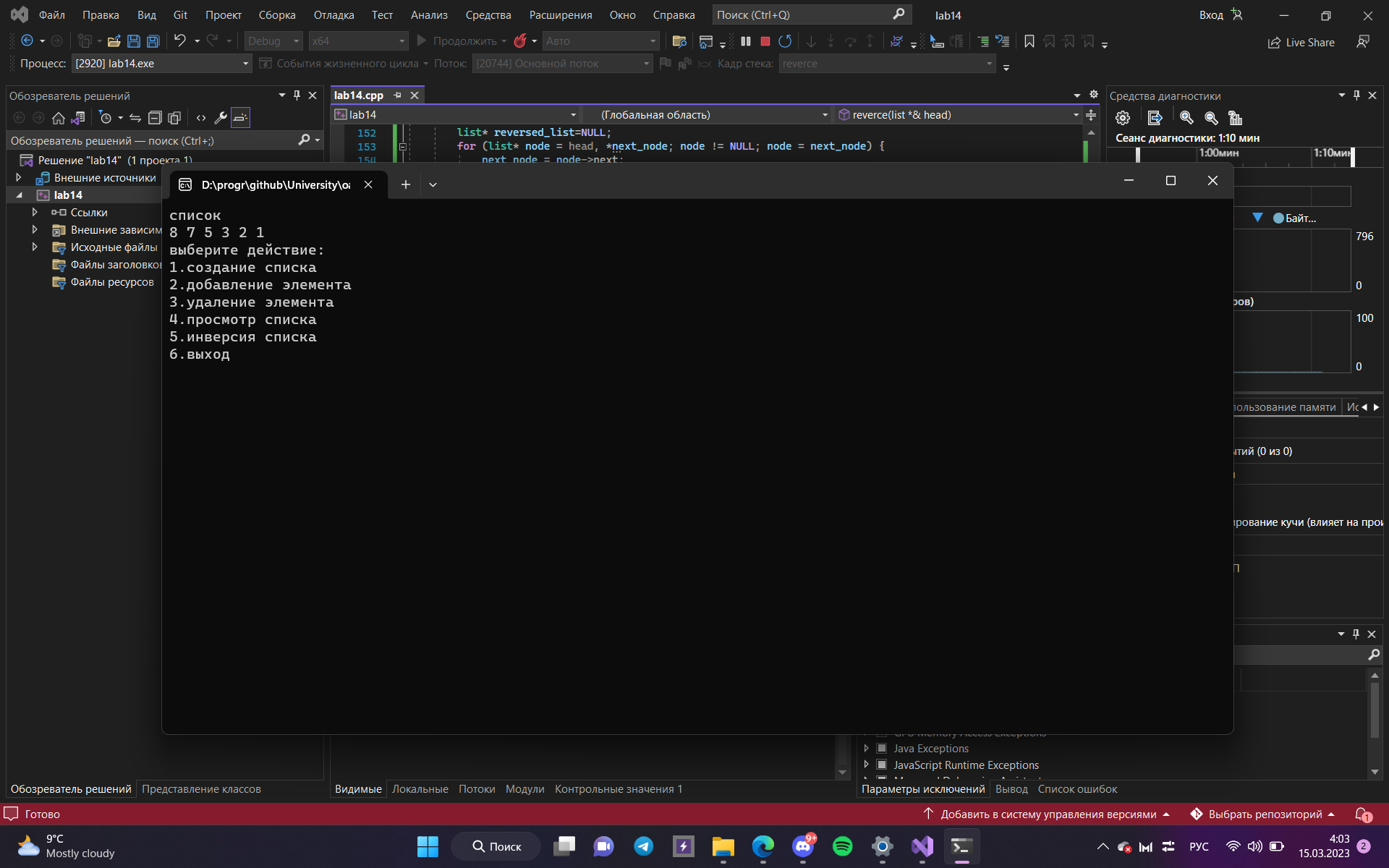
**}**

Результаты выполнения программы:









Задание 2:

Написать программу, которая вводит с клавиатуры список целых чисел, считает длину списка, затем, из элементов списка создает сбалансированное дерево и подсчитывает среднее арифметическое элементов дерева.

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

TreeNode(int x) : val(x), left(NULL), right(NULL) {}

};

TreeNode\* sortedArrayToBST(vector<int>& nums, int left, int right) {

if (left > right) {

return NULL;

}

int mid = left + (right - left) / 2;

TreeNode\* root = new TreeNode(nums[mid]);

root->left = sortedArrayToBST(nums, left, mid - 1);

root->right = sortedArrayToBST(nums, mid + 1, right);

return root;

}

double getAvg(TreeNode\* root) {

if (!root) {

return 0.0;

}

double sum = root->val;

double count = 1.0;

if (root->left) {

sum += getAvg(root->left);

count += 1.0;

}

if (root->right) {

sum += getAvg(root->right);

count += 1.0;

}

return sum / count;

}

int main() {

int n;

cout << "введите длину листа: ";

cin >> n;

vector<int> nums(n);

cout << "введите элементы: ";

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

cin >> nums[i];

}

sort(nums.begin(), nums.end());

TreeNode\* root = sortedArrayToBST(nums, 0, n - 1);

double avg = getAvg(root);

cout << "среднее дерева: " << avg << endl;

return 0;

}

Задание 3:

Написать программу, реализующую в виде списка представление многочлена P(x) = a0+a1x+...+aixi, где ai –вещественные числа, i – целые положительные числа.Пользователь должен иметь возможность произвольно добавлять элементы-слагаемые через меню. Реализовать процедуру взятия производной многочлена.

#include <iostream>

#include <cmath>

using namespace std;

// Структура для хранения слагаемого многочлена

struct Term {

double coeff; // коэффициент

int exp; // степень

Term\* next; // указатель на следующий элемент списка

};

// Функция для добавления нового слагаемого в список

void addTerm(Term\*& head, double coeff, int exp) {

Term\* newTerm = new Term;

newTerm->coeff = coeff;

newTerm->exp = exp;

newTerm->next = NULL;

if (head == NULL) {

head = newTerm;

}

else {

Term\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newTerm;

}

}

// Функция для вывода многочлена на экран

void displayPolynomial(Term\* head) {

if (head == NULL) {

cout << "0" << endl;

return;

}

int k = 0;

while (head != NULL) {

if (k > 0) {

cout << "+";

}

k++;

if (head->exp == 0) {

cout << head->coeff;

}

else if (head->exp == 1) {

cout << head->coeff << "x";

}

else {

cout << head->coeff << "x^" << head->exp;

}

head = head->next;

}

cout << endl;

}

// Функция для взятия производной многочлена

void differentiate(Term\*& head) {

if (head == NULL) {

cout << "Многочлен пуст." << endl;

return;

}

Term\* temp = head;

while (temp != NULL) {

temp->coeff = temp->coeff \* temp->exp;

temp->exp = temp->exp - 1;

if (temp->exp < 0) {

// удаление слагаемого с отрицательной степенью

if (temp == head) {

head = temp->next;

delete temp;

temp = head;

}

else {

Term\* prev = head;

while (prev->next != temp) {

prev = prev->next;

}

prev->next = temp->next;

delete temp;

temp = prev->next;

}

}

else {

temp = temp->next;

}

}

}

// Главная функция

int main() {

Term\* head = NULL;

int option = 0;

while (option != 3) {

cout << "Выберите действие:" << endl;

cout << "1. Добавить слагаемое в многочлен" << endl;

cout << "2. Взять производную многочлена" << endl;

cout << "3. Выйти из программы" << endl;

cout << "Ваш выбор: ";

cin >> option;

if (option == 1) {

double coeff;

int exp;

cout << "Введите коэффициент и степень слагаемого: ";

cin >> coeff >> exp;

addTerm(head, coeff, exp);

cout << "Слагаемое добавлено." << endl;

cout << "Текущий многочлен: ";

displayPolynomial(head);

}

else if (option == 2) {

differentiate(head);

cout << "Производная многочлена: ";

displayPolynomial(head);

}

else if (option == 3) {

// Освобождение памяти

Term\* temp;

while (head != NULL) {

temp = head->next;

delete head;

head = temp;

}

cout << "Программа завершена." << endl;

}

else {

cout << "Некорректный выбор. Попробуйте еще раз." << endl;

}

}

return 0;

}

Вывод: приобрел навык работы с динамической памятью и указателями на С/C++; изучил принципы работы с динамическими структурами данных: списками и деревьями.