Федеральное государственное автономное образовательное учреждение

высшего образования

«Нижегородский государственный университет

им. Н.И. Лобачевского»

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе

Вычисление арифметических выражений

Выполнил:

студент ИИТММ ПМИ гр. 382003-1

Линев. Е.А.

Проверил:

Ассистент кафедры МОСТ ИИТММ

Волокитин В. Д.

Нижний Новгород

2021 г.

Оглавление

[Введение 3](#_Toc91617268)

[Постановка задачи 4](#_Toc91617269)

[Руководство пользователя 5](#_Toc91617270)

[Описание функций и классов 6](#_Toc91617271)

[Файл «stack.h» 6](#_Toc91617272)

[Файл «arithmetic.h» 7](#_Toc91617273)

[Описание алгоритмов 8](#_Toc91617274)

[Сравнение приоритетов двух операций (*bool PolishHelp)*: 8](#_Toc91617275)

[Удаление пробелов (*string DelateSpaces)* 8](#_Toc91617276)

[Проверка изначального выражения на корректность ввода (bool exam) 8](#_Toc91617277)

[Разбиение строки на лексемы (*vector <string> start)* 9](#_Toc91617278)

[Построение польской записи и вычисление результата (*double Polish)* 9](#_Toc91617279)

[Часть 1. Превращение в польскую запись. 9](#_Toc91617280)

[Часть 2. Вычисление результата. 10](#_Toc91617281)

[Описание файловой системы 11](#_Toc91617282)

[Заключение 13](#_Toc91617283)

[Приложение (код функций) 14](#_Toc91617284)

[Сравнение приоритетов двух операций 14](#_Toc91617285)

[Удаление пробелов в введенном выражении 14](#_Toc91617286)

[Проверка изначального выражения на корректность ввода 14](#_Toc91617287)

[Разбиение строки на лексемы 16](#_Toc91617288)

[Построение польской записи и вычисление результата 18](#_Toc91617289)

# Введение

В различных практических задачах люди зачастую сталкиваются с громоздкими выражениями, значение которых необходимо вычислить. Считать результат вручную -очень непривлекательная перспектива, и потому возникает необходимость разработки программы, которая умеет работать с арифметическими выражениями и способна быстро их вычислять.

# Постановка задачи

Требуется разработать программу, выполняющую построение польской записи на основе исходного выражения, а затем вычисление результата. Как итог пользователь получает вещественное число.

Выражение в качестве операндов может содержать переменные и вещественные числа, а также переменные х0, х1, х2. При вводе переменных пользователю далее будет необходимо ввести их числовые значения.

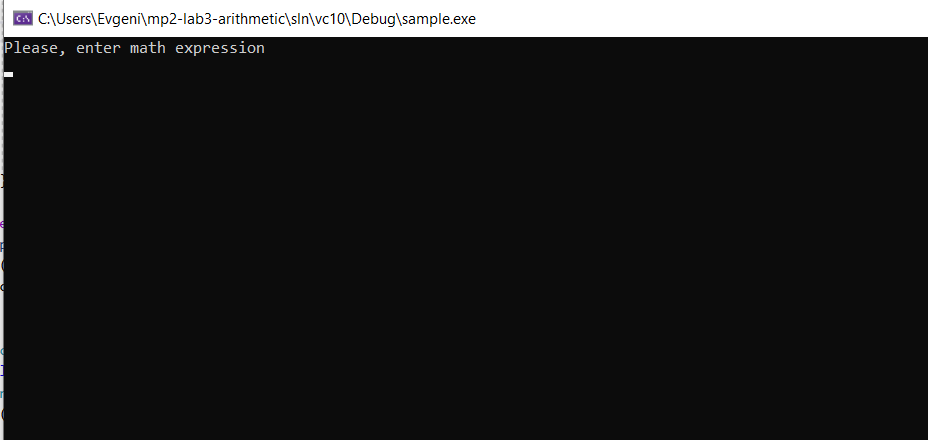
Допустимые операции известны: +, -, /, \*, а также закрывающиеся и открывающиеся скобки. Допускается наличие знака "-" в начале выражения или после открывающей скобки, но в этом случае символ "~" будет изменен на "~". Программа должна выполнять предварительную проверку корректности выражения и сообщать пользователю: ошибка в изначальном вводе; ошибка после ввода х.

Программой предусмотрено своевременное избавление от пробелов, поэтому пользователь может вводить свое выражение вместе с пробелами.

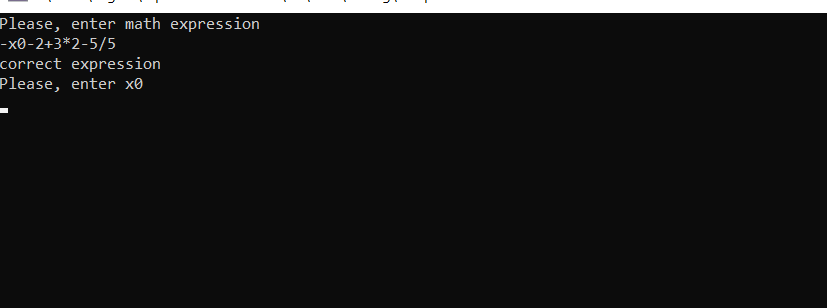
Отдельно все функции тестируются с помощью GoogleTest – ов. Для этого необходимо запустить файл «test\_main.cpp».

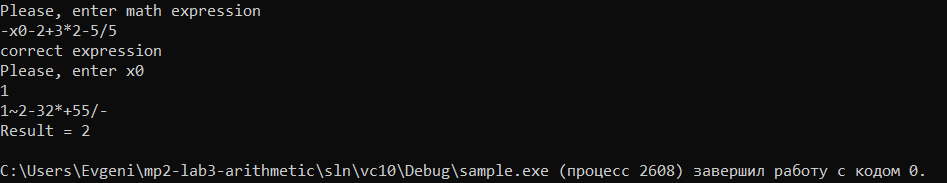
# Руководство пользователя

Для начала работы необходимо запустить файл «main\_arithmetic.cpp», после чего пользователю будет предложено ввести требуемое выражение.



Предположим, что исходное выражение следующее: -x0-2+3\*2-5/5.

Это было корректное выражение, согласно оглашенным ранее условиям, и программа это подтвердила. Далее предлагается ввести х0. Пусть х0=1.

После ввода х0 запускается преобразование выражения в польскую запись, которая выводится на экран. Затем производятся вычисления, и пользователь получает на выходе требуемый результат.

В данном случае посчитать итог вручную не составляет труда: значение арифметического выражения равно двум. Программа получила тот же результат.

# Описание функций и классов

## Файл «stack.h»

В этом файле описана реализация класса TStack. Эта структура предназначена для хранения данных и используется для реализации алгоритмов построения польской записи и вычисления результата. Помимо конструкторов и диструктора, класс имеет следующие методы:

* *void push – вставка элемента;*
* *bool IsEmpty – проверка на пустоту;*
* *T pop – извлечение элемента;*
* *void clear – очистка стэка;*
* *int GetSize - получение количества элементов;*
* *T peek – просмотр верхнего элемента без удаления.*

Реализация класса:

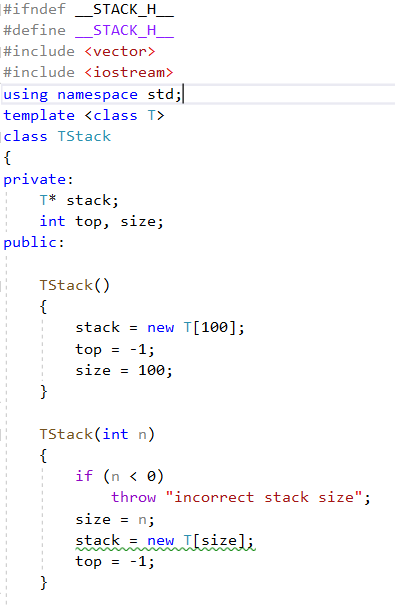


Рисунок . Класс TStack picture 1

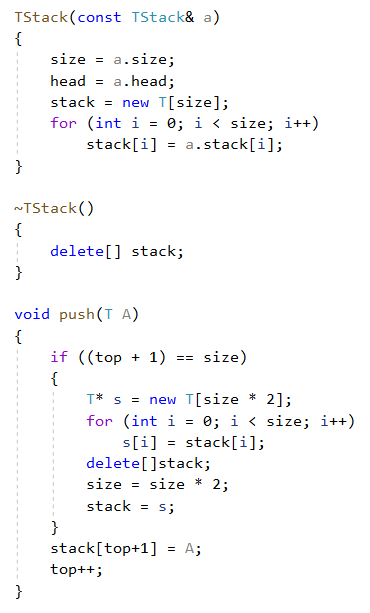


Рисунок . Класс TStack picture 2

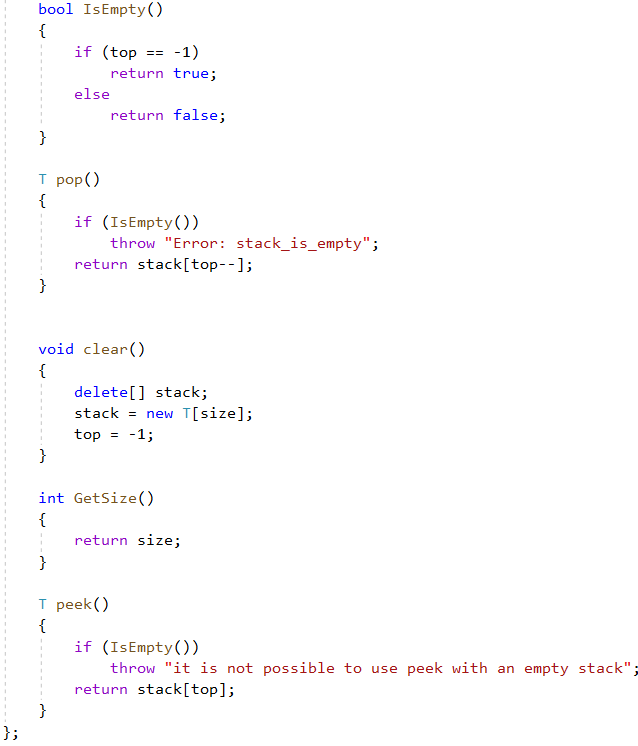
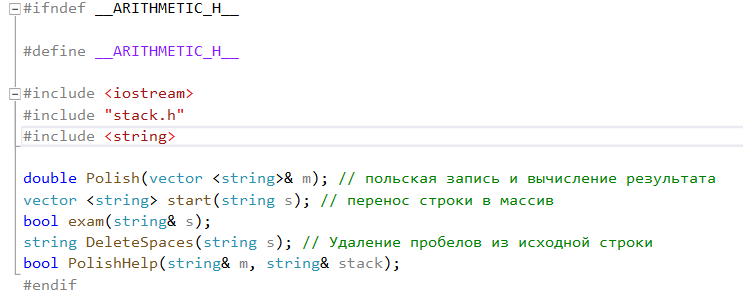


Рисунок . Класс TStack picture 3

## Файл «arithmetic.h»

В этом файле приведены прототипы функций, которые далее будут использоваться при построении польской записи и вычислении результата:

* *double Polish – построение польской записи и вычисление результата;*
* *vector <string> start – разбиение строки на лексемы;*
* *bool exam – проверка изначального выражения на корректность ввода;*
* *string DelateSpaces – удаление пробелов в введенном выражении;*
* *bool PolishHelp – сравнение приоритетов двух операций.*

**

# Описание алгоритмов

## Сравнение приоритетов двух операций (*bool PolishHelp)*:

Функция будет использоваться только при построении польской записи. По большей части она создана для читабельности кода. На вход подаются две строки, которые заведомо могут быть лишь следующими: "-", "+", "\*", "/", ")", "(", "~". Приоритет операций следующий:

* 0 – );
* 1 – (;
* 2 – -, +;
* 3 – \*, /;
* 4 - ~;

На основе этой системы идет сравнение приоритетов двух значений. Если приоритет первого значения меньше или равен приоритету второго, то функция возвращает значение True, иначе - False.

## Удаление пробелов (*string DelateSpaces)*

Работает функция следующим образом:

1. На вход подается исходная строка;
2. Заводится новая строка;
3. С помощью цикла проходимся по исходной строке. Если текущее значение не пробел, то заносим его в новую строку.
4. Возвращаем новую строку без символов.

На выходе получаем строку.

## Проверка изначального выражения на корректность ввода (bool exam)

Работает функция следующим образом:

1. На вход подается строка без пробелов;
2. С помощью цикла идет проверка корректности ввода по всем элементам. Функция switch помогает проверить текущий элемент.

* Если текущий элемент х, то следующий может быть только 1, 2 или 3;
* Если текущий элемент ".", то следующий только цифра, при это точка не может быть первым элементом;
* Если текущий элемент цифра, то следующим может быть точка (если ранее их не было, предыдущее значение не х, следующее значение цифра), либо операция, либо закрывающаяся скобка;
* Если операции +,\*,/, то следующий символ либо открывающаяся скобка, либо х, либо число. При этом данные операции не могут стоять в начале;
* Операция минус имеет те же ограничения что и остальные, но может стоять в начале;
* Открывающаяся скобка может стоять перед такой же скобкой, числом, х или минусом;
* Закрывающаяся скобка может стоять перед такой же скобкой или операцией. Не может стоять в начале. Их не может быть больше чем открывающих.

1. Идет проверка последнего элемента и количества скобок. Если количество открывающих и закрывающих скобок не совпадает, или если последний элемент открывающая скобка или знак, то возвращает False. Если все предыдущие условия выполнены, то возвращает True.

Функция возвращает значения True/False.

## Разбиение строки на лексемы (*vector <string> start)*

На вход дается исходная строка без пробелов. Заведомо известно, что ввод корректен и все данные верны. Функция работает следующим образом:

1. Если текущий элемент – число, то идем дальше по исходной строке. Если нет – проверяем, является ли следующий элемент точкой. Если да, то снова идем по всем числам после точки. После этого полученное число записываем в массив как строку.
2. Если текущий элемент х, то следующий либо 0, либо 1, либо 2. Рассмотрим на примере х0.   
   Если это первый раз, когда он встречается, то пользователю предлагается ввести его, а затем идет проверка корректности введенного значения. Если все хорошо, то теперь при каждом упоминании х0 будет заменяться заданным числом, и оно будет записываться в массив как строка.
3. Если следующий элемент: ),(,+,/,\* то записываем как строку в массив.
4. Если следующий элемент -, то, если он унарный, меняем его на ~. Если бинарный, то оставляем -. Записываем как строку в массив.

На выходе получаем вектор строк.

## Построение польской записи и вычисление результата (*double Polish)*

На вход подается вектор строк. Заведомо известно, что построение корректно и все данные верны. Функция работает следующим образом:

### Часть 1. Превращение в польскую запись.

1. Если текущий элемент число, то записываем его в результат.
2. Если текущий элемент (, то помещаем его в стек.
3. Если текущий элемент ), то извлекаем все операции до ( и помещаем их в результат. Избавляемся от (.
4. Если текущий элемент операция, берем из стека все операции, приоритет которых не ниже приоритета данной операции, записываем их в результат, а потом данную операцию записываем в стек.
5. После прохода по всему вектору, проверяем есть ли еще операции в стеке. Если есть, то записываем их в результат.

Получаем вектор строк, но выражение представлено уже в постфиксной записи.

### Часть 2. Вычисление результата.

1. Если текущий элемент число, то записываем его в стек.
2. Если текущий элемент унарный минус (~), то изменяем верхний элемент стека на противоположный по знаку.
3. Если текущий элемент + или \*, то переводим два верхних элемента из типа string в тип double, проводим операцию, а затем представляем результат в виде строки и помещаем в конец стека.
4. Если текущий элемент – или /, то делаем то же что и в пункте 3, но с учетом порядка записи. Так как элементы находятся в другом порядке, то это нужно учитывать при извлечении.

Конечный результат переводим из типа string в тип double. Это и будет результат.

На выходе получаем значение типа double.

# Описание файловой системы

1. main\_arithmetic.cpp – основная программа для запуска вычислений арифметического выражения.
2. arithmetic.h – представлены прототипы функций.
3. arithmetic.cpp – реализованы функции, необходимые для вычисления арифметических выражений.
4. stack.h – объявление и реализация класса TStack.
5. test\_main.cpp – основная программа для запуска тестов.
6. test\_ arithmetic.cpp – содержит тесты для всех объявленных функций.
7. test\_ stack.cpp – содержит тесты для класса TStack.

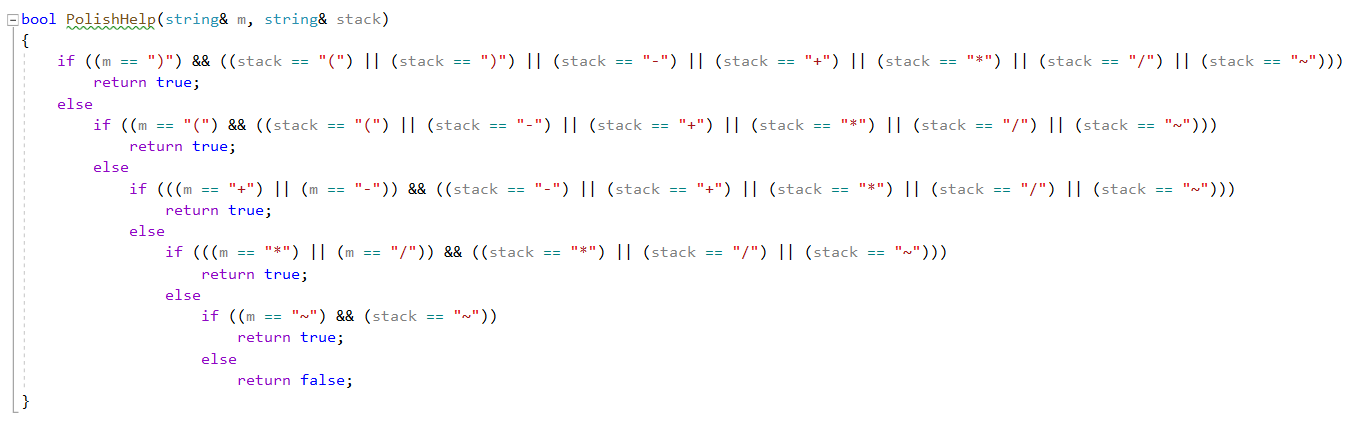
# Заключение

В данной работе было реализовано преобразование изначального выражения в польскую запись и вычисление результата на ее основе. Поддерживаются четыре бинарных операции (-,+,\*,/), а также унарный минус и скобки. При необходимости пользователь может ввести и затем задать переменные х0, х1, х2.

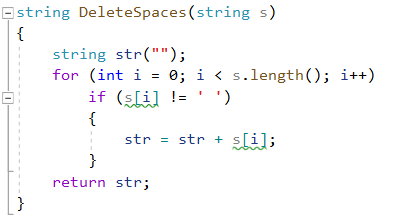
В дополнение ко всему были созданы тесты, которые проверили и подтвердили корректность работы программы.

# Приложение (код функций)

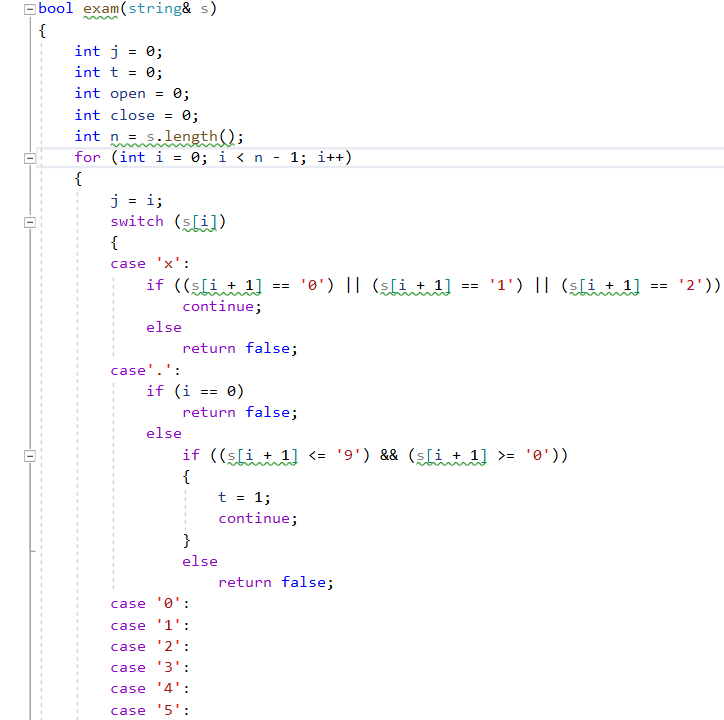
## Сравнение приоритетов двух операций

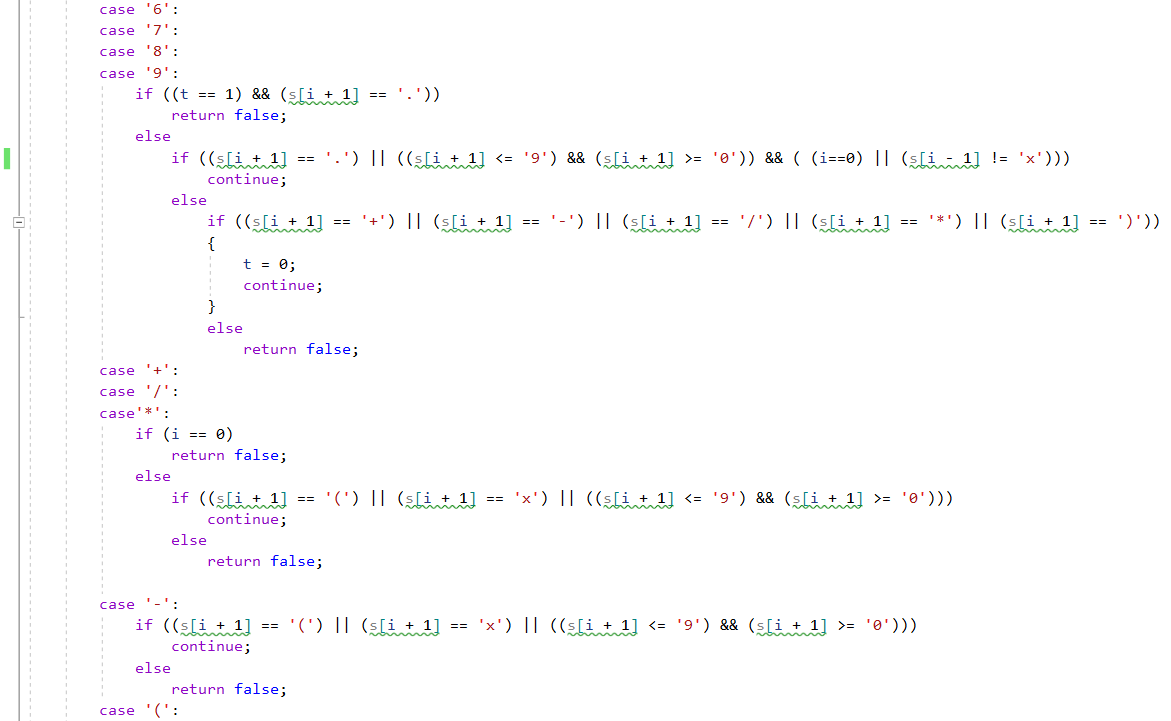


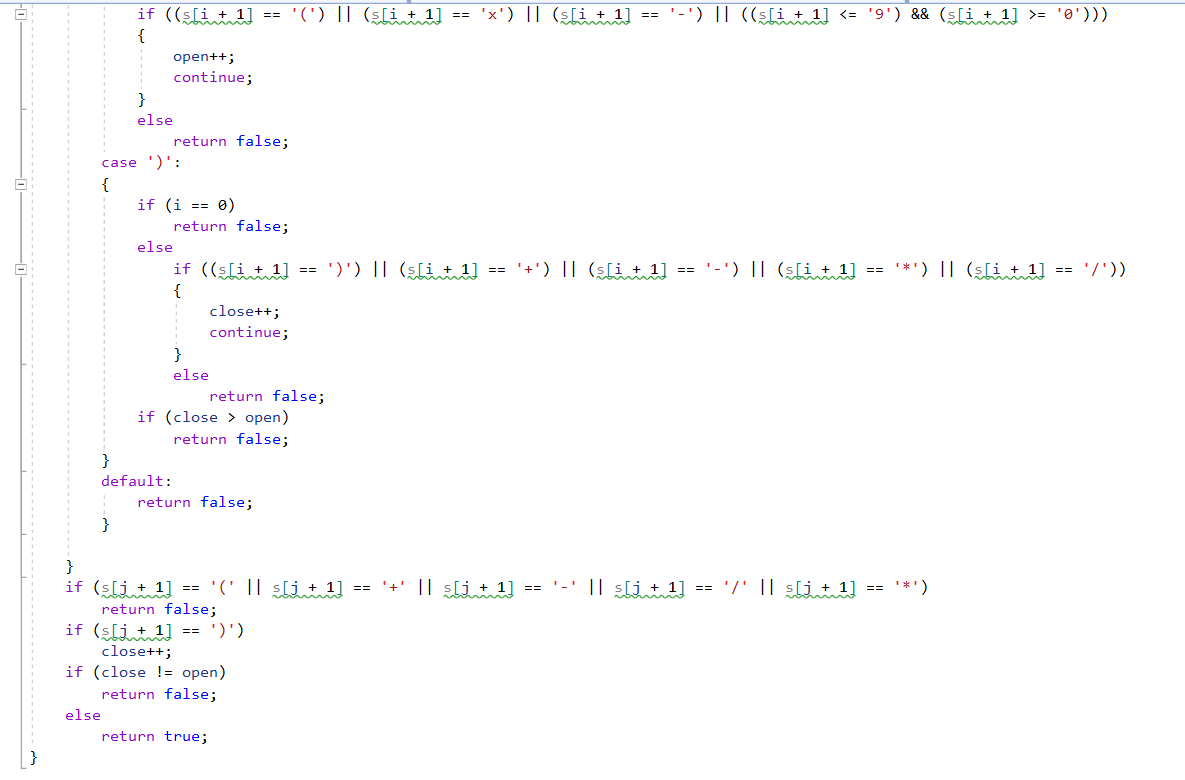
## Удаление пробелов в введенном выражении



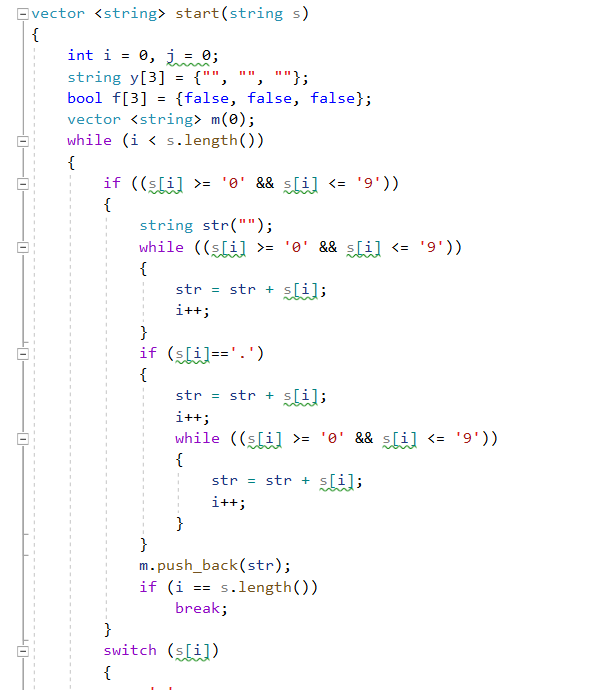
## Проверка изначального выражения на корректность ввода

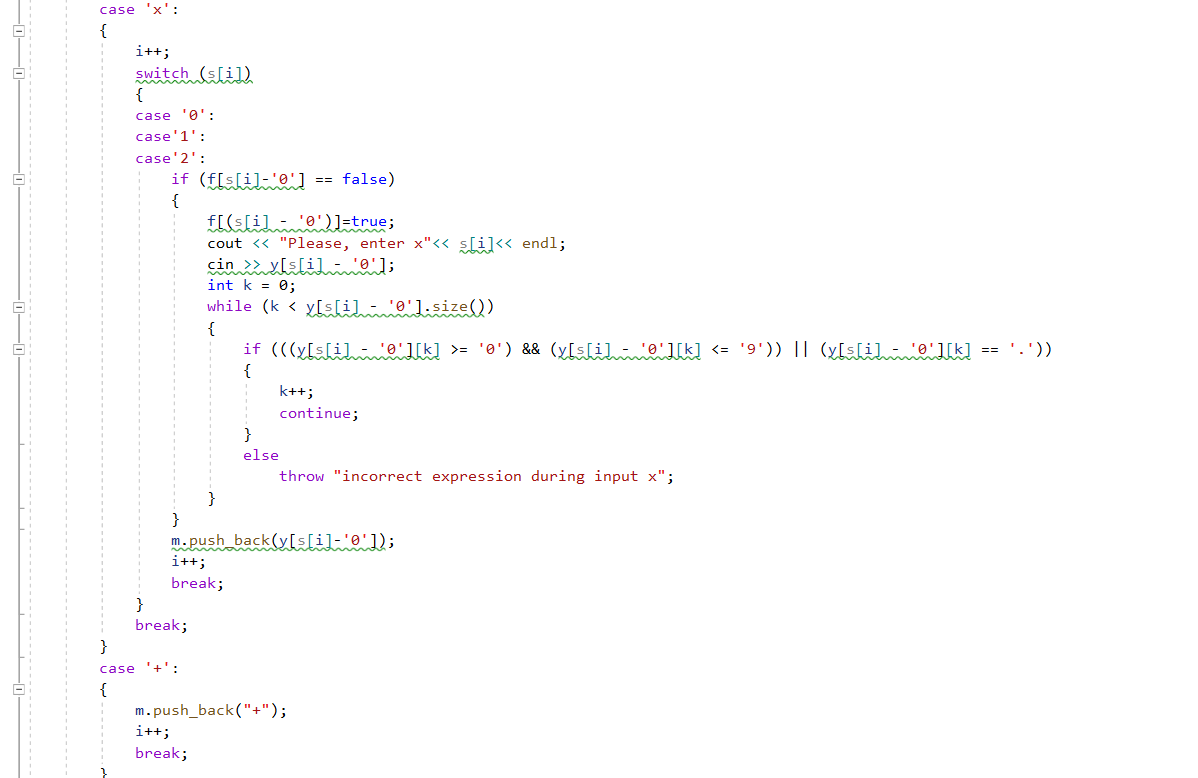


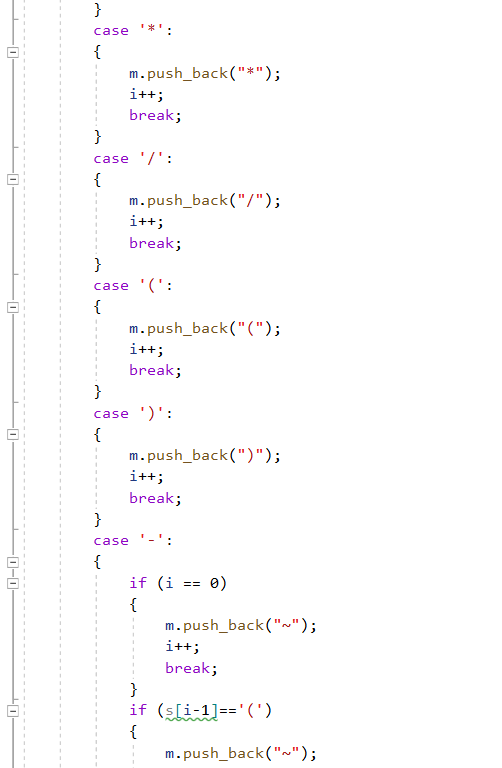


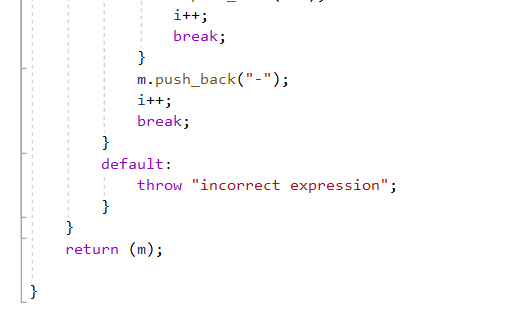


## Разбиение строки на лексемы









## Построение польской записи и вычисление результата

