

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт математики и фундаментальной информатики СФУ
Кафедра математического анализа и дифференциальных уравнений

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Фроленков И.В.
подпись инициалы, фамилия
« _____ » _____ 20 ____ г.

**ОТЧЕТ О ПРАКТИКЕ
ПО ПОЛУЧЕНИЮ ПРОФЕССИОНАЛЬНЫХ УМЕНИЙ И
ОПЫТА ПРОФЕССИОНАЛЬНОЙ ДЕЯТЕЛЬНОСТИ**

Место прохождения практики:
Институт математики и фундаментальной информатики

Тема практики: **Написать программу, упрощающую элементарные выражения.**

Руководитель	_____	_____	_____
	подпись, дата	должность, ученая степень	инициалы, фамилия
Студент	_____	_____	_____
	номер группы	номер зачетной книжки	подпись, дата инициалы, фамилия

Красноярск 2022

Содержание

1. Постановка задачи	3
2. Основные определения	4
3. Описание программы	4
3.1. Среда разработки программы	5
3.2. Алгоритм решения задачи	5
3.3. Описание основных функций программы	8
3. Примеры результатов работы программы	12
Список использованных источников	14
Приложение	15

1. Постановка задачи

Целью работы является создание программы на языке программирования C++, упрощающую элементарные выражения. Программа на вход получает элементарное выражение, а на выходе выводит упрощенное выражение. Под элементарным выражением подразумевается выражения, состоящие из многочленов с произвольными операциями, кроме деления. Под упрощенным выражением понимается, что над многочленами проделаны все заданные операции.

2. Основные определения

Инфиксная запись — это форма записи математических и логических формул, в которой операторы записаны в инфиксном стиле между операндами, на которые они воздействуют (например, $2 + 2$). Задача разбора выражений, записанных в такой форме, для компьютера сложнее по сравнению с постфиксной ($2\ 2\ +$). Эта запись используется в большинстве языков программирования как более естественная для человека.

Обратная польская запись (постфиксная запись) – форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. В общем виде запись выглядит следующим образом:

1. Запись набора операций состоит из последовательности операндов и знаков операций. Операнды в выражении при письменной записи разделяются пробелами.
2. Выражение читается слева направо. Когда в выражении встречается знак операции, выполняется соответствующая операция над двумя последними встретившимися перед ним операндами в порядке их записи. Результат операции заменяет в выражении последовательность её операндов и её знак, после чего выражение вычисляется дальше по тому же правилу.
3. Результатом вычисления выражения становится результат последней вычисленной операции.

В инфиксной записи, в отличие от постфиксной, скобки, окружающие группы операндов и операторов, определяют порядок, в котором будут выполнены операции. При отсутствии скобок операции выполняются согласно правилам приоритета операторов.

Подробные примеры преобразований из постфиксной записи в инфиксную и обратно рассмотрены в пункте **3.2.1**, таблицах 1, 2, 3.

3. Описание программы

3.1 Среда разработки программы

Программа реализована в среде разработки Microsoft Visual Studio 2019 на языке C++ с применением библиотек: «Windows.h» - библиотека содержит заявление для всех функций в Windows API, все общие макросы, все типы данных, используемых различными функциями и подсистемами; «string» - класс с методами и переменными для организации работы со строками в языке программирования C++; «stack», «vector», «map» - готовые шаблоны контейнеров stack, vector, multimap; «algorithm» - библиотека включающая в себя набор функций, для выполнения алгоритмических операций над контейнерами.

3.2 Алгоритм решения задачи

3.2.1 Описание алгоритма решения задачи

При запуске программы появляется окно консольное окно. В данном окне пользователь вводит выражение. После этого начинает работать алгоритм программы. Приведем последовательность действий алгоритма работы программы:

1. Алгоритм передаёт входную строку (выражение) в функцию «PostfixNotation», которое из инфиксной записи выражения, строит постфиксную, по следующему принципу:

Входная строка символов просматривается слева направо, операнды переписываются в выходную строку, знаки операций заносятся в стек.

Если стек пуст, операция из входной строки переписывается в стек. Текущая операция выталкивает из стека все операции с большим или равным приоритетом (определение приоритета - функция «Priority») в выходную строку. Открывающая скобка записывается в стек, закрывающая скобка выталкивает в выходную строку все операции из стека до ближайшей открывающей скобки, сами скобки в выходную строку не попадают,

уничтожая друг друга. Если символы в строке закончились, а стек не пуст, то выталкиваются все оставшиеся операторы.

Символ	1	2	3	4	5	6	7	8	9	10	11	
Вход	(A	+	B)	*	(C	+	D)	
Состояние стека	((+	+		*	((+	+	*	
			((*	*	((
									*	*		
Выход		A		B	+			C		D	+	*

Таблица 1 – пример построения постфиксной записи выражения $(A+B)*(C+D)$

- Постфиксная выражение передается в функцию «InfixView», которая преобразует выражение в инфиксную (стандартную) форму, перемножая все элементы (если это требуется в выражении).

Входная строка просматривается слева направо, операнды добавляются в стек. Если следующий символ является оператором, на последние два элемента из стека применяется операция, после чего полученное выражение записывается в стек. Если операция является умножением, то последние два элемента стека передаются в функцию «Multiplication», которая возвращает поэлементно перемноженные операнды (подробнее о работе функции в пункте 2.2.3). Если оператор вычитания, то он изменяется на строку: «+(-1)*» (функция «Sign»). Верхний элемент стека записывается в строку, являющейся инфиксной записью выражения, на которой выполнены перемножения (если это требуется).

Символ	1	2	3	4	5	6	7
Вход	A	B	+	C	D	+	*

Состояние стека	A	B	A+B	A+B C	A+B C D	A+B C+D	C*B+D*B+C*A+D*A
--------------------	---	---	-----	----------	---------------	------------	-----------------

Таблица 2 – пример построения иксфиксной записи выражения $AB+CD^*$

Символ	1	2	3	4	5	6	7	8	9
Вход	3	2	x	*	4	y	*	-	*
Состояние стека	3	3 2	3 2 x	3 2*x	3 2*x 4	3 2*x 4 y	3 2*x 4*y	3 2*x+(-1)*4*y	x*2*3+(-1)*y*4*3

Таблица 3 – пример построения иксфиксной записи выражения $3*(2*x-4*y)$ из постфиксной $3\ 2\ x\ *\ 4\ y\ *\ -\ *$

- Полученная строка передается в функцию «CalcView», которая вычисляет коэффициенты при переменных, сортирует их с помощью встроенной функции «sort» из библиотеки «algorithm», определяет их знак (функция «DefinitonSign») и записывает в вид (функция «CanonView»), удобный для «сложения» коэффициентов при одинаковых переменных.

Вход: $x*3*2+(-1)*6*2+5$

Выход: $6*x;-12*;5*$

- Полученная строка передается в функцию «Operation», которая «складывает» коэффициенты при одинаковых переменных, путем использования словаря multimap. Где ключами являются переменные, а значениями являются коэффициенты при переменных. Если ключ встречается второй раз, то коэффициенты складываются. Затем идет цикл по всем ключам словаря, который записывает их в выходную строку.
- Полученная строка отправляется в функцию «Printing», которая чистит вывод и приводит его к более правильному виду.

3.2.2 Описание основных функций программы

Функция определения приоритета символа «Priority», в неё передается символ и в соответствии с таблицей 4, возвращается приоритет (число).

Операция	Приоритет
(0
)	1
+, -	2
*	3
<i>сторонний символ</i>	4

Таблица 4 – приоритеты символов

Функция определения является ли символ числом «BoolInt», в неё передается символ и возвращается 1, если символ является числом и 0 в противоположном случае.

Функция построения постфиксной формы «PostfixNotation», в неё передается строка. Подробнее о работе функции написано в части 2.2.2, пункте 1.

Функция построения инфиксной формы «InfixView», в неё передается строка. Подробнее о работе функции написано в части 2.2.2, пункте 2.

Функция изменения знака вычитания, «Sign», она вызывается из функции «InfixView», когда встречается оператор вычитания. На вход подается операнд, перед которым стоит минус он заменяется на строку «+(-1)*». Выходная строка представляет собой строку «+(-1)*» объединенную с операндом, причем в операнде все плюсы заменены на минус и наоборот.

Входная строка	Выходная строка
x+y	+(-1)*x-y

Таблица 5 – пример работы функции «Sign»

Функция перемножения операндов «Multiplication», вызывается из функции «InfixView», когда встречается оператор умножения. На вход подаются два операнда (далее А и В), которые необходимо перемножить. А и В анализируются и сравниваются по количеству в них знаков различных операций, если какой-то из них больше, они меняются местами. Затем по длине самого короткого по

знакам операнда запускается цикл (в нашем случае это будет В), который записывает в стек1 все переменные и числа в В, а в стек2 все операции в В. По итогу у нас получается стек1, заполненный всеми операндами В и стек2, заполненный всеми при этих операндах. Затем запускается цикл, пока стек1 не окажется пустым. По строке А, начиная с i-того элемента идет проверка символов. Если у символа приоритет 4 (см. таблицу 4) (причем символ не равен единице), а за ним следует символ с приоритетом 2, то к выходной строке добавляется знак умножения, верхний элемент стека1 и A[i] элемент строки (операнд операнда А умножается на операнд операнда В). В случае, если это условие не выполняется и индекс i равен длине строки А, то в выходную строку записывается знак умножения и верхний элемент стека1, после этого к выходной строке добавляется оператор из стека2, верхние элементы стека1 и стека2 удаляются и индекс i обнуляется. Таким образом мы «умножили» строку А на верхний элемент стека1. В противном случае, если все условия не выполняются, то к выходной строке добавляем A[i] элемент. По итогу всей функции, мы получаем «перемноженные» строки А и В. Например, пусть $A := x+y$, а $B := x+y+z$, тогда короткая строка А запишется в виде стеков, а длинная строка В, будет умножаться (см. Таблицу 7)

Шаг	1	2	3	4	...	6
Строка В	$x+y+z$	$x+y+z$	$x+y+z$	$x*x+y*x+z*x$...	$x*y*x+y*y*x+z*y*x$
Стек1	y x	y x	y x	x	x	
Стек2	+	+	+			
Выход	$x*y+y+z$	$x*y+y*y+z$	$x*y+y*y+z*y$	$x*y*x+y*x+z*x$...	$x*y*x+y*y*x+z*y*x$

Таблица 7 – пример работы функции «Multiplication»

Функция «CanonView» записывает входную строку в более удобный для анализа и последующего вычисления коэффициентов вид. В неё передается строка, которая анализируется слева направо, если встречается символ с приоритетом 4, либо комбинация $(-1)^*$, то она записывается в строку digits. Если встречается символ с приоритетом 2 (причем не упомянутая выше комбинация),

то в выходную строку записывается строка digits, символ «;» и символ с приоритетом 2, строка digits обнуляется и процедура повторяется до тех пор, пока не проанализируется вся строка.

Входная строка	Выходная строка
$x*(-1)*y+(-1)*y*(-1)*y+x*x+(-1)*y*x$	$+x*-1*y;+1*y*-1*y;+x*x;+1*y*x;$

Таблица 5 – пример работы функции «CanonView»

Функция определения знакам при переменной «DefinitonSign», в неё передается строка, полученная из функции «CanonView». Анализируется строка слева направо, если встретился символ «-», а за ним следует 1, то к счетчику counter прибавляется единица. Если в строке мы встретили «;» и counter не кратен двойки, то перед переменной ставится знак «-», в ином случае «+», счетчик counter обнуляется, и процедура повторяется, пока не наступит конец строки.

Входная строка	Выходная строка
$+x*-1*y;+1*y*-1*y;+x*x;+1*y*x;$	$-x*1*y;+1*y*1*y;+x*x;-1*y*x;$

Таблица 8 – пример работы функции «DefinitionSign»

Функция вычисления коэффициентов при переменных «CalcView», на вход подается строка из функции «DefinitionSign». Слева направо анализируется строка аналогично функции «CanonView», с рядом изменений. Если символ является оператором сложения/вычитания, то он записывается в стек, если символ является переменной, то он записывается в контейнер типа vector, если символ является числом (проверяется с помощью функции «BoolInt»), то он записывается в строку digits, также в строку digits записывается оператор, стоящий при этих числах (так как деление не было реализовано, то это оператор умножения). Если встречается символ «;», то строка digits отправляется в функцию «CalcCoef», которая вычисляет числа в переменной digits (подробнее о функции ниже). По итогу этого круга, в выходную строку записывается вычисленное число, знак умножения и, отсортированный с помощью встроенной функции sort, vector. И так до тех пор, пока строка не будет проанализирована полностью.

Входная строка	Выходная строка
$+x*4*3*4*3;-1*x*9*8$	$144*x;-71*x;$

Таблица 9 – пример работы функции «CalcView»

Функция вычисления чисел в строке «CalcCoef», вызывается из функции «CalcView», на вход подается строка, содержащая числа и знаки умножения. Если строка является числом, то она возвращается, в противном случае по этой строке строится постфиксная запись (функция «PostfixNotation») и вызывается функция «PostfixCalc» (подробнее о функции ниже), которая по постфиксной записи вычисляет выражение. Полученный результат конвертируется в выходную строку.

Функция вычисления чисел по постфиксной записи «PostfixCalc». На вход поступает строка, которая анализируется слева направо. Работа функции аналогична функции «InfixView», за исключением того, что функция не записывает в выходную строку символы, а вычисляет их. Если встречается оператор, то вызывается функция «Operator», в которую подаются два верхних операнда из стека и вычисленное выражение вновь записывается в стек. Так до тех пор, пока строка не будет обработана. Возвращается верхний элемент стека.

Функция сложения коэффициентов при переменных «Operation» (не путать с «Operator»), на вход получает строку из функции «CalcView». Создается контейнер multimap (далее словарь), ключами которого являются переменные, а их значения являются числа. В силу особенности языка C++, при записи одинаковых ключей с разными значениями, он хранит их в разных ячейках памяти, но в быстром доступе (эти ключи идут подряд друг за другом). Затем, идет цикл по ключам словаря и складываются одинаковые коэффициенты при ключах, т.е. при одинаковых переменных и сразу же выводятся в выходную строку. В конце концов, мы получаем упрощенное выражение.

Входная строка	Выходная строка
$-1*xy; 1*yu; 1*xx; -1*xy$	$1*xx - 2*xy + 1*yu$

Таблица 10 – пример работы функции «Operation»

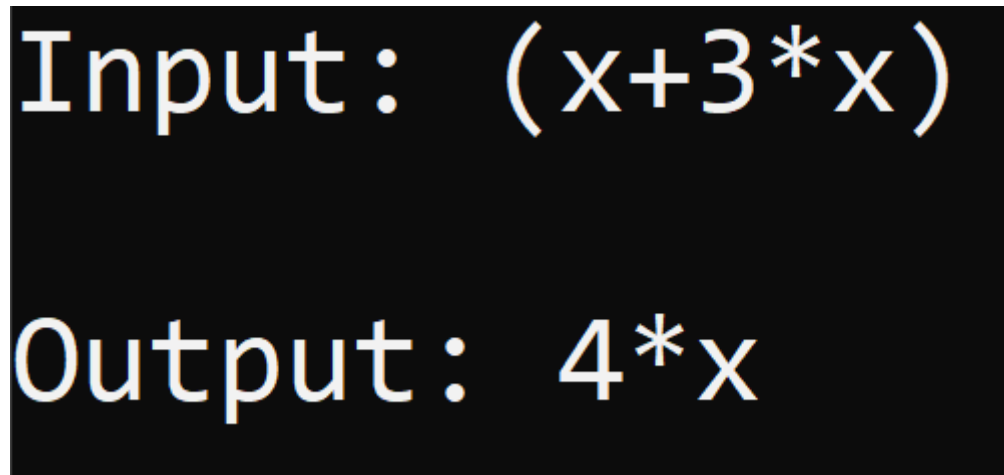
Функция вывода «Printing», чистит вывод, убирая лишние символы.

Входная строка	Выходная строка
$1*xx - 2*xy + 1*yu$	$xx - 2*xy + yu$

Таблица 11 – пример работы функции «Printing»

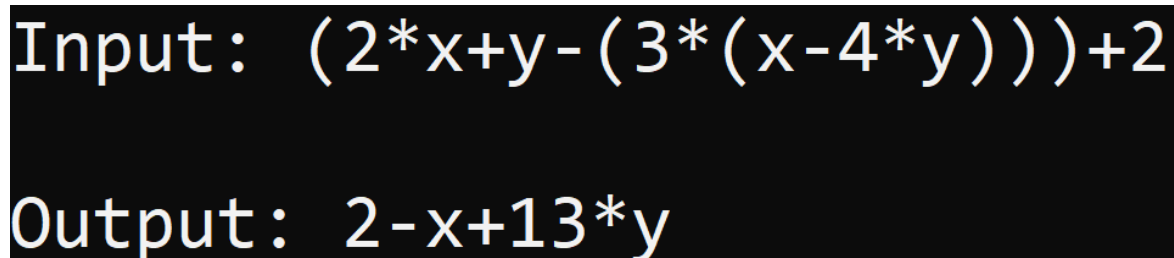
4. Примеры результатов работы программы

Приведем тестовые примеры работы программы для некоторых выражений.



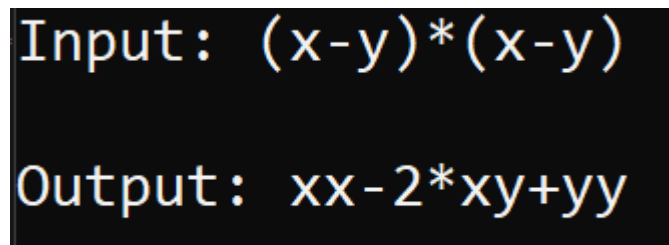
Input: $(x+3*x)$
Output: $4*x$

Рисунок 1 – пример работы программы



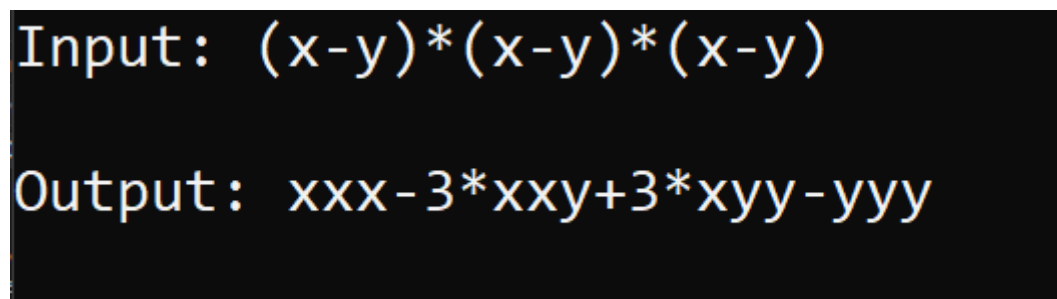
Input: $(2*x+y-(3*(x-4*y)))+2$
Output: $2-x+13*y$

Рисунок 2 – пример работы программы



Input: $(x-y)*(x-y)$
Output: $xx-2*xy+yy$

Рисунок 3 – пример работы программы



Input: $(x-y)*(x-y)*(x-y)$
Output: $xxx-3*xxxy+3*xуу-ууу$

Рисунок 4 – пример работы программы

Input: $(x-y)*(x-y)*(x+y)$

Output: $xxx-xyy-xyy+yyy$

Рисунок 5 – пример работы программы

Input: $(x-y-z)*(x-y-z)$

Output: $xx-2*xy-2*xz+yy+2*yz+zz$

Рисунок 6 – пример работы программы

Input: $12*3*x+12*x-3*2*x*y*(x-y)*(x-y)+15$

Output: $15+48*x-6*xxxxy+12*xxxy-6*xyyy$

Рисунок 7 – пример работы программы

Список использованных источников

Портал коллективных блогов Habr [Электронный ресурс] – Польская нотация или как легко распарсить алгебраическое выражение – Режим доступа:

<https://habr.com/ru/post/596925/>

Приложение

```
#include <iostream>
#include <windows.h>
#include <string>
#include <stack>
#include <vector>
#include <algorithm>
#include <map>

using namespace std;

int Priority(char c) {
    if (c == '(')
        return 0;
    if (c == ')')
        return 1;
    if (c == '+' || c == '-')
        return 2;
    if (c == '*' || c == '/')
        return 3;
    return 4;
}

int BoolInt(char c) {
    if (isdigit(c)) return 1;
    else return 0;
}

string PostfixNotation(string a) {
    string vvod, vvod;
    stack <char> steck;
    if (a[0] == '-') {
        a[0] = '*';
        vvod += "(-1)" + a;
    }
    else vvod = a;
    for (int i = 0; i <= vvod.length() - 1; i++) {
        if (Priority(vvod[i]) == 4) {
            vvod.push_back(vvod[i]);
            if (Priority(vvod[i]) == Priority(vvod[i + 1]) && i != vvod.length() - 1) {
                i++;
                while (Priority(vvod[i]) == 4) {
                    if (i == vvod.length()) break;

```

```

        vuvod.push_back(vvod[i]);
        i++;
    }
}
vuvod.push_back(' ');
if (i == vvod.length() - 1) {
    while (size(steck) != 0) {
        if (steck.top() == '(') {
            steck.pop();
            continue;
        }
        vuvod.push_back(steck.top());
        vuvod.push_back(' ');
        steck.pop();
    }
}
}
if (Priority(vvod[i]) == 2 || Priority(vvod[i]) == 3) {
    if (size(steck) == 0) {
        steck.push(vvod[i]);
        continue;
    }
    while (Priority(vvod[i]) <= Priority(steck.top()) && steck.top() != '(') {
        vuvod.push_back(steck.top());
        vuvod.push_back(' ');
        steck.pop();
        if (size(steck) == 0) break;
    }
    steck.push(vvod[i]);
}
if (Priority(vvod[i]) == 1 && vvod[i-3] != '(') {
    while (size(steck) != 0 && steck.top() != '(') {
        vuvod.push_back(steck.top());
        vuvod.push_back(' ');
        steck.pop();
    }
    if (size(steck) != 0) steck.pop();
}
if (Priority(vvod[i]) == 0 && vvod[i+1] != '-') {
    steck.push(vvod[i]);
}
if (Priority(vvod[i]) == 0 && vvod[i + 1] == '-') {
    vuvod += "(-1) ";
    i += 3;
}
}

```



```

    if (i == vvod.length()) {
        while (size(steck) != 0) {
            vuvod.push_back(steck.top());
            vuvod.push_back(' ');
            steck.pop();
        }
    }
}
while (steck.size() != 0) {
    if (steck.top() == '(') {
        steck.pop();
        continue;
    }
    vuvod.push_back(steck.top());
    vuvod.push_back(' ');
    steck.pop();
}

return vuvod;
}

```

```

float Operators(float a, float b, char op) {
    if (op == '-') return a - b;
    if (op == '+') return a + b;
    if (op == '*') return a * b;
    if (op == '/') return a / b;
}

```

```

int PostfixCalc(string str) {
    stack <float> steck;
    string digits;
    for (int i = 0; i != str.length(); i++) {
        if (str[i] == ' ') continue;
        if (Priority(str[i]) == 4) {
            digits.push_back(str[i]);
            if (Priority(str[i + 1]) == 4 && i != str.length() - 1) {
                i++;
                while (str[i] != ' ') {
                    digits.push_back(str[i]);
                    i++;
                }
            }
            steck.push(stof(digits));
            digits.clear();
            continue;
        }
    }
}

```

```

    }
    if (Priority(str[i]) == 3 || Priority(str[i]) == 2) {
        float a = steck.top();
        steck.pop();
        float b = steck.top();
        steck.pop();
        steck.push(Operators(b, a, str[i]));
    }
}
return steck.top();
}

```

```

string Multiplication(string a, string b, char op) {
    stack <string> steck;
    stack <char> oper;
    string digits;
    int aznak = 0;
    int bznak = 0;
    int j = 0;
    for (int i = 0; i < a.length(); i++)
        if (Priority(a[i]) == 2) aznak++;
    for (int i = 0; i < b.length(); i++)
        if (Priority(b[i]) == 2) bznak++;
    if (aznak <= bznak) swap(a, b);
    for (int i = 0; i < b.length(); i++) {
        if (Priority(b[i]) == 4 || Priority(b[i]) == 3 || Priority(b[i]) == 0) {
            if (Priority(b[i]) == 0)
                while (Priority(b[i]) != 1) {
                    digits.push_back(b[i]);
                    i++;
                }
            digits.push_back(b[i]);
        }
        else {
            oper.push(b[i]);
            steck.push(digits);
            digits.clear();
        }
    }
    if (digits.length() != 0) steck.push(digits);

    string result;
    int i = 0;
    while (steck.size() != 0) {

```

```

    if (Priority(a[i]) == 2 && a[i+1] != '1') {
        result += op + steck.top() + a[i];
    }
    else if (i == a.length()) {
        if (aznak == 0)
            return result += op + steck.top();
        result += op + steck.top();
        if (oper.size() != 0) {
            result += oper.top();
            oper.pop();
        }
        steck.pop();
        i = 0;
        continue;
    }
    else {
        result += a[i];
    }
    i++;
}
return result;
}

string Sign(string a) {
    string b;
    b += "+(-1)*";
    for (int i = 0; i < a.length(); i++) {
        if (a[i] == '+') a[i] = '-';
        else if (a[i] == '-' && a[i + 1] != '1') a[i] = '+';
    }
    return b+a;
}

string InfixView(string str) {
    stack<string> steck;
    string digits;
    string a, b, c, op;
    int j = 0;
    for (int i = 0; i <= str.length(); i++) {
        if (str[i] == ' ') continue;
        if (Priority(str[i]) == 4 && i <= str.length() - 1) {
            while (str[i] != ')') {
                digits.push_back(str[i]);
                i++;
            }
            steck.push(digits);

```

```

        digits.clear();
        continue;
    }
    if (Priority(str[i]) == 0 && i <= str.length() - 1) {
        while (str[i] != ')') {
            digits.push_back(str[i]);
            i++;
        }
        steck.push(digits);
        digits.clear();
        continue;
    }
    if (Priority(str[i]) == 2 && str[i-1] != '(') {
        j = i;
        op = str[j];
        a = steck.top();
        steck.pop();
        b = steck.top();
        steck.pop();
        if (op == "-") {
            a = Sign(a);
            steck.push(b + a);
        }
        else steck.push(b + op + a);
    }
    else {
        if (Priority(str[i]) == 3 && i != j) {
            a = steck.top();
            steck.pop();
            b = steck.top();
            steck.pop();
            steck.push(Multiplication(b, a, str[i]));
        }
    }
}

return steck.top();
}

string CalcCoef(string a) {
    if (Priority(a[0]) == 3) a.erase(0, 1);
    if (a.find("*") != string::npos)
        a = to_string(PostfixCalc(PostfixNotation(a)));
    return a;
}

```

```

string CalcView(string elem) {
    vector<char> vect;
    stack<char> op;
    string digits, result;
    for (int i = 0; i < elem.length(); i++) {
        if (elem[i] == ';') {
            digits = CalcCoef(digits);
            if (op.top() == '-') result.push_back(op.top());
            op.pop();
            if (digits.size() != 0) {
                result += digits;
                result.push_back('*');
            }
            else {
                result.push_back('1');
                result.push_back('*');
            }
            sort(vect.begin(), vect.end());
            for (int j = 0; j < vect.size(); j++) {
                result.push_back(vect[j]);
                if (j == vect.size() - 1) break;
            }
            result += ";";
            digits.clear();
            vect.clear();
            continue;
        }
        if (Priority(elem[i]) == 2) {
            op.push(elem[i]);
            continue;
        }

        if (BoolInt(elem[i]) == 0 && Priority(elem[i]) != 3) {
            vect.push_back(elem[i]);
            continue;
        }
        if (BoolInt(elem[i + 1]) == 1 && Priority(elem[i]) == 3) {
            digits.push_back(elem[i]);
            continue;
        }
        if (BoolInt(elem[i]) == 1) digits.push_back(elem[i]);
    }
    return result;
}

```

```

string DefinitionSign(string elem) {
    string a, digits;
    int counter = 0;
    for (int i = 0; i < elem.length(); i++) {
        if (elem[i] == ';') {
            if (counter % 2) a.push_back('-');
            else a.push_back('+');
            a += digits;
            a.push_back(';');
            digits.clear();
            counter = 0;
            continue;
        }
        if (elem[i] == '-' && elem[i + 1] == '1') {
            counter++;
        }
        else if (elem[i] != '+') digits.push_back(elem[i]);
    }
    return a;
}

```

```

string CanonView(string a) {
    string elem;
    string digits;
    for (int i = 0; i < a.length(); i++) {
        if (i == 0 && Priority(a[i]) == 4) {
            digits.push_back(a[i]);
            elem.push_back('+');
            continue;
        }
        if (i == 0 && Priority(a[i]) == 2) {
            elem.push_back(a[i]);
            elem += ",";
            continue;
        }
        if (a[i] == '(') {
            i++;
            while (a[i] != ')') {
                digits.push_back(a[i]);
                i++;
            }
        }
        if (Priority(a[i]) == 4 || Priority(a[i]) == 3) {
            digits.push_back(a[i]);

```

```

    }
    if (Priority(a[i]) == 2) {
        elem += digits;
        elem += ",";
        if (a[i] == '-') elem += "-1*";
        else elem += a[i];
        digits.clear();
    }
    if (i == a.length() - 1) {
        elem += digits + ",";
        digits.clear();
        break;
    }
}

return CalcView(DefinitionSign(elem));
}

string Operation(string a) {
    string digits;
    string b,c;
    multimap<string, int> m;
    for (int i = 0; i < a.length(); i++) {
        if (a[i] == ';') {
            for (int j = 0; j < digits.length(); j++) {
                if (digits[j] == '-') { b.push_back(digits[j]); continue; }
                if (BoolInt(digits[j]) == 1 && digits[j] != '*') {
                    b.push_back(digits[j]);
                }
                else if (digits[j] != '*') {
                    c.push_back(digits[j]);
                }
            }
            m.insert({ c, stoi(b) });
            digits.clear();
            b.clear();
            c.clear();
            continue;
        }
        digits.push_back(a[i]);
    }
    digits.clear();
    multimap<string, int>::iterator it1, it2, it3;
    it1 = m.begin();
    it2 = it1;

```

```

it2++;
it3 = m.end();
string result;
int k = 0;
while (it1 != it3) {
    if (it2 != m.end()){
        if (it1->first == it2->first) {
            while (it1->first == it2->first) {
                k += it1->second;
                it1++;
                if (it1 == m.end()) break;
            }
            result += to_string(k) + "*" + it2->first + "+";
            if (it1 == m.end()) break;
            it2 = it1;
            it2++;
            k = 0;
            continue;
        }
    }
    result += to_string(it1->second) + "*" + it1->first + "+";

    it1++;
    if (it1 == m.end()) break;
    it2++;
}
result.erase(result.length() - 1, 1);
return result;
}

void Printing(string a) {
    string b;
    for (int i = 0; i < a.length(); i++) {
        if (a[i] == '*' && Priority(a[i + 1]) == 2) a[i] = ' ';
        if (a[i] == '*' && i == a.length() - 1) a[i] = ' ';
        if (Priority(a[i]) == 2 && Priority(a[i + 1]) == 2) a[i] = ' ';
        if (a[i] == '1' && a[i + 1] == '*') {
            if (Priority(a[i + 2]) == 2) continue;
            a[i] = ' ';
            a[i + 1] = ' ';
            i++;
        }
    }
    for (int i = 0; i < a.length(); i++) {
        if (a[i] == ' ') continue;

```



```

        b.push_back(a[i]);
    }
    cout << b;
}

int main(){
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    string a;
    cout << "Input: ";
    cin >> a;

    cout << endl << "Output: ";
    a = Operation(CanonView(InfixView(PostfixNotation(a))));
    Printing(a);
    cout << endl << endl << endl;
}

```