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

ОТЧЕТ

по практической работе №3
по дисциплине «Программирование»
Тема: Польские нотации. Стек и очередь

Студент гр. _____ Сотина Е.А.

Преподаватель _____ Глущенко А.Г.

Санкт-Петербург
2020

Цель работы.

Получение практических навыков работы со стеками и очередями; изучение обратной и прямой польской нотации; проведение сравнительного анализа этих структур данных.

Основные теоретические положения.

Стек – это частный случай однонаправленного списка, добавление элементов в который и выборка из которого выполняется с одного конца, называемого вершиной стека. Другие операции со стеком не определены. При выборке элемент исключается из стека. Говорят, что стек реализует принцип обслуживания LIFO (последним пришел – первым ушел).

Основные операции над стеками:

1) чтение верхнего элемента;
2) добавление нового элемента;
3) удаление существующего элемента.

Графически его удобно изобразить в виде вертикального списка (рис. 1), например, стопки книг, где для того чтобы воспользоваться одной из них и не нарушить установленный порядок, нужно поднять все книги, которые лежат выше нее, а положить книгу можно лишь поверх всех остальных.

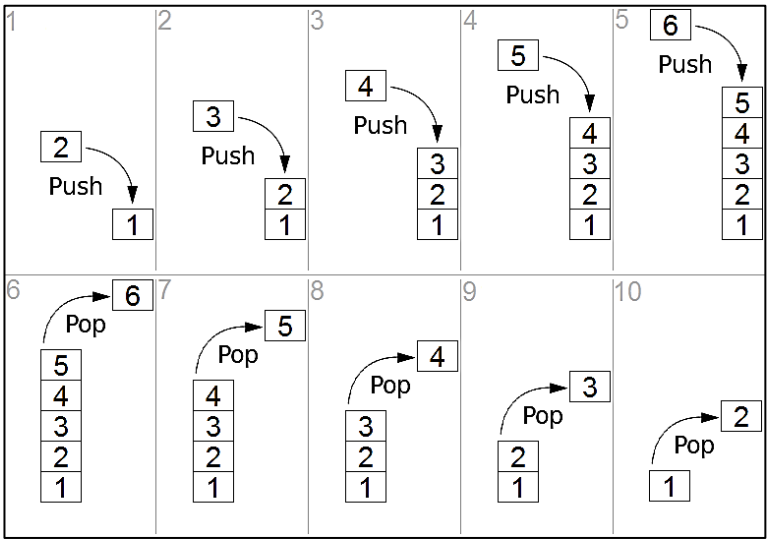


Рис. 1. Графическое представление стека

Стек чаще всего реализуется на основе обычных массивов, односвязных и двусвязных списков. В зависимости от конкретных условий выбирается одна из этих структур данных.

Очередь – частный случай однонаправленного списка, добавление элементов в который выполняется в один конец, а выборка – из другого конца. Другие операции с очередью не определены. При выборке элемент исключается из очереди. Говорят, что очередь реализует принцип обслуживания FIFO (первым пришел – первым ушел). В программировании очереди применяются при моделировании, диспетчеризации задач операционной системой, буферизованном вводе/выводе.

Графически ее удобно изобразить в виде вертикального списка (рис. 2), например, очередь в магазине, где для того чтобы дойти до кассы и не нарушить установленный порядок, нужно дождаться, пока все покупатели перед вами не приобретут товар. Разумеется, будут появляться новые покупатели, которые будут занимать свое место в очереди в ожидании покупки.

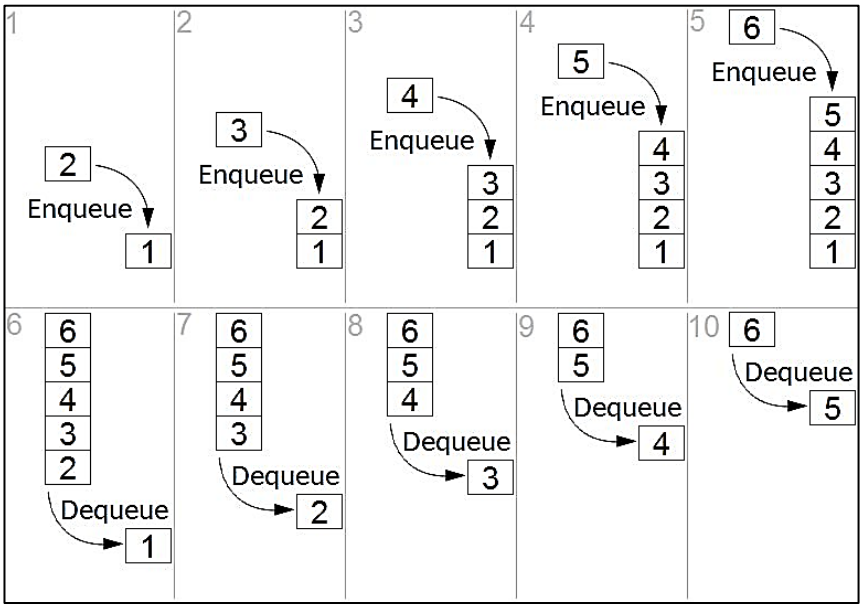


Рис. 2. Графическое представление очереди

Основные операции над очередями:

1) чтение первого элемента;
2) добавление нового элемента;
3) удаление существующего элемента.

Если для стека в момент добавления или удаления элемента допустимо задействование лишь его вершины, то касательно очереди эти две операции должны быть применены так, как это регламентировано в определении этой структуры данных, т. е. добавление – в конец, удаление – из начала.

Выделяют два способа программной реализации очереди. Первый основан на базе массива, второй – на базе указателей (связного списка). Первый способ

– статический, так как очередь представляется в виде простого статического массива, второй – динамический.

Кольцевой буфер так же известен, как кольцевая очередь или циклический буфер и является распространенной формой очереди. Это популярный, легко реализуемый стандарт, и хотя он представлен в виде круга (рис. 3) в базовом коде он является линейным. Кольцевая очередь существует как массив фиксированной длины с двумя указателями: один представляет начало очереди, другой – хвост.

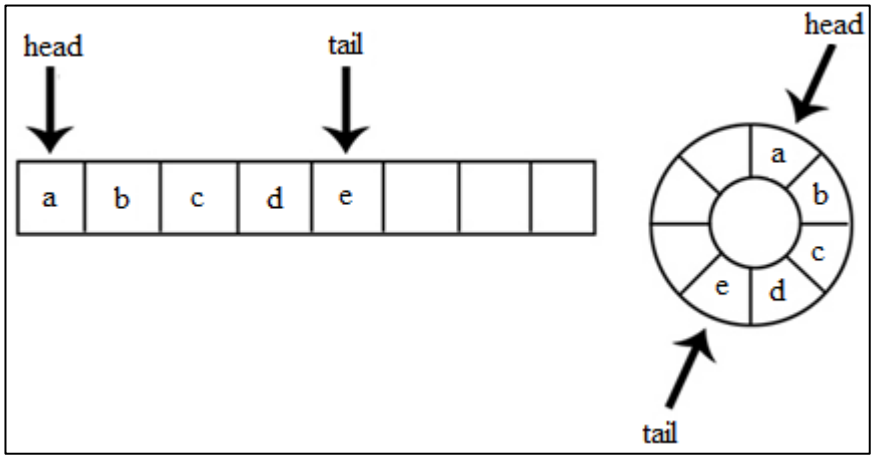


Рис. 3. Кольцевая очередь

Операции с очередями работают следующим образом.

1. Два указателя, называемые HEAD и TAIL, используются для отслеживания первого и последнего элементов в очереди.

2. При инициализации очереди значения HEAD и TAIL устанавливаются равными –1.

3. При добавлении элемента постепенно увеличивается значение индекса TAIL и помещается новый элемент в положение, на которое указывает TAIL.

4. При снятии очереди с элемента возвращается значение, на которое указывает HEAD, и постепенно увеличивается индекс HEAD.

5. Перед постановкой в очередь проверяется, заполнена ли очередь.

6. Перед снятием очереди проверяется, пуста ли очередь.

7. При инициализации первого элемента устанавливается значение HEAD в 0.

8. При удалении последнего элемента сбрасываются значения HEAD и TAIL в –1.

Недостатком метода является его фиксированный размер. Для очередей, где элементы должны быть добавлены и удалены в середине, а не только в начале и конце буфера, реализация в виде связанного списка является предпочтительным подходом.

Обратная польская запись (нотация) (рис. 4) – форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Обратная польская запись имеет ряд преимуществ перед инфиксной записью при выражении алгебраических формул, одно из них то, что инфиксные операторы имеют приоритеты, которые произвольны и нежелательны.

Простое выражение	Прямая польская запись	Обратная польская запись
X + 3 * Y	+ X * 3 Y	X 3 Y * +
(X + 3) * Y	* + X 3 Y	X 3 + Y *
1 + 2	+ 1 2	1 2 +

Рис. 4. Представление прямой и обратной польской записи

Обратная польская запись отлично подходит для вычисления выражений при помощи стека. Причем сам алгоритм достаточно прост. Необходимо просто прочитать обратную польскую запись слева направо. Если встречается операнд, то его нужно поместить в стек. Если встречается оператор, нужно выполнить заданную им операцию.

Постановка задачи.

Необходимо написать программу, которая выполняет следующее:

1. Реализует преобразование введенного выражения (если используются переменные, то пользователь должен их инициализировать). Ввод выражения должен быть реализовать двумя способами: с клавиатуры и с файла.

2. Реализует проверку на корректность простого выражения и выражения, записанного в прямой и обратной польских нотациях (на выбор пользователя). Ввод выражения должен быть реализовать двумя способами: с клавиатуры и с файла.

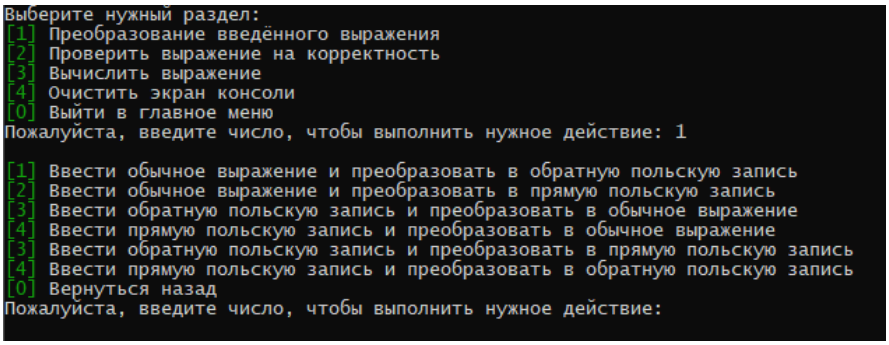
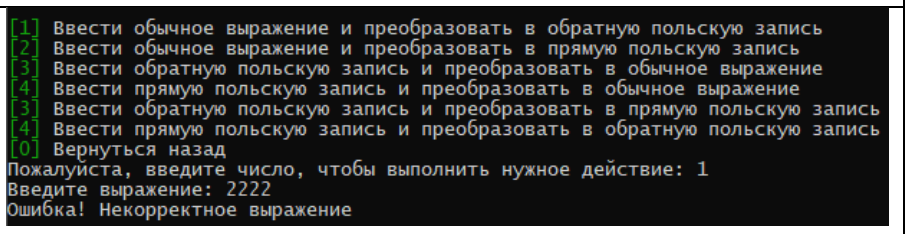
3. Реализует вычисления простого выражения и выражения, записанного в прямой и обратной польских нотациях (на выбор пользователя). Ввод выражения должен быть реализовать двумя способами: с клавиатуры и с файла.

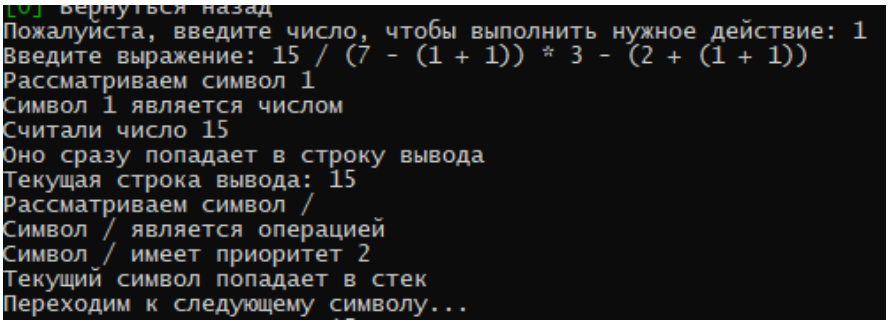
4. Генерирует несколько (на выбор пользователя) вариантов проверочной работы по польской нотации (прямой и обратной). Задание и ответы к ним необходимо вывести в отдельные файлы (ответы должны быть максимально подробными).

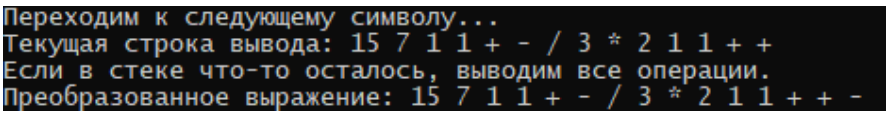
Программа должна выводить и описывать все промежуточные действия.

Выполнение работы.

Код программы представлен в приложении А.

<table>
<tr><td>Ввод пользователем и обработка данных</td><td>Работа алгоритма и вывод на экран</td></tr>
<tr><td colspan="2">Меню</td></tr>
<tr><td>При запуске программы перед пользователем появляется окно с меню, где он может выбрать тип будущей вводимой последовательности.</td><td>Меню:
 
</td></tr>
<tr><td colspan="2">Преобразование выражение из одной записи в другую</td></tr>
<tr><td>Пользователь может выбрать, из какой записи в какую преобразовать введённое им выражение. При этом, каждый раз его выражение будет проверяться на корректность.
Все промежуточные действия преобразования выводятся пользователю на экран</td><td> 




</td></tr>
</table>

Окончание Таблицы

<table>
<tr><td colspan="2">Проверка на корректность</td></tr>
<tr><td>Чтобы проверить выражение на корректность, не преобразовывая его, пользователь может воспользоваться для этого специальной функцией</td><td>
<pre>Выберите, что хотите сделать:
[1] Проверить на корректность простого выражения
[2] Проверить на корректность выражения в обратной польской записи
[3] Проверить на корректность выражения в прямой польской записи
[0] Вернуться назад
Пожалуйста, введите число, чтобы выполнить нужное действие: 1
Введите выражение, которое нужно проверить на корректность: 2+2
Следующее выражение является правильным: 2+2</pre>
В случае, если пользователь ввёл некорректное выражение, он узнает почему:
<pre>Выберите, что хотите сделать:
[1] Проверить на корректность простого выражения
[2] Проверить на корректность выражения в обратной польской записи
[3] Проверить на корректность выражения в прямой польской записи
[0] Вернуться назад
Пожалуйста, введите число, чтобы выполнить нужное действие: 2
Введите выражение, которое нужно проверить на корректность: 2 + 2
Выражение должно иметь в начале два числа или переменные
Выражение содержит ошибку</pre>
</td></tr>
<tr><td colspan="2">Вычисление выражений</td></tr>
<tr><td>Пользователь также может вычислить выражение в любой форме. Промежуточные действия будут также выведены на экран</td><td>
<pre>Выберите, что хотите сделать:
[1] Вычислить обычное выражение
[2] Вычислить выражение в обратной польской записи
[3] Вычислить выражение в прямой польской записи
[0] Вернуться назад
Пожалуйста, введите число, чтобы выполнить нужное действие: 1
Введите выражение, которое нужно вычислить: 2 + 2
Рассматриваем символ 2
Символ 2 является числом
Считали число 2
Оно сразу попадает в строку вывода
Текущая строка вывода: 2
Рассматриваем символ +
Символ + является операцией
Символ + имеет приоритет 1
Текущий символ попадает в стек
Переходим к следующему символу...
Текущая строка вывода: 2
Рассматриваем символ 2
Символ 2 является числом
Считали число 2
Оно сразу попадает в строку вывода
Текущая строка вывода: 2 2
Если в стеке что-то осталось, выводим все операции.
Рассматриваем символ 2
Символ 2 является числом
Получаем число 2. Оно попадает в стек
Рассматриваем символ 2
Символ 2 является числом
Получаем число 2. Оно попадает в стек
Рассматриваем символ +
Символ + является операцией
Вытащили второе выражение из стека 2
Вытащили первое выражение из стека 2
Сложили и получили число 4
Добавляем его в стек
Рассматриваем символ
Ответ: 4</pre>
</td></tr>
</table>

Выводы.

Были получены практические навыки работы со стеками и очередями; изучены обратная и прямая польская нотации; проведён сравнительный анализ этих структур данных.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

Название файла: lab3.cpp

```
#include <iostream>
#include <string>
using namespace std;

//удаляет лишние пробелы, введённые пользователем
string DelSpaces(string s)
{
    for (size_t j = 0; j < s.length(); j++)
    {
        if (s[j] == ' ')
        {
            while (s[j + 1] == ' ') s.erase(j + 1, 1);
        }
    }
    if (s[0] == ' ') s.erase(0, 1);
    if (s[s.length() - 1] == ' ') s.erase(s.length() - 1, 1);
    return s;
}

//проверка, является ли символ цифрой
bool isNumber(char s)
{
    if ((s <= '9') && (s >= '0')) { return true; }
    return false;
}

//проверка, является ли текущий символ операцией. если да - возвращает
его приоритет
int priorOperation(char symb)
{
    switch (symb) {
    case '+':
    case '-':
        return 1;
    case '*':
    case '/':
        return 2;
    case '(':
        return -1;
    case ')':
        return -2;
    default:
        return 0; //не является операцией
    }
}
```

```
//структура стека для символов
struct stack
{
    string data;
    stack* prev = 0;
};

//создание (инициализирование) стека и добавление в него первого элемента
stack* init(string data) {
    stack* tmp = new stack; //создаём начало стека
    tmp->data = data;
    tmp->prev = 0;
    return tmp;
}

//добавление элемента в существующий стек
stack* pushBack(stack* stk, string data) {
    stack* newHead = new stack;
    newHead->data = data;
    newHead->prev = stk;
    return newHead;
}

//добавление элементов в стек
void push(stack*& stk, string data) {
    if (!stk) { stk = init(data); } //если стек пуст
    else { stk = pushBack(stk, data); }
}

//вывод элементов из стека в строку. oper - разный вывод в зависимости от
операции
string outputStk(stack*& head, int oper)
{
    //oper = 0 вывести весь стек
    //oper = -2 вывести стек до "(" уничтожая "("
    //oper = -1 вывести стек до ")" уничтожая ")"
    //oper = 1 вывести весь стек до "(" не уничтожая его
    //oper = 3 вывести верхний элемент
    string output = "\0";

    if (oper == 3)
    {
        output += head->data;
        head = head->prev;
    }
    else
    {
        while (head)
        {
            if (priorOperation(head->data[0]) > 0) //игнорируются скобки
            {
```

```
                output += head->data;
                output += " ";
            }
            head = head->prev;
            if (oper && head && ((head->data == "(")|| (head->data ==
")"))) //если не весь стек, то до "(" или ")"
            {
                if (oper == -2 || oper == -1) { head = head->prev; }
//уничтожается "("
                break; //выход из вывода
            }
        }
    }

    return output; //в конце строки всегда должен быть пробел
}

//обратная польская нотация
string polishNotation(string& expr)
{
    stack* operation = 0; //стек операций
    string push_symb = "\0";
    string result = "\0"; //обработанное выражение

    int k = 0; //индекс строки
    string strval = "\0";
    int prior = 0;

    do {
        if (!expr[k]) { break; }
        cout << "Рассматриваем символ " << expr[k] << "\n";
        prior = priorOperation(expr[k]);
        if (prior) //если является операцией/скобкой
        {
            cout << "Символ " << expr[k] << " является операцией\n";
            //Если подытожить:
            //Игнор если: "++-"
            //НЕ ИГНОР: ")", "*+"
            //Если ")" - текущий символ уничтожается
            //Если не игнор "+" или "-", текущий символ переходит в
пустой стек

            switch (prior)
            {
            case -1: //все открывающиеся скобки - сразу в стек операций
                cout << "Символ " << expr[k] << " является открывающейся
скобкой, поэтому он идёт в стек\n";
                push_symb = expr[k];
                push(operation, push_symb);
                break;
```

```
            case 2: //"*" и "/" - вывод стека, если прошлый символ "*"
или "/"
                cout << "Символ " << expr[k] << " имеет приоритет 2\n";
                if (operation && (priorOperation(operation->data[0]) ==
2))
                {
                    cout << "Приоритет прошлого символа также равен 2,
поэтому происходит выход символов из стека\n";
                    result += outputStk(operation, 1);
                }
                cout << "Текущий символ попадает в стек\n";
                push_symb = expr[k];
                push(operation, push_symb);
                break;
            case -2: //закрывается скобка - выход операций до
открывающейся скобки
                cout << "Символ " << expr[k] << " является закрывающейся
скобкой, поэтому происходит выход всех операций до открывающейся
скобки\n";
                result += outputStk(operation, -2);
                break;
            case 1:
                //перед помещением в стек операций + и -, смотрится
последняя операция в стеке
                //если у текущего символа приоритет меньше или равно -
освобождение стека. Текущий символ помещается в стек
                cout << "Символ " << expr[k] << " имеет приоритет 1\n";
                if (operation && (priorOperation(operation->data[0]) >=
1))
                {
                    result += outputStk(operation, 1);
                    cout << "Приоритет прошлого символа равен 1 или
меньше, поэтому происходит выход символов из стека\n";
                }
                push_symb = expr[k];
                push(operation, push_symb); //если приоритет у текущего
символа больше - в стек
                cout << "Текущий символ попадает в стек\n";
                break;
            }
            k++;
            cout << "Переходим к следующему символу...\n";
        }
        else
        {
            cout << "Символ " << expr[k] << " является числом\n";
            strval = "\0"; //очищение строки
            do { //пока текущий элемент НЕ является операцией
                strval += expr[k]; //сюда попадают ТОЛЬКО числа
                k++;
            } while (expr[k] && !priorOperation(expr[k]));
```

```
            cout << "Считали число " << strval << "\n";
            result += strval; //все числа попадают сразу же в строку
выхода
            cout << "Оно сразу попадает в строку вывода\n";
            result += " "; //пробел после числа
        }
        cout << "Текущая строка вывода: " << result << "\n";
    } while (expr[k]);

    if (operation) { result += outputStk(operation, 0); } //освобождение
всего стека, если там чёт есть
    cout << "Если в стеке что-то осталось, выводим все операции. \n";

    return result;
}

//Обратную польскую в обычное
string fromPolishNotation(string& expr)
{
    string result = "\0"; //обработанное выражение
    stack* numbs = 0; //стек для чисел/выражений
    size_t k = 0; //индекс строки
    string strval = "\0"; //строка для чисел/выражений
    string tmp1 = "\0"
        , tmp2 = "\0";

    //все числа - в стек
    //как встречена операция - забираем числа, используем на них операцию
    do {
        cout << "Рассматриваем символ " << expr[k] << "\n";
        if (priorOperation(expr[k])) //если является операцией
        {
            cout << "Символ " << expr[k] << " является операцией\n";
            strval = "\0"; //очищение строку
            tmp2 = outputStk(numbs, 3); //вытаскиваем второе выражение
            cout << "Вытащили второе выражение из стека " << tmp2 << "
\n";
            tmp1 = outputStk(numbs, 3); //вытаскиваем первое выражение
            cout << "Вытащили первое выражение из стека " << tmp1 << "
\n";
            if (expr[k + 1] && (expr[k] == '+' || expr[k] == '-')) {
strval += "("; } //если НЕ последнее действие, ставит скобки
            strval += tmp1 + expr[k] + tmp2;
            if (expr[k + 1] && (expr[k] == '+' || expr[k] == '-')) {
strval += ")"; }
            cout << "Получившееся выражение: " << strval << ". Добавляем
в стек\n";
            push(numbs, strval); //Добавляем в стек получившееся
выражение
            k++; //переходим на следующий символ
```

```
        }
        else //если число или пробел
        {
            if (isNumber(expr[k]))
            {
                cout << "Символ " << expr[k] << " является числом\n";
                strval = "\0"; //очищение строку
                while (!(expr[k] == ' ')) //пока текущий элемент НЕ
является пробелом
                {
                    strval += expr[k]; //считываем числа
                    k++;
                }
                cout << "Считали число " << strval << ". Оно идёт в
стек\n";
                push(numbs, strval); //все числа попадают в стек
            }
            k++;
        }
    } while (expr[k]);
    result = outputStk(numbs, 3);
    return result;
}

//в прямую польскую нотацию
string directPolishNotation(string& expr)
{
    stack* operation = 0 //стек операций
        , * output = 0; //стек выхода
    string push_symb = "\0";
    string result = "\0"; //обработанное выражение

    int k = expr.length() - 1; //индекс строки
    string strval = "\0";
    int prior = 0;

    do {
        if (!expr[k]) { break; }
        cout << "Рассматриваем символ " << expr[k] << "\n";
        prior = priorOperation(expr[k]);
        if (prior) //если является операцией/скобкой
        {
            cout << "Символ " << expr[k] << " является операцией\n";
            push_symb = "\0";
            //Если подытожить:
            //Игнор если: "++-"
            //НЕ ИГНОР: "(", "*+"
            //Если "(" - текущий символ уничтожается
            //Если не игнор "+" или "-", текущий символ переходит в
пустой стек
```

```
            switch (prior)
            {
            case -2: //все закрывающиеся скобки - сразу в стек операций
                cout << "Символ " << expr[k] << " является закрывающейся
скобкой. Помещаем в стек\n";
                push_symb = expr[k];
                push(operation, push_symb);
                break;
            case 2: //"*" и "/" - сразу в стек операций
                cout << "Символ " << expr[k] << " имеет приоритет 2.
Помещаем в стек\n";
                push_symb = expr[k];
                push(operation, push_symb);
                break;
            case -1: //открывается скобка - выход операций до
закрывающейся скобки
                cout << "Символ " << expr[k] << " является открывающейся
скобкой. Вывод операций до закрывающейся\n";
                result += outputStk(operation, -1);
                break;
            case 1:
                //перед помещением в стек операций + и -, смотрится
последняя операция в стеке
                //если у текущего символа приоритет меньше или равно -
освобождение стека. Текущий символ помещается в стек
                cout << "Символ " << expr[k] << " имеет приоритет 1\n";
                if (operation && (priorOperation(operation->data[0]) >=
1))
                {
                    cout << "Прошлая операция в стеке имеет приоритет
равный 1 или больше. Вывод операций\n";
                    result += outputStk(operation, 1);
                }
                cout << "Символ " << expr[k] << " помещаем в стек\n";
                push_symb = expr[k];
                push(operation, push_symb); //если приоритет у текущего
символа больше - в стек
                break;
            }
            k--;
        }
        else
        {
            cout << "Символ " << expr[k] << " является числом\n";
            strval = "\0"; //очищение строки
            do { //пока текущий элемент НЕ является операцией
                strval += expr[k]; //сюда попадают ТОЛЬКО числа (в
обратной записи)
                k--;
            } while ((k > 0) && !priorOperation(expr[k]));
```

```
            push_symb = "\0";
            for (size_t i = strval.length(); i > 0; i--) { push_symb +=
strval[i-1]; } //отражаем полученное число
            cout << "Получаем число " << push_symb << " и помещаем в
строку вывода\n";

            result += strval + " "; //число сразу в стек выхода
        }
        cout << "Текущая строка вывода: " << result << "\n";
    } while (k >= 0);

    if (operation) { result += outputStk(operation, 0); } //освобождение
всего стека, если там чёт есть
    cout << "Освобождаем стек, если там что-то осталось. Текущая строка
вывода: " << result << "\n";
    strval = result;
    result = "\0";
    cout << "Отражаем строку\n";
    for (size_t i = strval.length(); i > 0; i--) { result += strval[i-1];
} //отражаем полученное выражение

    return result;
}

//Прямую польскую в обычное
string fromDirectPolishNotation(string& expr)
{
    string result = "\0"; //обработанное выражение
    stack* numbs = 0; //стек для чисел/выражений
    int k = expr.length() - 1; //индекс строки
    string strval = "\0" //строка для чисел/выражений
        , tmp1 = "\0"
        , tmp2 = "\0"
        , push_symb ="\0";

    //все числа - в стек
    //как встречена операция - забираем числа, используем на них операцию
    do {
        cout << "Рассматриваем символ " << expr[k] << "\n";
        if (priorOperation(expr[k])) //если является операцией
        {
            cout << "Символ " << expr[k] << " является операцией\n";
            strval = "\0"; //очищение строку
            tmp2 = outputStk(numbs, 3); //вытаскиваем второе выражение
            cout << "Вытащили второе выражение из стека " << tmp2 << "
\n";
            tmp1 = outputStk(numbs, 3); //вытаскиваем первое выражение
            cout << "Вытащили первое выражение из стека " << tmp1 << "
\n";
            if ((k != 0) && (expr[k] == '+' || expr[k] == '-')) { strval
+= "("; } //если НЕ последнее действие, ставит скобки
```

```
            strval += tmp2 + expr[k] + tmp1;
            if ((k != 0) && (expr[k] == '+' || expr[k] == '-')) { strval
+= ")"; }
            cout << "Получившееся выражение: " << strval << ". Добавляем
в стек\n";
            push(numbs, strval); //Добавляем в стек получившееся
выражение
            k--; //переходим на следующий символ
        }
        else //если число или пробел
        {
            if (isNumber(expr[k]))
            {
                cout << "Символ " << expr[k] << " является числом\n";
                strval = "\0"; //очищение строки
                while (!(expr[k] == ' ')) //пока текущий элемент НЕ
является пробелом
                {
                    strval += expr[k]; //считываем числа
                    k--;
                }

                push_symb = "\0";
                for (size_t i = strval.length(); i > 0; i--) { push_symb
+= strval[i - 1]; } //отражаем полученное число
                cout << "Считали число " << push_symb << ". Оно идёт в
стек\n";

                push(numbs, push_symb); //все числа попадают в стек
            }
            k--;
        }
    } while (k >= 0);
    result = outputStk(numbs, 3);
    return result;
}

//инициализация переменных. возвращает 0, если всё ок
bool initVar(string& expr)
{
    char sw = '\0';
    for (size_t i = 0; i < expr.length() - 1; i++)
    {
        if (!priorOperation(expr[i]) && !isNumber(expr[i])) //если не
является операцией и не является числом
        {
            cout << "Найден неизвестный символ: " << expr[i] << ".
Инициализировать его как переменную? (Y/N)\n";
            cin >> sw;
            while (cin.get() != '\n') { sw = ' '; }; //если строка
содержит более одного символа, возвращается ошибка
```

```
            switch (sw)
            {
            case 'N':
                return 1;
            case 'Y':
                string toReplace = "\0"
                    , replaceWith = "\0";
                toReplace = expr[i];
                cout << "Введите значение переменной " << expr[i] << ":
";
                getline(cin, replaceWith);
                for (size_t j = 0; j < replaceWith.length() - 1; j++)
                {
                    if (!isNumber(replaceWith[j])) //если встречено не
число
                    {
                        cout << "Значение переменной не может содержать
иные символы кроме цифр.\n";
                        return 1;
                    }
                }
                size_t pos = 0;
                while ((pos = expr.find(toReplace, pos)) != string::npos)
                {
                    expr.replace(pos, toReplace.size(), replaceWith);
                    pos += replaceWith.size();
                }
                i = 0;
                break;
            }
        }
    }
    return 0;
}

//проверка на корректность ввода и обработка выражения. возвращает 1,
если найдена ошибка в вводе, 0 если всё ок
bool invalidInput(string& expr)
{
    if (priorOperation(expr[0]) > 0) { cout << "Выражение не может
начинаться операцией\n"; return 1; } //выражение не может иметь операцию
в начале
    if (priorOperation(expr[expr.length() - 1]) > 0) { cout << "Выражение
не может заканчиваться операцией\n"; return 1; } //выражение не может
иметь операцию в конце

    //удаление всех пробелов
    for (size_t j = 1; j < expr.length() - 1; j++) //пробел не может
стоять между числами
    {
```

```
        if (expr[j] == ' ')
        {
            if (!(priorOperation(expr[j - 1]) || priorOperation(expr[j +
1]))) //если пробелом разделены два числа
            {
                cout << "Выражение содержит лишние символы\n"; return 1;
//ввод некорректный
            }
            expr.erase(j, 1); //если пробел разделяет число и операцию -
стираем пробел
        }
    }

    int opBrace = 0,
        cBrace = 0;
    bool op = false;
    //добавляет "*" там, где его не хватает. Проверяет корректность
потенциальных переменных
    for (size_t i = 0; i < expr.length(); i++)
    {
        if (!expr[i]) { break; }
        if (!(isNumber(expr[i]) || priorOperation(expr[i]))) //не
является числом или операцией (т.е. потенциальная переменная)
        {
            //работаем со следующим символом
            if (expr[i + 1] != '\0') //если он существует
            {
                if (expr[i] == expr[i + 1]) { cout << "Выражение содержит
лишние символы\n"; return 1; } //если две одинаковой переменной подряд -
ошибка
                if (isNumber(expr[i + 1])) { cout << "Выражение содержит
лишние символы\n"; return 1; } //если следующий символ число - ошибка
                if (!(priorOperation(expr[i + 1]) && expr[i + 1] != '('))
//если является операцией и не ")" - ничего не делаем
                {
                    expr.insert(i + 1, "*"); //если "(" или другая
переменная - вставляем "*"
                }
            }

            //работа с предыдущим символом
            if (i > 0) //если он существует
            {
                if (!(priorOperation(expr[i - 1]) && expr[i - 1] != ')'))
//если является операцией и не ")" - ничего не делаем
                {
                    expr.insert(i, "*"); //если число или ")" - вставляем
"*"
                }
            }
            //таким образом:
```

```
            // x5 xx - ошибка
            // x* x+ *x +x x) (x - игнор
            // xb )x x( 5x - превращается в x*b )*x x*( 5*x
        }

        if (isNumber(expr[i])) //является числом
        {
            //работа со следующим символом || 2( превращается в 2*(
            if (expr[i + 1] != '\0') //если он существует
            {
                if (expr[i + 1] == '(') { expr.insert(i + 1, "*"); }
//если "(" - вставляем "*"
            }

            //работа с предыдущим символом  || )2 превращается в )*2
            if (i > 0) //если он существует
            {
                if (expr[i - 1] == ')') { expr.insert(i, "*"); } //если
")" - вставляем "*"
            }
        }

        if (priorOperation(expr[i]) > 0) //если является операцией, не
скобки
        {
            op = true; //запоминаем, что в выражении есть операция
            //работа со следующим символом
            if (expr[i + 1] != '\0') //если он существует
            {
                if (priorOperation(expr[i + 1]) > 0) { cout << "Две
операции не могут находиться рядом\n"; return 1; } //если две операции
рядом - ошибка
                if (priorOperation(expr[i + 1]) == ')') { cout <<
"Неправильный порядок скобок\n"; return 1; } //если после операции ")" -
ошибка. пример: *)
            }

            //работа со следующим символом
            if (i > 0) //если он существует
            {
                if (priorOperation(expr[i - 1]) == '(') { cout <<
"Неправильный порядок скобок\n"; return 1; } //если после операции "(" -
ошибка. пример: (*
            }
        }

        if (expr[i] == '(')
        {
            opBrace++; //подсчёт "("
            //внутри скобок должны быть как минимум одна операция с двумя
переменными/числами
```

```
            size_t j = i + 1;
            bool check = false;
            while (expr[j] != '\0') //поиск символа-операции
            {
                switch (priorOperation(expr[j]))
                {
                case -1:
                case 0: //если не операция или "(", скип
                    j++;
                    break;
                case 1:
                case 2: //если встречена операция + - * /, запоминаем
                    check = true;
                    break;
                case -2: //если ")" встречена раньше других операций - 
ошибка
                    cout << "Неправильный порядок скобок\n"; return 1;
                }
                if (check) { break; }
            }
        }
        if (expr[i] == ')')
        {
            cBrace++; //подсчёт ")"
            if (cBrace > opBrace) { cout << "Неправильный порядок 
скобок\n"; return 1; } //если закрывающаяся скобка встречена раньше 
открывающейся - ошибка
        }
    }
    if (!op) { cout << "Количество операций не соответствует количеству 
чисел\n"; return 1; } //если в выражении нет операций, это ошибка
    if (opBrace != cBrace) { cout << "Количество операций не 
соответствует количеству чисел\n"; return 1; } //если количество скобок 
не совпадает

    if (initVar(expr)) { return 1; }

    return 0;
}

//Проверка на корректность написания обратной польской
bool invalidInputPolish (string& expr)
{
    //Первые два объекта - числа или переменные
    size_t j = 0;
    while (expr[j] != ' ')
    {
        if (priorOperation(expr[j])) { cout << "Выражение должно иметь в 
начале два числа или переменные\n"; return 1; }
        j++;
```

```
        if (j == expr.length()) { cout << "Количество операций не
соответствует количеству чисел\n"; return 1; } //если не найдено пробела
или операции
    }
    j++;
    while (expr[j] != ' ')
    {
        if (priorOperation(expr[j])) { cout << "Выражение должно иметь в
начале два числа или переменные\n"; return 1; }
        j++;
        if (j == expr.length()) { cout << "Количество операций не
соответствует количеству чисел\n"; return 1; } //если не найдено пробела
или операции
    }

    //Всегда заканчивается на операцию
    if (!priorOperation(expr[expr.length() - 1])) { cout << "Выражение
должно заканчиваться операцией\n"; return 1; }

    //Количество операций = количество чисел - 1
    size_t oper = 0
        , numbs = 0;
    for (size_t i = 0; i < expr.length(); i++)
    {
        if (priorOperation(expr[i])) { oper++; }
        else //не должно быть переменной+число, т.е. 5x x5 sg5d 3436f3
        {
            if (expr[i] != ' ') //пробел пропускаем
            {
                if (!isNumber(expr[i])) //если встречена потенциальная
переменная
                {
                    if (expr[i + 1] != ' ') { cout << "Выражение имеет
лишние символы\n"; return 1; } //после неё должен быть пробел
                    numbs++; //если всё ок, считаем её за число
                }
                else //если встречено число
                {
                    j = i + 1;
                    while (expr[j] != ' ') //проверяем, нет ли внутри
числа лишнего до пробела
                    {
                        if (!isNumber(expr[i])) { cout << "Выражение
имеет лишние символы\n"; return 1; }
                        j++;
                    }
                    i = j; //пропускаем все символы до пробела
                    numbs++;
                }
            }
        }
```

```
    }
    if (oper != (numbs - 1)) { cout << "Количество операций не
соответствует количеству чисел\n"; return 1; }

    if (initVar(expr)) { return 1; }

    return 0;
}

//Проверка прямой польской
bool invalidInputDirectPolish (string& expr)
{
    //Последние два объекта - числа или переменные
    size_t j = expr.length() - 1;
    while (expr[j] != ' ')
    {
        if (priorOperation(expr[expr.length() - 1])) { cout << "Выражение
должно иметь в конце два числа или переменные\n"; return 1; }
        j--;
        if (j == 0) { cout << "Количество операций не соответствует
количеству чисел\n"; return 1; } //если не найдено пробела или операции
    }
    j--;
    while (expr[j] != ' ')
    {
        if (priorOperation(expr[expr.length() - 1])) { cout << "Выражение
должно иметь в конце два числа или переменные\n"; return 1; }
        j--;
        if (j == 0) { cout << "Количество операций не соответствует
количеству чисел\n"; return 1; } //если не найдено пробела или операции
    }

    //Всегда начинается на операцию
    if (!priorOperation(expr[0])) { cout << "Выражение должно начинаться
с операции\n"; return 1; }

    //Количество операций = количество чисел - 1
    size_t oper = 0
        , numbs = 0;
    for (size_t i = 0; i < expr.length(); i++)
    {
        if (priorOperation(expr[i])) { oper++; }
        else //не должно быть переменной+число, т.е. 5x x5 sg5d 3436f3
        {
            if (expr[i] != ' ') //пробел пропускаем
            {
                if (!isNumber(expr[i])) //если встречена потенциальная
переменная
                {
                    if (expr[i + 1] != ' ') { cout << "Выражение содержит
лишние символы\n"; return 1; } //после неё должен быть пробел
```

```
                    numbs++; //если всё ок, считаем её за число
                }
                else //если встречено число
                {
                    j = i + 1;
                    while (j < expr.length() && expr[j] != ' ')
//проверяем, нет ли внутри числа лишних символов до пробела
                    {
                        if (!isNumber(expr[i])) { cout << "Выражение
содержит лишние символы\n"; return 1; }
                        j++;
                    }
                    i = j; //пропускаем все символы до пробела
                    numbs++;
                }
            }
        }
    }
    if (oper != (numbs - 1)) { cout << "Количество операций не
соответствует количеству чисел\n"; return 1; }

    if (initVar(expr)) { return 1; }

    return 0;
}

//Вычисление в обратной
float calculate(string expr)
{
    float result = 0; //обработанное выражение
    stack* numbs = 0; //стек для чисел
    size_t k = 0; //индекс строки
    string strval = "\0"; //строка для чисел/выражений
    float tmp1 = 0
        , tmp2 = 0;

    //все числа - в стек
    //как встречена операция - забираем числа, используем на них операцию
    do {
        cout << "Рассматриваем символ " << expr[k] << "\n";
        if (priorOperation(expr[k])) //если является операцией
        {
            cout << "Символ " << expr[k] << " является операцией\n";
            strval = "\0"; //очищение строку
            tmp2 = stof(outputStk(numbs, 3)); //вытаскиваем второе
выражение
            cout << "Вытащили второе выражение из стека " << tmp2 << "
\n";
            tmp1 = stof(outputStk(numbs, 3)); //вытаскиваем первое
выражение
```

```
            cout << "Вытащили первое выражение из стека " << tmp1 << "
\n";
            switch (expr[k])
            {
            case '+':
                result = tmp1 + tmp2;
                cout << "Сложили и получили число " << result << " \n";
                break;
            case '-':
                result = tmp1 - tmp2;
                cout << "Вычли и получили число " << result << " \n";
                break;
            case '*':
                result = tmp1 * tmp2;
                cout << "Умножили и получили число " << result << " \n";
                break;
            case '/':
                result = tmp1 / tmp2;
                cout << "Разделили и получили число " << result << " \n";
                break;
            }
            strval = to_string(result);
            cout << "Добавляем его в стек \n";
            push(numbs, strval); //Добавляем в стек получившееся
выражение
            k++; //переходим на следующий символ
        }
        else //если число или пробел
        {
            if (isNumber(expr[k]))
            {
                cout << "Символ " << expr[k] << " является числом\n";
                strval = "\0"; //очищение строку
                while (!(expr[k] == ' ')) //пока текущий элемент НЕ
является пробелом
                {
                    strval += expr[k]; //считываем числа
                    k++;
                }
                cout << "Получаем число " << strval << ". Оно попадает в
стек\n";
                push(numbs, strval); //все числа попадают в стек
            }
            k++;
        }
    } while (expr[k]);

    result = stof(outputStk(numbs, 3)); //вытаскиваем получившееся выра-
жение
    return result;
}
```

```
// == ФУНКЦИИ ПО ПУНКТАМ ==
//Обычное выражение в обратную польскую нотацию
string toPolishNotation()
{
    string expr; //выражение пользователя
    string result; //результат преобразования
    //Ввод выражения двумя способами: с клавиатуры и * с файла
    cout << "Введите выражение: ";
    getline(cin, expr);
    expr = DelSpaces(expr); //удаляет пробелы в начале и в конце, пробе-
лы, идущие подряд

    if (!invalidInput(expr))
    {
        result = polishNotation(expr);
        cout << "Преобразованное выражение: " << result << '\n';
    }
    else { cout << "Ошибка! Некорректное выражение\n"; return "\0"; }

    //После обработанного выражения:

    return result;
}

//Обратную польскую в обычное
string fromPolishNotationToInfix()
{
    string expr; //выражение пользователя
    string result; //результат преобразования
    //Ввод выражения двумя способами: с клавиатуры и * с файла
    cout << "Введите выражение: ";
    getline(cin, expr);
    expr = DelSpaces(expr); //удаляет пробелы в начале и в конце, пробе-
лы, идущие подряд

    if (!invalidInputPolish(expr))
    {
        result = fromPolishNotation(expr);
        cout << "Преобразованное выражение: " << result << '\n';
    }
    else { cout << "Ошибка! Некорректное выражение\n"; return "\0"; }

    //После обработанного выражения:
    return result;
}

//Обычное выражение в прямую польскую нотацию
string toDirectPolishNotation()
{
    string expr; //выражение пользователя
```

```
    string result; //результат преобразования
    //Ввод выражения двумя способами: с клавиатуры и * с файла
    cout << "Введите выражение: ";
    getline(cin, expr);
    expr = DelSpaces(expr); //удаляет пробелы в начале и в конце, пробе-
лы, идущие подряд

    if (!invalidInput(expr))
    {
        result = directPolishNotation(expr);
        cout << "Преобразованное выражение: " << result << "\n";
    }
    else { cout << "Ошибка! Некорректное выражение\n"; return "\0"; }

    //После обработанного выражения:

    return result;
}

//Прямую польскую в обычное
string fromDirectPolishNotationToInfix()
{
    string expr; //выражение пользователя
    string result; //результат преобразования
    //Ввод выражения двумя способами: с клавиатуры и * с файла
    cout << "Введите выражение: ";
    getline(cin, expr);
    expr = DelSpaces(expr); //удаляет пробелы в начале и в конце, пробе-
лы, идущие подряд

    if (!invalidInputDirectPolish(expr))
    {
        result = fromDirectPolishNotation(expr);
        cout << "Преобразованное выражение: " << result << "\n";
    }
    else { cout << "Ошибка! Некорректное выражение\n"; return "\0"; }

    //После обработанного выражения:

    return result;

}

//Обратную польскую в прямую
string fromPolishNotationToDirect()
{
    string expr; //выражение пользователя
    string result; //результат преобразования
    //Ввод выражения двумя способами: с клавиатуры и * с файла
    cout << "Введите выражение: ";
    getline(cin, expr);
```

```
    expr = DelSpaces(expr); //удаляет пробелы в начале и в конце, пробе-
лы, идущие подряд

    if (!invalidInputPolish(expr))
    {
        result = fromPolishNotation(expr);
        result = directPolishNotation(result);
        cout << "Преобразованное выражение: " << result << '\n';
    }
    else { cout << "Ошибка! Некорректное выражение\n"; return "\0"; }

    //После обработанного выражения:
    return result;
}

//Прямую польскую в обратную
string fromDirectToPolishNotation()
{
    string expr; //выражение пользователя
    string result; //результат преобразования
    //Ввод выражения двумя способами: с клавиатуры и * с файла
    cout << "Введите выражение: ";
    getline(cin, expr);
    expr = DelSpaces(expr); //удаляет пробелы в начале и в конце, пробе-
лы, идущие подряд

    if (!invalidInputDirectPolish(expr))
    {
        result = fromDirectPolishNotation(expr);
        result = polishNotation(result);
        cout << "Преобразованное выражение: " << result << '\n';
    }
    else { cout << "Ошибка! Некорректное выражение\n"; return "\0"; }

    //После обработанного выражения:
    return result;
}

//Проверка на корректность обычного выражения
void taskInvalidInput()
{
    string expr; //выражение пользователя
    cout << "Введите выражение, которое нужно проверить на корректность:
";
    getline(cin, expr);
    expr = DelSpaces(expr); //удаляет пробелы в начале и в конце,
пробелы, идущие подряд

    if (!invalidInput(expr)) { cout << "Следующее выражение является
правильным: " << expr << "\n"; }
    else { cout << "Выражение содержит ошибку\n"; }
```

```
}

//Проверка на корректность обратной польской записи
void taskInvalidInputPolish()
{
    string expr; //выражение пользователя
    cout << "Введите выражение, которое нужно проверить на корректность:
";
    getline(cin, expr);
    expr = DelSpaces(expr); //удаляет пробелы в начале и в конце,
пробелы, идущие подряд

    if (!invalidInputPolish(expr)) { cout << "Следующее выражение
является правильным: " << expr << "\n"; }
    else { cout << "Выражение содержит ошибку\n"; }
}

//Проверка на корректность прямой польской записи
void taskInvalidInputDirectPolish()
{
    string expr; //выражение пользователя
    cout << "Введите выражение, которое нужно проверить на корректность:
";
    getline(cin, expr);
    expr = DelSpaces(expr); //удаляет пробелы в начале и в конце,
пробелы, идущие подряд

    if (!invalidInputDirectPolish(expr)) { cout << "Следующее выражение
является правильным: " << expr << "\n"; }
    else { cout << "Выражение содержит ошибку\n"; }
}

//Вычисление в обычной форме
void taskCalculateInfix()
{
    string expr; //выражение пользователя
    cout << "Введите выражение, которое нужно вычислить: ";
    getline(cin, expr);
    expr = DelSpaces(expr); //удаляет пробелы в начале и в конце,
пробелы, идущие подряд

    if (!invalidInput(expr))
    {
        expr = polishNotation(expr);
        cout << "Ответ: " << calculate(expr) << "\n";
    }
    else { cout << "Выражение некорректно\n"; }
}

//Вычисление обратной польской
void taskCalculate()
```

```
{
    string expr; //выражение пользователя
    cout << "Введите выражение, которое нужно вычислить: ";
    getline(cin, expr);
    expr = DelSpaces(expr); //удаляет пробелы в начале и в конце,
пробелы, идущие подряд

    if (!invalidInputPolish(expr))
    {
        cout << "Ответ: " << calculate(expr) << "\n";
    }
    else { cout << "Выражение некорректно\n"; }
}

//Вычисление прямой польской
void taskCalculateDirect()
{
    string expr; //выражение пользователя
    cout << "Введите выражение, которое нужно вычислить: ";
    getline(cin, expr);
    expr = DelSpaces(expr); //удаляет пробелы в начале и в конце,
пробелы, идущие подряд

    if (!invalidInputDirectPolish(expr))
    {
        expr = fromDirectPolishNotation(expr);
        expr = polishNotation(expr);
        cout << "Ответ: " << calculate(expr) << "\n";
    }
    else { cout << "Выражение некорректно\n"; }
}

int main()
{
    setlocale(0, "");
    string result = "\0"; //строка выхода
    float res = 0;

    bool check = true; //выход из меню
    bool check1 = false; //выход из подменю
    //false - заканчивает цикл, приводя непосредственно к выходу
    do {
        char sw = ' '; //переключатель главного меню
        char sw1 = ' '; //переключатель саб-меню
        cout << "\nВыберите нужный раздел: \n";
        cout << "\x1b[32m[1]\x1b[0m Преобразование введённого
выражения\n";
        cout << "\x1b[32m[2]\x1b[0m Проверить выражение на
корректность\n";
        cout << "\x1b[32m[3]\x1b[0m Вычислить выражение\n";
        cout << "\x1b[32m[4]\x1b[0m Очистить экран консоли\n";
```

```
        cout << "\x1b[32m[0]\x1b[0m Выйти в главное меню\n";
        cout << "Пожалуйста, введите число, чтобы выполнить нужное
действие: ";

        cin >> sw;
        while (cin.get() != '\n') { sw = ' '; }; //если строка содержит
более одного символа, возвращается ошибка

        switch (sw)
        {

        case '1': //[1] Преобразование введённого выражения
            do {
                check1 = false;
                sw1 = ' ';
                cout << "\n\x1b[32m[1]\x1b[0m Ввести обычное выражение и
преобразовать в обратную польскую запись\n";
                cout << "\x1b[32m[2]\x1b[0m Ввести обычное выражение и
преобразовать в прямую польскую запись\n";
                cout << "\x1b[32m[3]\x1b[0m Ввести обратную польскую
запись и преобразовать в обычное выражение\n";
                cout << "\x1b[32m[4]\x1b[0m Ввести прямую польскую запись
и преобразовать в обычное выражение\n";
                cout << "\x1b[32m[3]\x1b[0m Ввести обратную польскую
запись и преобразовать в прямую польскую запись\n";
                cout << "\x1b[32m[4]\x1b[0m Ввести прямую польскую запись
и преобразовать в обратную польскую запись\n";
                cout << "\x1b[32m[0]\x1b[0m Вернуться назад\n";
                cout << "Пожалуйста, введите число, чтобы выполнить
нужное действие: ";

                cin >> sw1;
                while (cin.get() != '\n') { sw1 = ' '; };

                switch (sw1)
                {
                case '1': //[1] обычное выражение => обратная польская
                    toPolishNotation();
                    break;
                case '2': //[2] обычное выражение => прямая польская
                    toDirectPolishNotation();
                    break;
                case '3': //[3] обратная польская => обычное выражение
                    fromPolishNotationToInfix();
                    break;
                case '4': //[4] прямая польская => обычное выражение
                    fromDirectPolishNotationToInfix();
                    break;
                case '5': //[5] обратная польская => прямая польская
                    fromPolishNotationToDirect();
                    break;
```

```
                case '6': //[6] прямая польская => обратная польская
                    fromDirectToPolishNotation();
                    break;
                case '0': //[0] Назад
                    break;
                default:
                    cout << "Ошибка! Пожалуйста, попробуйте снова\n";
                    check1 = true; //цикл пойдёт заново
                    break;
                }
            } while (check1);
            break;

        case '2': //[2] Проверить выражение на корректность
            do {
                check1 = false;
                sw1 = ' ';
                cout << "\nВыберите, что хотите сделать: \n";
                cout << "\x1b[32m[1]\x1b[0m Проверить на корректность
простого выражения\n";
                cout << "\x1b[32m[2]\x1b[0m Проверить на корректность
выражения в обратной польской записи\n";
                cout << "\x1b[32m[3]\x1b[0m Проверить на корректность
выражения в прямой польской записи\n";
                cout << "\x1b[32m[0]\x1b[0m Вернуться назад\n";
                cout << "Пожалуйста, введите число, чтобы выполнить
нужное действие: ";

                cin >> sw1;
                while (cin.get() != '\n') { sw1 = ' '; };

                switch (sw1)
                {
                case '1': //[1] корректность простого выражения
                    taskInvalidInput();
                    break;
                case '2': //[2] в обратной польской записи
                    taskInvalidInputPolish();
                    break;
                case '3': //[3] в прямой польской записи
                    taskInvalidInputDirectPolish();
                    break;
                case '0': //[0] Назад
                    break;
                default:
                    cout << "Ошибка! Пожалуйста, попробуйте снова\n";
                    check1 = true; //цикл пойдёт заново
                    break;
                }
            } while (check1);
            break;
```

```
        case '3': //[3] Вычислить выражение
            do {
                check1 = false;
                sw1 = ' ';
                cout << "\nВыберите, что хотите сделать: \n";
                cout << "\x1b[32m[1]\x1b[0m Вычислить обычное
выражение\n";
                cout << "\x1b[32m[2]\x1b[0m Вычислить выражение в
обратной польской записи\n";
                cout << "\x1b[32m[3]\x1b[0m Вычислить выражение в прямой
польской записи\n";
                cout << "\x1b[32m[0]\x1b[0m Вернуться назад\n";
                cout << "Пожалуйста, введите число, чтобы выполнить
нужное действие: ";

                cin >> sw1;
                while (cin.get() != '\n') { sw1 = ' '; };

                switch (sw1)
                {
                case '1': //[1] Вычислить обычное выражение
                    taskCalculateInfix();
                    break;
                case '2': //[2] Вычислить выражение в обратной польской
записи
                    taskCalculate();
                    break;
                case '3': //[3] Вычислить выражение в прямой польской
записи
                    taskCalculateDirect();
                    break;
                case '0': //[0] Назад
                    break;
                default:
                    cout << "Ошибка! Пожалуйста, попробуйте снова\n";
                    check1 = true; //цикл пойдёт заново
                    break;
                }
            } while (check1);
            break;

        case '4': //[4] Очистка экрана
            system("cls");
            break;

        case '0': //[0] Закрыть программу
            cout << "Выход из программы...\n";
            check = false; //выход из цикла
            break;
```

```
        default: //в случае, если введено что-то иное
            cout << "Ошибка! Пожалуйста, попробуйте снова\n";
            break;
        }

    } while (check);

    system("Pause");
    return 0;
}
```