МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2
по дисциплине «АиСД»
Тема: Иерархические списки

Студент гр. 9382 _____ Пя С.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Научиться создавать иерархические списки на основе структур. Ознакомиться с их реализацией с помощью рекурсии на примере языка C++, получить навыки в понимании и применении структур. Выполнить работу, способную производить вычисления в соответствии с заданием.

Основные теоретические положения.

Иерархический список представляет собой список, элементы, которых могут быть также списки и т.д. Иерархические списки позволяют хранить информацию в соответствии с подчинением одних данных другим.

Рекурсия – это определение, описание, изображение какого-либо объекта или процесса внутри самого этого объекта или процесса, то есть ситуация, когда объект является частью самого себя.

Задание

Вариант №25.

Символьное дифференцирование алгебраического выражения, рассматриваемого как функция от одной из переменных. На входе выражение в виде иерархического списка и переменная, по которой следует дифференцировать. На выходе – производная исходного выражения. После дифференцирования возможно упрощение выражения. Набор операций (функций), которые могут входить в выражение, определяется вариантом.

25) +, -, *, упростить после дифференцирования

Ход работы.

1) Разработан алгоритм:

Из выражения, записанного в виде иерархического списка (в польской записи) формируется иерархический список из структур, в которых записаны выражения слева от знака операции и справа. Вложенность списка зависит от

количества скобок. Исходя из формулы дифференцирования, сначала дифференцирование выражение разбивается на два дифференцированных выражения (в случае сложения и вычитания) или на два произведения из дифференцированных и недифференцированных выражений (в случае умножения). Если выражение простое, то есть не имеет знаков плюса или минуса, то его можно дифференцировать (в нашем случае). Мы доходим до простого выражения, дифференцируем его, и благодаря рекурсии выполняем соответствующие действия на прошлом уровне списка. Затем упрощаем выражение.

Если список невложенный, то производится обычное дифференцирование с последующим выводом на экран.

Были устранены все утечки памяти, написана реакция на не открытие файла и на пустую строку.

Предусмотрен механизм простейшего взаимодействия с пользователем, позволяющий понять алгоритм исполнения программы, с помощью сообщений о нахождении в определенной функции, вывода промежуточных данных, отображения арифметических операций и глубины рекурсии.

2) Использованы функции:

1. main

Сигнатура: int main().

Назначение: является основной функцией и телом программы.

Описание аргументов: без аргументов.

Возвращаемое значение: Функция возвращает 0.

2. createList

Сигнатура: node* createList(std::fstream& fin).

Назначение: создает иерархический список.

Описание аргументов: fin, переменная, дающая доступ к считываемому файлу.

Возвращаемое значение: объект структуры shift, являющийся хэдером.

3. diff

Сигнатура: char* diff(char*& str, char x).

Назначение: Дифференцирует простейшее выражение.

Описание аргументов: адрес указателя на строку str, символ x, являющийся переменной, относительно которой будет производиться операция.

Возвращаемое значение: указатель на строку str, в которой будет готовый результат.

4. differentiateList

Сигнатура: void differentiateList(node* shift, char x).

Назначение: Производит дифференцирование всего списка.

Описание аргументов: ссылка на структуру shift, из которой будут браться выражения и символ x, являющийся переменной, относительно которой будет производиться операция.

Возвращаемое значение: Функция ничего не возвращает.

5. simplify

Сигнатура: char* simplify(char* t, node* shift, char x).

Назначение: составляет из правого и левого выражения одно и упрощает его.

Описание аргументов: указатель на строку t, shift – структура, откуда берутся выражения, x – переменная, относительно которой производится операция.

Возвращаемое значение: указатель на строку с упрощенным выражением.

6. makeMinus

Сигнатура: char* makeMinus(char* str1).

Назначение: записывает выражение как отрицательное.

Описание аргументов: указатель на строку str1.

Возвращаемое значение: указатель на строку str1, в которой будет отрицательное выражение.

7. deleteBrackets

Сигнатура: char* deleteBrackets(char*& str1).

Назначение: удаляет скобки у простейшего отрицательного выражения.

Описание аргументов: адрес указателя на строку str1.

Возвращаемое значение: указатель на строку str1, в которой будет отрицательное выражение без скобок.

8. brackets

Сигнатура: char* brackets(char*& str).

Назначение: упрощает выражение путем избавления от скобок и перемножения.

Описание аргументов: адрес указателя на строку str.

Возвращаемое значение: указатель на строку str1, в которой будет упрощенное выражение без скобок.

9. compare

Сигнатура: int compare(const void * x1, const void * x2).

Назначение: сравнивает два символа, служит для сортировки.

Описание аргументов: указатель на первый и указатель на второй символы x1 и x2.

Возвращаемое значение: < 0 , если первый символ меньше второго (алфавитный порядок) и наоброт, 0, если равны.

10. simplifyEnd

Сигнатура: char* simplifyEnd(char* str).

Назначение: упрощает конечное выражение.

Описание аргументов: указатель на строку str.

Возвращаемое значение указатель на строку str, в которой будет упрощенное выражение без скобок.

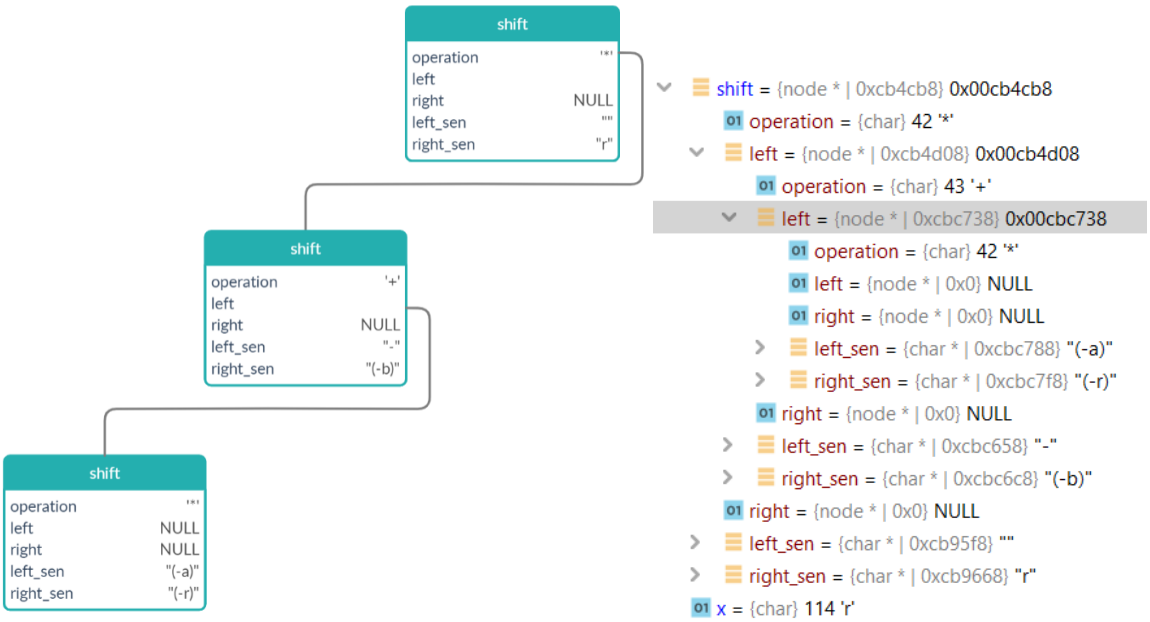
3) Описание главной рекурсивной функции:

Реализация:

В зависимости от знака операции происходят дальнейшие действия. Проходит по структурам сначала левой части, пока не дойдет до конца, затем правой. Как только встречает простейшее выражение, дифференцирует его с помощью функции diff, также с правым выражением. Затем при умножении дифференцированные выражения записываются в произведения с недифференцированными. После происходит возвращение на прошлый уровень рекурсии.

Выражение с нижнего уровня, будучи упрощенным и собранным в одно, записывается в выражение верхнего, и структура удаляется. Так происходит вплоть до начала списка, затем хэдер упрощается и выводится на экран.

Пример работы программы.



Входные данные	Выходные данные
`(*(+-(*-a r) b) r) r`	in left differentiateList:* in left differentiateList:+ in left differentiateList:* 0 out left differentiateList:* end in right differentiateList:* -1 out right differentiateList:* end (-0*r+a*1) out left differentiateList:+ in right differentiateList:+ 0 out right differentiateList:+ end ((-0*r+a*1)+0)

	out left differentiateList:* in right differentiateList:* 1 out right differentiateList:* end (((-0*r+a*1)+0)*r+(1*(-1*a*r)-b)) end before simplify b*(-1) end after simplify

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	`(+-(*-78 x) x) x`	-1	Проверка на корректность работы с отрицательными значениями
2.	`(*(+x 5) (-3 7)) x`	-4	Проверка на корректность работы частично без переменной
3.	`(*(+-4 x) (-x 3)) x`	x*(-2)-1	Проверка на корректность работы с перемножением переменных
4.	`(*(*-x x) (*-x x)) x`	x*x*x*4	Проверка на корректность работы с множественным перемножением переменных

5.	`(+-(*-a b) x) x`	-1	Проверка на корректность работы с буквенной частью
6.	`(*(+-(*-a r) b) r) r`	b*(-1)	Проверка на корректность работы с разными переменными
7.	`787 x`	0	Проверка на корректность работы без иерархического списка
8.		Empty string!	Пустая строка

Правильный ввод выражения для тестирования:

Простейшие выражения отделяются друг от друга пробелами, минусы, принадлежащие одному выражению не ставятся, так как это ломает польскую запись. Сначала пишутся скобки, если выражение сложное, затем символ или два (в случае отрицания выражения в скобках), после без пробела левое выражение, затем правое. После выражения через пробел пишется переменная, относительно которой будет вестись дифференцирование.

Выводы.

В ходе работы была освоена реализация иерархического списка на основе рекурсии и структур, отработано понимание его применения, и отработаны навыки письма в C++.

Код программы можно найти в приложении А.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
/*#define   CRTDBG_MAP_ALLOC
#include <crtdbg.h>
#define DEBUG_NEW new(_NORMAL_BLOCK, __FILE__, __LINE__)
#define new DEBUG_NEW*/
#include "Header.h"


int main() {
    std::fstream fin(".//Tests//8.txt");//файл для считывания выражений
    if (!fin.is_open())
        std::cout << "File is not open\n";
    else {
        char *f = new char[50]();
        f[0] = fin.get();
        if (f[0] == '(') {
            node *head = createList(fin); //создание иерархического списка
            while ((f[0] = fin.get()) != ' ') {
            }
            f[0] = fin.get();//переменная, относительно которой производится
дифференцирование
            fin.close();
            differentiateList(head, f[0]); //дифференциование списка
            char* str1 = new char[50]();
            str1 = simplifyEnd(simplify(str1, head, f[0]));//упрощение списка
            std::cout << str1 << " end after simplify\n";//вывод результата на
экран
            delete[] str1;
            delete head;
        } else {
            if (!fin) {
                std::cout << "Empty string!";
                return 0;
            } else {
                int i = 0;
                while (f[i] != ' ') {
                    i++;
                    f[i] = fin.get();
                }
                f[i] = '\0';
                char x = fin.get();
                std::cout << diff(f, x) << " end after sim-
plify\n";//дифференцирование выражения с дальнейшим выводом на экран
            }
        }
        delete[] f;
    }
   /* if (_CrtDumpMemoryLeaks())
        printf("\nMemory leack deteckted\n");
    else
        printf("\nMemory is not leack deteckted\n");*/
    return 0;
}
```

Название файла: Header.cpp

```
#ifndef UNTITLED3_HEADER_H
#define UNTITLED3_HEADER_H
#include <iostream>
```

```
#include <fstream>
#include <cstring>
#include <cctype>
#include <cstdlib>
typedef struct node {//структура, с помощью которой реализуется иерархический
список
    char operation;//знак между выражениями
    node* left;//левый указатель на следующую структуру
    node* right;//правый указатель на следующую структуру
    char* left_sen;//левое выражение
    char* right_sen;//правое выражение
    node() {//конструктор
        left_sen = new char[50]();
        right_sen = new char[50]();
        left = nullptr;
        right = nullptr;
    }
    ~node() {//деструктор
        left = nullptr;
        right = nullptr;
        delete[] left_sen;
        delete[] right_sen;
    }
} node;
char* makeMinus(char* str1) {//превращает выражение в отрицательное
    char *str = new char[50]();
        strcpy(str, "(-");
    int m = 0;
    for (int i = 0;i < strlen(str1);i++) {
        if (str1[i] == '+') {
            strncat(str, str1, i - m);
            m = i;
            strcat(str, "-");
        }
        if (str1[i] == '-') {
            strncat(str, str1, i - m);
            m = i;
            strcat(str, "+");
        }
    }
    strncat(str, str1, strlen(str1) - m);
    strcat(str, ")");
    delete[] str1;
    str1 = new char[50]();
    strcpy(str1, str);
    delete[] str;
    return str1;
}
char* deleteBrackets(char*& str1) {//удаление скобок из отрицательного простого
выражения
    int i = 0, p = 0;
    char *str2 = new char[50]();
    if (strlen(str1) >= 4) {
        while (i < strlen(str1)) {
            if (str1[i] == '(' && str1[i + 1] == '-') {
                i++;
                p = 2;
                while (str1[i] != ')') {
                    strncat(str2, &str1[i], 1);
                    i++;
                    if (str1[i] == '*' || str1[i] == '+' || str1[i] == '-') {
                        p = 1;
                        char *str3 = new char[20]();
                        for (int u = 0; u < strlen(str1); u++) {
```

```
                            if (str1[u] == '+' && str1[u + 1] == '-') {
                                strcat(str3, "-");
                                u += 2;
                            }
                            if (str1[u] == '-' && str1[u + 1] == '-') {
                                strcat(str3, "+");
                                u += 2;
                            }
                            strncat(str3, &str1[u], 1);
                        }
                        delete[] str2;
                        str2 = new char[50]();
                        strcat(str2, str3);
                        delete[] str3;
                        break;
                    }
                }
                if (p == 1)
                    break;
                if (p == 2) {
                    p = 0;
                    i++;
                    continue;
                }
            }
            strncat(str2, &str1[i++], 1);
        }
    } else strcpy(str2, str1);
    delete[] str1;
    str1 = new char[50]();
    strcpy(str1, str2);
    delete[] str2;
    return str1;
}
char* brackets(char*& str) {//раскрытие скобок и удаление лишних
    char *str1 = new char[50]();
    int count = 0, m = 0;
    for (int i = 0; i < strlen(str); i++) {
        if (str[i] == '(')//подсчет скобок
            count++;
        if (str[i] == ')')
            count--;
        if ((str[i] == '+' || str[i] == '-' || i == (strlen(str) - 1)) && count
== 0) {//деление выражения на отдельные со скобками
            if (m != 0)
                strncat(str1, &str[m - 1], 1);
            char *str2 = new char[50]();
            if (i != (strlen(str) - 1))
                strncat(str2, str + m, i - m);
            else
                strncat(str2, str + m, i - m + 1);
            if (strchr(str2, '(')) {
                int n = 0, n1 = 0; bool flag = false;
                char *t_str = new char[50]();
                for (int i = 0; i < strlen(str2); i++) {//упрощение выражения с
помощью рекурсии, начиная с одних парных скобок
                    if (str2[i] == '(') {
                        n++;
                        if (n == 1)
                            n1 = i;
                        if (n > 1) {
                            flag = true;
                            n = 0;
                            char* qpch = new char[50]();
```

```
                        int m = 1; i = n1 + 1;
                        while (!(str2[i] == ')' && m == 1)) {
                            if (str2[i] == '(')
                                m++;
                            if (str2[i] == ')')
                                m--;
                            strncat(qpch, &str2[i], 1);
                            i++;
                        }
                        strcat(qpch, "\0");
                        strcat(t_str, "(");
                        strcat(t_str, brackets(qpch));//рекурсия
                        strcat(t_str, ")");
                        delete[] qpch;
                    }
                } else if (n == 0)
                    strncat(t_str, &str2[i], 1);
            }
            if (!flag)
                strcpy(t_str, str2);
            char *t_pch = new char[50]();//раскрытие скобок путем
перемножения каждого члена на множитель
            char *mul_pch = new char[50]();//результат перемножения будет
лежать в t pch
            if (m != 0 && str[m - 1] == '-')//если перед скобками стоял
минус, раскрываем, инветируя знаки
                strcpy(mul_pch, "-1");
            char *pch = strtok(t_str, ")(");
            while (pch != NULL) {
                if (!strcmp(pch, "*"))
                    pch = strtok(NULL, ")(");
                if (pch[0] == '*' || pch[strlen(pch) - 1] == '*')
{//записываем множители
                    if (strcmp(mul_pch, "")) {
                        strcat(mul_pch, "*");
                    }
                    if (pch[0] == '*')
                        strncat(mul_pch, pch + 1, strlen(pch) - 1);
                    else
                        strncat(mul_pch, pch, strlen(pch) - 1);
                } else if (!strcmp(t_pch, "")) {//выражение со скобками
                    strcat(t_pch, pch);
                } else {//если выражений со скобками оказалось несколько,
перемножаем их
                    char *p_str = new char[50](); int k = 0;
                    for (int i = 0; i < strlen(t_pch); i++) {
                        if (t_pch[i] == '+' || (t_pch[i] == '-' && i != 0)
||
                            i == (strlen(t_pch) - 1)) {
                            char *k1_pch = new char[50]();
                            if (k != 0)
                                strncat(p_str, &t_pch[k - 1], 1);
                            else if (str[m - 1] == '-') {
                                strcat(k1_pch, "-");
                            }
                            if (i != (strlen(t_pch) - 1))
                                strncat(k1_pch, t_pch + k, i - k);
                            else
                                strncat(k1_pch, t_pch + k, i - k + 1);
                            int n = 0;
                            for (int y = 0; y < strlen(pch); y++) {
                                if (pch[y] == '+' || (pch[y] == '-' && y !=
0) ||
                                    y == (strlen(pch) - 1)) {
```

```
                                        if (n != 0)
                                            strncat(p_str, &pch[n - 1], 1);
                                        char *k2_pch = new char[50]();
                                        if (y != (strlen(pch) - 1))
                                            strncat(k2_pch, pch + n, y - n);
                                        else
                                            strncat(k2_pch, pch + n, y - n + 1);
                                        strcat(p_str, k2_pch);
                                        strcat(p_str, "*");
                                        strcat(p_str, k1_pch);
                                        delete[] k2_pch;
                                        n = y + 1;
                                    }
                                }
                                delete[] k1_pch;
                                k = i + 1;
                            }
                        }
                        delete[] t_pch;
                        t_pch = new char[50](); strcat(t_pch, p_str); delete[]
p_str;
                    }
                    if (strcmp(mul_pch, "") && strcmp(t_pch, "")) {
                        char *str1 = new char[50](); int m = 0;
                        for (int i = 0; i < strlen(t_pch); i++) {
                            if (t_pch[i] == '+' || (t_pch[i] == '-' && i != 0)
||
                                i == (strlen(t_pch) - 1)) {
                                if (m != 0)
                                    strncat(str1, &t_pch[m - 1], 1);
                                char *k_pch = new char[50]();
                                if (i != (strlen(t_pch) - 1))
                                    strncat(k_pch, t_pch + m, i - m);
                                else
                                    strncat(k_pch, t_pch + m, i - m + 1);
                                strcat(str1, k_pch);
                                strcat(str1, "*");
                                strcat(str1, mul_pch);
                                delete[] k_pch;
                                m = i + 1;
                            }
                        }
                        delete[] mul_pch;
                        delete[] t_pch;
                        t_pch = new char[50](); mul_pch = new char[50]();
                        strcat(t_pch, str1);
                        delete[] str1;
                    }
                    pch = strtok(NULL, ")(");
                }
                strcat(str1, t_pch);//результат записываем в str1
                delete[] t_pch; delete[] t_str; delete[] mul_pch;
            } else {
                strcat(str1, str2);//если в выражении не было скобок изначально
            }
            delete[] str2;
            m = i + 1;
        }
    }
    delete[] str;
    str = new char[50]();
    strcpy(str, str1);
    delete[] str1;
    return str;
```

```
}
int compare(const void * x1, const void * x2) { return ( *(char*)x1 - *(char*)x2
);}//сравнивание символов для qsort
char* simplifyEnd(char* str) {std::cout << str << " end before sim-
plify\n";//финальное упрощение
    char *str1 = new char[50]();
    if (str[0] == '(')
        strncat(str1, str + 1, strlen(str) - 2);
    else
        strcat(str1, str);
    str1 = brackets(str1);//избавление от скобок
    char* pstr1 = new char[50]();
    strcpy(pstr1, str1);
    delete[] str1;
    str1 = new char[50]();
    delete[] str;
    str = new char[50]();
    int m = 0;
    for (int i = 0; i < strlen(pstr1); i++) {//удаление лишних знаков
        if (pstr1[i] == '+' && pstr1[i + 1] == '-') {
            strncat(str1, pstr1 + m, i - m);
            m = i + 2;
            strcat(str1, "-");
        }
        if (pstr1[i] == '-' && pstr1[i + 1] == '-') {
            strncat(str1, pstr1 + m, i - m);
            m = i + 2;
            strcat(str1, "+");
        }
    }
    strncat(str1, pstr1 + m, strlen(pstr1) - m);
    delete[] pstr1;
    char *rep = new char[50]();
    m = 0;
    int digit = 1;
    for (int i = 0; i < strlen(str1); i++) {//поиск и сложение/вычитание
возможных выражений
        if (str1[i] == '+' || (str1[i] == '-' && i != 0) || i == (strlen(str1) -
1)) {
            int tdigit = 1;
            char *t_str = new char[20]();
            char *k_str = new char[20]();
            if (i != (strlen(str1) - 1))
                strncat(k_str, str1 + m, i - m);
            else
                strncat(k_str, str1 + m, i - m + 1);
            strcpy(t_str, k_str);
            char* pch = strtok(k_str, "*");
            char *t_pch = new char[50]();
            int value = 1;
            while (pch != NULL) {
                bool flag = false;
                for (int y = 0; y < strlen(pch); y++) {//запись буквенной части
                    if (!(isdigit(pch[y]) || pch[y] == '-')) {
                        flag = true;
                    }
                }
                if (!flag) {
                    value *= atoi(pch);
                } else {
                    tdigit = 0;
                    strcat(t_pch, pch);
                    strcat(t_pch, "*");
                }
```

```
                pch = strtok(NULL, "*");
            }
            if (str1[m - 1] == '-')
                    value *= -1;
            int n = i + 1;
            char *cm1 = new char[50]();
            strcpy(cm1, t_pch);
            std::qsort(cm1, strlen(cm1), sizeof(char), compare);//сортировка для
сравнения буквенной части
            for (int q = i + 1; q < strlen(str1); q++) {
                if (str1[q] == '+' || (str1[q] == '-' && q != 0) || q ==
(strlen(str1) - 1)) {//перебор и сравнение буквенных частей
                    char *k_str = new char[20]();
                    char* l_pch = new char[50]();
                    if (q != (strlen(str1) - 1))
                        strncat(k_str, str1 + n, q - n);
                    else
                        strncat(k_str, str1 + n, q - n + 1);
                    if (strpbrk(k_str, t_pch)) {
                        char *pch = strtok(k_str, "*");
                        int t_value = 1;
                        while (pch != NULL) {//запись буквенной части
                            bool flag = false;
                            for (int y = 0; y < strlen(pch); y++) {
                                if (!(isdigit(pch[y]) || pch[y] == '-')) {
                                    flag = true;
                                }
                            }
                            if (!flag) {
                                t_value *= atoi(pch);
                            } else {
                                strcat(l_pch, pch);
                                strcat(l_pch, "*");
                            }
                            pch = strtok(NULL, "*");
                        }
                        char *cm2 = new char[50]();
                        strcpy(cm2, l_pch);
                        std::qsort(cm2, strlen(cm2), sizeof(char), com-
pare);//сортировка буквенной части
                        if (!strcmp(cm1,cm2)) {//сравнение буквенных частей для
дальнейшего сложения
                            if (str1[n - 1] == '-')//в случае совпадения
сложение/вычитание коэффициентов
                                value -= t_value;
                            else
                                value += t_value;
                        }
                        delete[] cm2;
                    }
                    delete[] k_str; delete[] l_pch;
                    n = q + 1;
                }
            }
            m = i + 1;
            if (tdigit == 1) {//если буквенная часть отсутствовала
                digit = 0;
                if (value > 0 && (strcmp(str, "")))
                    strcat(str, "+");
                char *r = new char[20]();
                itoa(value, r, 10);
                strcat(str, r);
                delete[] r;
            }
```

```
            if (digit == 0) {
                delete[] t_pch; delete[] t_str; delete[] cm1; delete[] k_str;
continue;
            }
            char* t_rep = new char[40]();
            strcpy(t_rep, rep);
            char *rep_pch = strtok(t_rep, " ");
            bool t_flag = false;
            while (rep_pch != NULL) {//сравение буквенной части с прошлыми для
избавления повторений
                if (!strcmp(rep_pch, cm1)) {
                    t_flag = true;
                }
                rep_pch = strtok(NULL, " ");
            }
            delete[] t_rep;
            if (!t_flag) {//запись результата
                    strcat(rep, cm1);//запись буквенной части к прошлым
                    strcat(rep, " ");
                if (!value == 0) {
                    if (strcmp(str, ""))
                        strcat(str, "+");
                    strcat(str, t_pch);
                    char *r = new char[20]();
                    if (value < 0) {
                        value *= -1;
                        itoa(value, r, 10);
                        r = makeMinus(r);
                    } else
                        itoa(value, r, 10);
                    strcat(str, r);
                    delete[] r;
                }
            }
            delete[] t_pch;
            delete[] t_str;
            delete[] cm1;
            delete[] k_str;
        }
    }
    delete[] str1;
    delete[] rep;
    return str;
}
node* createList(std::fstream& fin) {//создание иерархического списка
    node* shift = new node();
    int flag = 0; //0 = неотрицательное выражение, 1 = случай с умножением, 2 =
остальное
    char c = fin.get();
    shift->operation = c;//запись знака
    c = fin.get();
    if (c == '-') {//отрицательное выражение
        c = fin.get();
        if (shift->operation == '*')
            flag = 1;
        else
            flag = 2;
    }
    if (c == '(') {//создание следующей структуры
        shift->left = createList(fin);
        c = fin.get();
    } else {
        if (isdigit(c)) {
            while (isdigit(c)) {
```

```
                strncat(shift->left_sen, &c, 1);
                c = fin.get();
            }
        }
        else {
            shift->left_sen[0] = c;
            c = fin.get();
        }
    }
    c = fin.get();
    if (c == '(') {
        shift->right = createList(fin);
        c = fin.get();
    } else {
        if (isdigit(c)) {
            while (isdigit(c)) {
                strncat(shift->right_sen, &c, 1);
                c = fin.get();
            }
        } else {
            shift->right_sen[0] = c;
            c = fin.get();
        }
    }
    if (flag != 0) {//если выражения еще нет, то пишется '-', иначе сразу
записывается отрицательное выражение
        if (strcmp(shift->left_sen, "") == 0 && flag != 1)
            strcpy(shift->left_sen, "-");
        else {
            shift->left_sen  = makeMinus(shift->left_sen);
        }
        if (strcmp(shift->right_sen, "") == 0)
            strcpy(shift->right_sen, "-");
        else {
            shift->right_sen = makeMinus(shift->right_sen);
        }
    }
    return shift;
}
char* simplify(char* t, node* shift, char x) {//упрощение выражения и
преобразование в строку
    if (!strcmp(shift->left_sen, "-")) {//получение левого отрицательного и
неотрицательного выражения
        delete[] shift->left_sen;
        shift->left_sen = new char[50]();
        shift->left_sen = makeMinus(simplify(shift->left_sen, shift->left, x));
    }
    else if (!strcmp(shift->left_sen, "")) {
        shift->left_sen = simplify(shift->left_sen, shift->left, x);
    }
    if (!strcmp(shift->right_sen, "-")) {//получение правого отрицательного и
неотрицательного выражения
        delete[] shift->right_sen;
        shift->right_sen = new char[50]();
        shift->right_sen = makeMinus(simplify(shift->right_sen,shift->right,
x));
    }
    else if (!strcmp(shift->left_sen, "")) {
        shift->right_sen = simplify(shift->right_sen, shift->right, x);
    }
    int flag1 = 0, flag2 = 0, here = 0;
    for (int i = 0; i < strlen(shift->left_sen); i++)//подсчет переменной,
относительно которой производится дифференцирование
        if (shift->left_sen[i] == x)
```

```
            flag1++;
        else if (isalpha(shift->left_sen[i])) {//если в левом выражении
присутствует другие буквы
            here = 1;
        }
    for (int i = 0; i < strlen(shift->right_sen); i++)
        if (shift->right_sen[i] == x) {
            flag2++;
        } else if (isalpha(shift->right_sen[i])) {//если в правом выражении
присутствует другие буквы
            here = 2;
        }
    int value = 0;
    shift->left_sen = deleteBrackets(shift->left_sen);//удаление скобок для
отрицательных простых выражений
    shift->right_sen = deleteBrackets(shift->right_sen);
    if (flag1 == 0 && flag2 == 0 && here == 0) {//если оба выражения оказались
числовыми
        switch (shift->operation) {
            case '+':
                value = atoi(shift->left_sen) + atoi(shift->right_sen);
                break;
            case '-':
                value = atoi(shift->left_sen) - atoi(shift->right_sen);
                break;
            case '*':
                value = atoi(shift->left_sen) * atoi(shift->right_sen);
                break;
        }
        itoa(value, t, 10);
    } else if (flag1 == flag2 && (shift->operation == '+' || shift->operation ==
'-')) {//если количество переменной совпали, и знак равен + или -
        char *left_sen = new char[50]();
        char *tleft_sen = new char[50]();
        char* sym = new char[3](); strcat(sym, "*"); strncat(sym, &x, 1);
        if (shift->left_sen[0] == '(')//избавление от скобок
            strncpy(left_sen, shift->left_sen + 1, strlen(shift->left_sen) - 2);
        else strcpy(left_sen, shift->left_sen);
        int digit_suml = 0, digit_sumr = 0;
        char *lt = new char[20]();//запись для буквенной части левого выражения
        char *rt = new char[20]();//правого выражения
        char *etc = new char[50]();
        int suml = 0;
        int sumr = 0, sumbr = 1, sumbl = 1;
        int m = 0;
        bool flag = false;
        for (int i = 0; i < strlen(left_sen); i++) {
            if (left_sen[i] == '*' && left_sen[i + 1] == x || left_sen[i + 1] ==
'*' && left_sen[i] == x) {
                char *tt_str = new char[20](); char *t_str = new char[20]();
                strncat(t_str, left_sen + m, i - m);
                bool minus = false;
                if (m != 0 && left_sen[m - 1] == '-') {//если выражение
отрицательное
                    minus = true;
                }
                i += 2;
                m = i;
                if (t_str[0] == '(') {//раскрытие скобок
                    for (int i = 0; i < strlen(t_str); i++) {
                        int g = 0;
                        if ((t_str[i] == '+' || t_str[i] == '-' || t_str[i] ==
')') && i != 1) {
                            if (g == 0 && t_str[1] == '-')
```

```
                                g++;
                            else if (g != 0 && t_str[g - 1] == '-' && minus) {
                                strcat(tt_str, "+");
                                g++;
                            } else if (g != 0 && t_str[g - 1] == '+' && minus) {
                                strcat(tt_str, "-");
                            } else if (g != 0 && !minus) {
                                strncat(tt_str, &t_str[g - 1], 1);
                            }
                            strncat(tt_str, t_str + g, i - g);//запись
                            g = i + 1;
                            strcat(tt_str, "*"); strncat(tt_str, &x, 1);
                        }
                    }
                } else {
                    //копирование оставшиеся символы
                    strcat(tt_str, t_str);
                    strcat(tt_str, "*"); strncat(tt_str, &x, 1);
                }
                strcat(tleft_sen, tt_str);
                delete[] tt_str; delete[] t_str;
            }
        }
        strncat(tleft_sen, left_sen + m, strlen(left_sen) - m);
        delete[] left_sen;
        left_sen = new char[50]();
        strcpy(left_sen, tleft_sen);
        delete[] tleft_sen;
        m = 0;
        for (int i = 0; i < strlen(left_sen); i++) {
            if (left_sen[i] == '+' || (left_sen[i] == '-' && i != 0) || (i ==
(strlen(left_sen) - 1) && flag)) {//упрощение выражения относительно переменной
                flag = true;
                char *t_str = new char[20]();
                if (i == (strlen(left_sen) - 1) || m != 0)
                    strncat(t_str, left_sen + m, i - m + 1);
                else
                    strncat(t_str, left_sen + m, i - m);
                if (strchr(t_str, x)) {
                    char *l_pch = new char[20]();
                    char* r_pch = strtok(t_str, sym);
                    strcpy(l_pch, r_pch);//левый множитель
                    r_pch = strtok(NULL, sym);//правый множитель
                    if (r_pch != NULL) {
                        if (!(strcmp(l_pch, "0") == 0 || strcmp(l_pch, "-0") ==
0) && !(strcmp(r_pch, "0") == 0 || strcmp(r_pch, "-0") == 0)) {
                            int lvalue = atoi(l_pch);
                            int rvalue = atoi(r_pch);
                            if (lvalue && rvalue) {//если без букв
                                suml *= lvalue * rvalue;
                            } else if (lvalue) {//запись букв
                                strcat(lt, r_pch);
                                strcat(lt, "*");
                                sumbl *= lvalue;
                            } else if (rvalue) {
                                strcat(lt, l_pch);
                                strcat(lt, "*");
                                sumbl *= rvalue;
                            } else {
                                strcat(lt, r_pch);
                                strcat(lt, "*");
                                strcat(lt, l_pch);
                                strcat(lt, "*");
                            }
```

```
                            delete[] l_pch;
                        }
                    } else {//если переменной нет
                        if (!(strcmp(l_pch, "0") == 0 || strcmp(l_pch, "-0") ==
0)) {
                            int lvalue = atoi(l_pch);
                            if (lvalue) {
                                if (m == 0 || left_sen[m - 1] == '+')
                                    suml += lvalue;
                                else if (left_sen[m - 1] == '-')
                                    suml -= lvalue;
                            } else {
                                strcat(lt, l_pch);
                                strcat(lt, "*");
                            }
                        }
                    }
                    delete[] l_pch;
                } else {//если переменной нет
                    if (strchr(t_str, '*')) {
                        char *r_pch = strtok(t_str, "*");
                        char *l_pch = new char[50]();
                        strcat(l_pch, r_pch);
                        r_pch = strtok(NULL, "*");
                        if (!(strcmp(l_pch, "0") == 0 || strcmp(l_pch, "-0") ==
0) &&
                            !(strcmp(r_pch, "0") == 0 || strcmp(r_pch, "-0") ==
0)) {
                            int lvalue = atoi(l_pch);
                            int rvalue = atoi(r_pch);
                            if (lvalue && rvalue)
                                digit_suml += lvalue * rvalue;
                            else {
                                strncat(etc, &shift->operation, 1);
                                strcat(etc, l_pch);
                                strcat(etc, "*");
                                strcat(etc, r_pch);
                            }
                        }
                        delete[] l_pch;
                    } else if (!(strcmp(t_str, "0") == 0 || strcmp(t_str, "-0")
== 0)) {
                        int lvalue = atoi(t_str);
                        if (lvalue) {
                            if (m == 0 || left_sen[m - 1] == '+')
                                digit_suml += lvalue;
                            if (left_sen[m - 1] == '-')
                                digit_suml -= lvalue;
                        } else {
                            strcat(lt, t_str);
                            strcat(lt, "*");
                        }
                    }
                }
                delete[] t_str;
                m = i + 1;
            }
        }
        if (!flag) {//если не было вхождения в прошлый цикл
            char *r_pch = strtok(left_sen, sym);
            char *l_pch = new char[50]();
            strcat(l_pch, r_pch);
            r_pch = strtok(NULL, sym);
            if (r_pch != NULL) {
```

```
                if (!(strcmp(l_pch, "0") == 0 || strcmp(l_pch, "-0") == 0) &&
                    !(strcmp(r_pch, "0") == 0 || strcmp(r_pch, "-0") == 0)) {
                    int lvalue = atoi(l_pch);
                    int rvalue = atoi(r_pch);
                    if (lvalue && rvalue)
                        digit_suml += lvalue * rvalue;
                    else {
                        strncat(etc, &shift->operation, 1);
                        strcat(etc, l_pch);
                        strcat(etc, "*");
                        strcat(etc, r_pch);
                    }
                }
            } else {//без переменной
                if (!(strcmp(l_pch, "0") == 0 || strcmp(l_pch, "-0") == 0)) {
                    int lvalue = atoi(l_pch);
                    if (lvalue) {
                        suml = lvalue;
                    } else {
                        strcat(lt, l_pch);
                        strcat(lt, "*");
                    }
                }
            }
            delete[] l_pch;
        }
        delete[] left_sen;
        char *right_sen = new char[50]();
        m = 0, flag = false;
        char *tright_sen = new char[50]();
        if (shift->right_sen[0] == '(')//для правого выражения
            strncpy(right_sen, shift->right_sen + 1, strlen(shift->right_sen) -
2);
        else strcpy(right_sen, shift->right_sen);
        for (int i = 0; i < strlen(right_sen); i++) {
            if (right_sen[i] == '*' && right_sen[i + 1] == x) {
                char *tt_str = new char[20]();
                char *t_str = new char[20]();
                strncat(t_str, right_sen + m, i - m);
                bool minus = false;
                if (m != 0 && right_sen[m - 1] == '-') {
                    minus = true;
                }
                i += 2;
                m = i;
                if (t_str[0] == '(') {
                    //реализовать скобки
                    for (int i = 0; i < strlen(t_str); i++) {
                        int g = 0;
                        if ((t_str[i] == '+' || t_str[i] == '-' || t_str[i] ==
')') && i != 1) {
                            if (g == 0 && t_str[1] == '-')
                                g++;
                            else if (g != 0 && t_str[g - 1] == '-' && minus) {
                                strcat(tt_str, "+");
                                g++;
                            } else if (g != 0 && t_str[g - 1] == '+' && minus) {
                                strcat(tt_str, "-");
                            } else if (g != 0 && !minus) {
                                strncat(tt_str, &t_str[g - 1], 1);
                            }
                            strncat(tt_str, t_str + g, i - g);
                            g = i + 1;
                            strcat(tt_str, sym);
```

```
                        }
                    }
                } else {
                    strcat(tt_str, t_str);
                    strcat(tt_str, sym);
                }
                strcat(tright_sen, tt_str);
                delete[] tt_str;
                delete[] t_str;
            }
        }
        strncat(tright_sen, right_sen + m, strlen(right_sen) - m);
        delete[] right_sen;
        right_sen = new char[50]();
        strcpy(right_sen, tright_sen);
        delete[] tright_sen;
        m = 0;
        for (int i = 0; i < strlen(right_sen); i++) {
            if (right_sen[i] == '+' || (right_sen[i] == '-' && i != 0) || (i ==
(strlen(right_sen) - 1) && flag)) {//НУЖНО ЗАДЕЙСТВОВАТЬ ФЛАГ
                flag = true;
                char *t_str = new char[20]();
                if (i == (strlen(right_sen) - 1) || m != 0)
                    strncat(t_str, right_sen + m, i - m + 1);
                else
                    strncat(t_str, right_sen + m, i - m);
                if (strchr(t_str, x)) {
                    char *l_pch = new char[20]();
                    char* r_pch = strtok(t_str, sym);
                    strcpy(l_pch, r_pch);
                    r_pch = strtok(NULL, sym);
                    if (r_pch != NULL) {//НУЖНО УБРАТЬ ВСЕ ИКСЫ С УМНОЖЕНИЕМ
                        if (!(strcmp(l_pch, "0") == 0 || strcmp(l_pch, "-0") ==
0) && !(strcmp(r_pch, "0") == 0 || strcmp(r_pch, "-0") == 0)) {
                            int lvalue = atoi(l_pch);
                            int rvalue = atoi(r_pch);
                            if (lvalue && rvalue) {//работа со знаками
                                sumr *= lvalue * rvalue;
                            } else if (lvalue) {
                                strcat(rt, r_pch);
                                strcat(rt, "*");
                                sumbr *= lvalue;
                            } else if (rvalue) {
                                strcat(rt, l_pch);
                                strcat(rt, "*");
                                sumbr *= rvalue;
                            } else {
                                strcat(rt, r_pch);
                                strcat(rt, "*");
                                strcat(rt, l_pch);
                                strcat(rt, "*");
                            }
                            delete[] l_pch;
                        }
                    } else {
                        if (!(strcmp(l_pch, "0") == 0 || strcmp(l_pch, "-0") ==
0)) {
                            int rvalue = atoi(l_pch);
                            if (rvalue) {//работа со знаками
                                if (m == 0 || right_sen[m - 1] == '+')
                                    sumr += rvalue;
                                else if (right_sen[m - 1] == '-')
                                    sumr -= rvalue;
                            } else {
```

```
                                strcat(rt, l_pch);
                                strcat(rt, "*");
                            }
                        }
                    }
                        delete[] l_pch;
                } else {
                    if (strchr(t_str, '*')) {
                        char *r_pch = strtok(t_str, "*");
                        char *l_pch = new char[50]();
                        strcat(l_pch, r_pch);
                        r_pch = strtok(NULL, "*");
                        if (!(strcmp(l_pch, "0") == 0 || strcmp(l_pch, "-0") ==
0) &&
                            !(strcmp(r_pch, "0") == 0 || strcmp(r_pch, "-0") ==
0)) {
                            int lvalue = atoi(l_pch);
                            int rvalue = atoi(r_pch);
                            if (lvalue && rvalue)
                                digit_sumr += lvalue * rvalue;
                            else {
                                strncat(etc, &shift->operation, 1);
                                strcat(etc, l_pch);
                                strcat(etc, "*");
                                strcat(etc, r_pch);
                            }

                        }
                    } else if (!(strcmp(t_str, "0") == 0 || strcmp(t_str, "-0")
== 0)) {
                        int rvalue = atoi(t_str);
                        if (rvalue) {
                            if (m == 0 || right_sen[m - 1] == '+')
                                digit_sumr += rvalue;
                            if (right_sen[m - 1] == '-')
                                digit_sumr -= rvalue;
                        } else {
                            strcat(rt, t_str);
                            strcat(rt, "*");
                        }
                    }
                }
                delete[] t_str;
                m = i + 1;
            }
        }
        if (!flag) {
            char *r_pch = strtok(right_sen, sym);
            char *l_pch = new char[50]();
            strcat(l_pch, r_pch);
            r_pch = strtok(NULL, sym);
            if (r_pch != NULL) {
                if (!(strcmp(l_pch, "0") == 0 || strcmp(l_pch, "-0") == 0) &&
                    !(strcmp(r_pch, "0") == 0 || strcmp(r_pch, "-0") == 0)) {
                    int lvalue = atoi(l_pch);
                    int rvalue = atoi(r_pch);
                    if (lvalue && rvalue)
                        digit_sumr += lvalue * rvalue;
                    else {
                        strncat(etc, &shift->operation, 1);
                        strcat(etc, l_pch);
                        strcat(etc, "*");
                        strcat(etc, r_pch);
                    }
```

```
            }
        } else {
            if (!(strcmp(l_pch, "0") == 0 || strcmp(l_pch, "-0") == 0)) {
                int rvalue = atoi(l_pch);
                if (rvalue) {
                    sumr = rvalue;
                } else {
                    strcat(rt, l_pch);
                    strcat(rt, "*");
                }
            }
        }
        delete[] l_pch;
    }
    delete[] right_sen;
    if (!strcmp(lt, rt)) {//если буквенные части совпадают
        value = suml + sumr;
        if (strcmp(lt, "") && value)
            strcat(t, "(");
        if (value) {
            char *r = new char[20]();
            itoa(value, r, 10);
            strcat(t, r);
            delete[] r;
        }
        if (strcmp(lt, "")) {//если буквенной части нет
            strcat(t, "+");
            strcat(t, lt);
            int value = sumbl + sumbr;
            char *r = new char[20]();
            itoa(value, r, 10);
            strcat(t, r);
            delete[] r;
        }
        if (strcmp(lt, "") && value)
            strcat(t, ")");
        for (int i = 0; i < flag1; i++)
            strcat(t, sym);
    } else {//если буквенные части разные
        std::strcat(t, "(");
        value = suml + sumr;
        if (value) {
            char *r = new char[20]();
            itoa(value, r, 10);
            strcat(t, "+");
            strcat(t, r);
            delete[] r;
        }
        if (strcmp(lt, "")) {
            strcat(t, lt);
            char *r = new char[20]();
            itoa(sumbl, r, 10);
            strcat(t, r);
            delete[] r;
        } else
            strcat(t, "1");
        std::strncat(t, &shift->operation, 1);
        if (strcmp(rt, "")) {
            strcat(t, rt);
            char *r = new char[20]();
            itoa(sumbr, r, 10);
            strcat(t, r);
            delete[] r;
        } else
```

```
                strcat(t, "1");
            std::strcat(t, ")"); strcat(t, sym);
            for (int i = 1; i < flag1; i++)
                strcat(t, sym);
        }
        value = digit_suml + digit_sumr;
        if (value != 0) {
            if (value > 0)
                strcat(t, "+");
            char *r = new char[20]();
            itoa(value, r, 10);
            strcat(t, r);
            delete[] r;
            if (strcmp(etc, "")) {
                strcat(t, "+");
                strncat(t, etc, strlen(etc) -1);
            }
        }
        delete[] lt;
        delete[] rt;
        delete[] etc;
        delete[] sym;
    } else if ((flag1 == 0 || flag2 == 0) && shift->operation == '*' && (flag2
!= 0 && shift->right_sen[0] == '(' || flag1 != 0 && shift->left_sen[0] == '('))
{
        char* str = new char[50]();//если переменных нет
        char* str2 = new char[50]();
        if (flag1 == 0 && here != 1) {
            value = atoi(shift->left_sen);
            strcpy(str,shift->right_sen);
        } else if (flag2 == 0 && here != 2) {
            value = atoi(shift->right_sen);
            strcpy(str,shift->left_sen);
        }
        if (value == 0) {
            strcpy(t, "0");
            delete[] str;
            delete[] str2;
            return t;
        }
        int num = 0, i = 0, m = 0;
        int n = 1; // 0 = не число, 1 = число, 2 = отрицательное не число
        while (str[i] != '+' && str[i] != '-') {
            if (i == 0 && str[i] == '(') {
                i++;
            }
            if (i == 1 && str[i] == '-') {
                i++;
                if (!isdigit(str[i]))
                    n = 2;
                else if (!isdigit(str[i]))
                    n = 0;
            } else n = 0;
            i++;
        }
        if (n == 1) {
            char *str1 = new char[20]();
            strncpy(str1, str + 1, i -1);
            strcpy(str2, "(");
            num = atoi(str1);
            num *= value;
            itoa(num, str1, 10);
            strcat(str2, str1);
            delete[] str1;
```

```
        } else if (n == 0){
            char* str1 = new char[10]();
            itoa(value, str1, 10);
            strcat(str2, "(");
            strcat(str2, str1);
            strcat(str2, "*");
            strncat(str2, str + 1, i -1);
            delete[] str1;
        } else {
            char* str1 = new char[10]();
            value *= -1;
            itoa(value, str1, 10);
            strcat(str2, "(");
            strcat(str2, str1);
            strcat(str2, "*");
            strncat(str2, str + 2, i - 1);
            delete[] str1;
        }
        m = i; n = 1; i++;
        while(str[i] != ')') {
            if ((m + 1) == i && str[i] == '-') {
                i++;
                if (!isdigit(str[i]))
                    n = 2;
                else if (!isdigit(str[i]))
                    n = 0;
            } else n = 0;
            i++;
        }
        if (n == 1) {
            strncat(str2, &str[m], 1);
            char *str1 = new char[20]();
            strncpy(str1, str + m + 1, strlen(str) - m - 2);
            num = atoi(str1);
            num *= value;
            itoa(num, str1, 10);
            strcat(str2, str1);
            strcat(str2, ")");
            delete[] str1;
        } else if (n == 0) {
            strncat(str2, &str[m], 1);
            char* str1 = new char[20]();
            itoa(value, str1, 10);
            strcat(str2, str1);
            strcat(str2, "*");
            strncat(str2, str + m + 1,strlen(str) - m - 2);
            strcat(str2, ")");
            delete[] str1;
        } else {
            char* str1 = new char[20]();
            value *= -1;
            itoa(value, str1, 10);
            strcat(str2, str1);
            strcat(str2, "*");
            strncat(str2, str + m + 2,strlen(str) - m - 2);
            delete[] str1;
        }
        delete[] str;
        strcpy(t, str2);
        delete[] str2;
    } else {//остальное
        if (shift->left_sen[0] == '-' && shift->right_sen[0] == '-' && shift-
>operation == '*') {//если выражение отрицательное, то минус вынести в начало
            std::strncat(t, shift->left_sen + 1, strlen(shift->left_sen) - 1);
```

```
            std::strncat(t, &shift->operation, 1);
            std::strncat(t, shift->right_sen + 1, strlen(shift->right_sen) - 1);
        } else if (shift->right_sen[0] == '-' && shift->left_sen[0] != shift-
>right_sen[0] && shift->operation == '*') {
            std::strcpy(t, "-");
            std::strcat(t, shift->left_sen);
            std::strncat(t, &shift->operation, 1);
            std::strncat(t, shift->right_sen + 1,strlen(shift->right_sen) - 1);
        } else {
            if (shift->operation != '*')
                std::strcpy(t, "(");
            std::strcat(t, shift->left_sen);
            std::strncat(t, &shift->operation, 1);
            std::strcat(t, shift->right_sen);
            if (shift->operation != '*')
                std::strcat(t, ")");
        }
    }
    if (!strcmp(t, ""))
        strcpy(t, "0");
    return t;
}
char* diff(char*& str, char x) {//дифференцирование простых выражений
    int flag = 0; //num = 0; count of x;
    char* str1 = new char[20]();
    char* str2 = new char[20]();
    for (int i = 0; i < strlen(str);i++) {
        if (str[i] == x)//подсчет переменных
            flag += 1;
    }
    if (flag > 0) {
        strncpy(str1, str, (strchr(str, x) - str));//удаление одной переменной и
замена на число по формуле
        itoa(flag, str2, 10);
        strcat(str1, str2);
        strcat(str1, strchr(str, x) + 1);
    } else {
        strcpy(str1, "0");//если переменных нет
    }
    delete[] str; str = new char[50]();
    strcpy(str, str1);
    delete[] str1; delete[] str2;
    return str;
}
void differentiateList(node* shift, char x) {//реализация дифференцирования
иерархического списка
    bool flag1 = false, flag2 = false;
    switch (shift->operation) { //в зависимости от знака между выражениями
        case '-':
        case '+':
            if (shift->left != nullptr) { std::cout << "in left differentiate-
List:+\n";//с помощью рекурсии дифференцируем выражение
                if (!strcmp(shift->left_sen, "-")) {
                    differentiateList(shift->left, x);
                    delete[] shift->left_sen;
                    shift->left_sen = new char[50]();
                    shift->left_sen = makeMinus(simplify(shift->left_sen, shift-
>left, x)); std::cout << shift->left_sen << "\n";
                } else {
                    differentiateList(shift->left, x);
                    shift->left_sen = simplify(shift->left_sen, shift->left, x);
std::cout << shift->left_sen << "\n";
                }
```

```
                delete shift->left; flag1 = true; std::cout << "out left differ-
entiateList:+\n";
            }
            if (shift->right != nullptr) { std::cout << "in right differentiate-
List:+\n";
                if (!strcmp(shift->right_sen, "-")) {
                    differentiateList(shift->right, x);
                    delete[] shift->right_sen;
                    shift->right_sen = new char[50]();
                    shift->right_sen = makeMinus(simplify(shift->right_sen,
shift->right, x)); std::cout << shift->right_sen << "\n";
                } else {
                    differentiateList(shift->right, x);
                    shift->right_sen = simplify(shift->right_sen, shift->right,
x); std::cout << shift->right_sen << "\n";
                }
                delete shift->right; flag2 = true; std::cout << "out right dif-
ferentiateList:+\n";
            }
            if (shift->left == nullptr && !flag1) {//конец списка,
непосредственное дифференцирование
                std::cout << "in left differentiateList:+\n"; shift->left_sen =
diff(shift->left_sen, x); std::cout << shift->left_sen << "\nout left differen-
tiateList:+ end\n";
            }
            if (shift->right == nullptr && !flag2) {
                std::cout << "in right differentiateList:+\n"; shift->right_sen
= diff(shift->right_sen, x); std::cout << shift->right_sen << "\nout right dif-
ferentiateList:+ end\n";
            }
            break;
        case '*'://по формуле сложение перемноженных выражений с
дифференцированными
            node *left_node = new node(); node *right_node = new node();//запись
производится здесь
            left_node->operation = '*'; right_node->operation = '*'; shift->op-
eration = '+';
            if (shift->left != nullptr) {
                if (!strcmp(shift->left_sen, "-"))
                    right_node->left_sen = makeMinus(simplify(right_node-
>left_sen, shift->left, x));
                else {
                    right_node->left_sen = simplify(right_node->left_sen, shift-
>left, x);
                }
            } else
                strcpy(right_node->left_sen, shift->left_sen);
            if (shift->right != nullptr) {
                if (!strcmp(shift->right_sen, "-"))
                    left_node->right_sen = makeMinus(simplify(left_node-
>right_sen, shift->right, x));
                else
                    left_node->right_sen = simplify(left_node->right_sen, shift-
>right, x);
            } else
                strcpy(left node->right sen, shift->right sen);
            if (shift->left != nullptr) { std::cout << "in left differentiate-
List:*\n";
                if (!strcmp(shift->left_sen, "-")) {
                    differentiateList(shift->left, x);
                    delete[] shift->left_sen;
                    shift->left_sen = new char[50]();
                    shift->left_sen = makeMinus(simplify(shift->left_sen, shift-
>left, x)); std::cout << shift->left_sen << "\n";
```

```
                } else {
                    differentiateList(shift->left, x);
                    shift->left_sen = simplify(shift->left_sen, shift->left, x);
std::cout << shift->left_sen << "\n";
                }
                delete shift->left; flag1 = true; std::cout << "out left differ-
entiateList:*\n";
            }
            if (shift->right != nullptr) { std::cout << "in right differentiate-
List:*\n";
                if (!strcmp(shift->right_sen, "-")) {
                    differentiateList(shift->right, x);
                    delete[] shift->right_sen;
                    shift->right_sen = new char[50]();
                    shift->right_sen = makeMinus(simplify(shift->right_sen,
shift->right, x)); std::cout << shift->right_sen << "\n";
                } else {
                    differentiateList(shift->right, x);
                    shift->right_sen = simplify(shift->right_sen, shift->right,
x); std::cout << shift->right_sen << "\n";
                }
                delete shift->right;  flag2 = true; std::cout << "out right dif-
ferentiateList:*\n";
            }
            if (shift->left == nullptr && !flag1) {
                std::cout << "in left differentiateList:*\n"; shift->left_sen =
diff(shift->left_sen, x); std::cout << shift->left_sen << "\nout left differen-
tiateList:* end\n";
            }
            if (shift->right == nullptr && !flag2) {
                std::cout << "in right differentiateList:*\n"; shift->right_sen
= diff(shift->right_sen, x); std::cout << shift->right_sen << "\nout right dif-
ferentiateList:* end\n";
            }
            strcpy(left_node->left_sen, shift->left_sen);
            strcpy(right_node->right_sen, shift->right_sen);
            delete[] shift->left_sen;
            delete[] shift->right_sen;
            shift->left_sen = new char[50]();
            shift->right_sen = new char[50]();
            shift->left_sen = simplify(shift->left_sen, left_node, x);//запись
упрощенного выражения обратно в наш список
            shift->right_sen = simplify(shift->right_sen, right_node, x);
            delete left_node;
            delete right_node;
            break;
    }
}
#endif //UNTITLED3_HEADER_H
```