ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

**Московский институт электроники и математики им. А. Н. Тихонова**

**Отчет по лабораторной работе №4 (№2)**

**по предмету «Языки Программирования»**

Выполнил: студент группы СКБ221

Нугманов М.И.

Москва 2022 г.

**Описание работы программы**

Программа реализует функции калькулятора, способного распознавать два вида записи математического выражения: в Инфиксной и Обратной Польской нотациях.

Пользователю необходимо ввести название exe файла, далее флаг, обозначающий нотацию (--**forward**, **--reverse**) или вывод инструкции (--**help**).

Если пользователь введет --**help**, то будет выведена инструкция, в которой он может ознакомиться с функционалом программы, а также с правилами ввода тех или иных команд.

Если пользователь ввел флаг, обозначающий нотацию, далее необходимо ввести флаг, обозначающий использование чтения конкретным способом (--**file** или --**console**). Если введен флаг --**console**, то чтение будет осуществляться из консоли, если --**file –** то с файла.

В случае если пользователь ввел --**file**, то далее необходимо ввести путь к файлу, с которого будет производиться чтение математических выражений.

Математическое выражение, вводимое пользователем в консоль или считываемое из файла, должно быть написано по правилам выбранной нотации.

В случае неправильного ввода флага или названия программы будет выведена соответствующая ошибка и предложение ввести флаг --**help** для вывода инструкции.

**Описание работы алгоритма**

**Основные функции:**

1. **main()**

Программа ждет от пользователя ввод названия exe файла, флаг нотации или вывода инструкции, флаг способа чтения, а также путь к файлу, если выбран способ чтения из файла.

Если введено только название файла, программа выведет следующую ошибку: **###\_ERROR: flag not found\_###**

Если введено название файла и один флаг, то этим флагом может являться только --**help**, поскольку он не требует дальнейших аргументов. Иначе будет выведена ошибка: **###\_ERROR: flag entered incorrectly\_###**

Если введено название файла и два флага, то первый должен означать нотацию, а второй – чтение с консоли, поскольку для чтения с файла необходимо дополнительно ввести путь к файлу. В случае выбора –forward и –console программа попросит ввести математическое выражения в Инфиксной нотации, а в случае выбора –reverse и –console – в Обратной Польской нотации. Далее запустятся соответствующие функции InfNotation() и PolNotation(). Иначе будет выведены следующие ошибки: **###\_ERROR: flag entered incorrectly\_###** или **###\_ERROR: file name not entered\_###** в зависимости от ситуации.

Если введено название файла и три флага, то первый должен означать нотацию, второй – чтение с файла, третий – путь к необходимому файлу. Важно написать верный путь к файлу, а также именно ту нотацию, в которой написаны математические примеры в файле. Иначе будет выведена следующая ошибка: **###\_ERROR: flag entered incorrectly\_###**.

1. **InfNotation()**

Функция предназначена для вычисления математического выражения, записанного в Инфиксной нотации. Программа считывает каждый отдельный символ с входного потока, который был записан в массив **stream**, далее этот символ проходит проверку на то, является ли он цифрой (**IsDigit()**). Если да, то он преобразовывается в вещественный тип и записывается в соответствующую переменную. Далее идет проверка на то, нет ли последующих цифр с целью выявить многозначное число. Если дальше идут цифры, записанное ранее число умножается на 10 и прибавляется следующий разряд. Когда подряд идущие цифры закончатся, полученное число записывается в массив чисел **numbers**, над которыми будет производиться вычисление. Если текущий символ оказался не цифрой, далее следует проверка на то, является ли этот символ Операцией вычисления (**IsOperation()**). Если да, то нам следует произвести вычисление с уже имеющимися числами, если у нас имеются ранее записанные операции и если приоритет текущей операции меньше либо равен приоритету последней записанной операции. Для этого мы достаем из массива **numbers** два последних записанных числа, а также из массива **operations** последнюю записанную операцию и производим вычисление с помощью функции **Calculate()**. Результат вычисления будет записан, как крайний элемент массива **numbers**. После выхода из цикла нужно записать текущую операцию в массив **operations**. Если символ не является цифрой или операцией, то идет проверка на открывающуюся скобку, и в случае истины символ будет записан в массив с операциями. Если символ не прошел предыдущие проверки, тогда идет проверка на закрывающуюся скобку, и если символ является таковым, запускается цикл ,в котором из массивов будут доставаться числа и операции, будет производиться вычисление и результат записываться крайним элементов массива **numbers** – и все это до тех пор, пока в массиве **operations** мы не встретим открывающуюся скобку. Когда встретится открывающаяся скобка, цикл завершится и снова будет происходить считывание символа с входного потока. Когда чтение символов завершится, текущий цикл прервется и начнется следующий, в котором будут производиться оставшиеся вычисления: будут доставаться из массива **numbers** последние два числа, из массива **operations** будет доставаться последняя операция, а результат вычисления будет записан крайним элементом массива **numbers**. Когда массива с операциями станет пустым, первым элементом массива **numbers** будет являться результат вычисления всего математического выражения, он выведется на экран и выполнение функции завершится.

1. **PolNotation()**

Функция предназначена для вычисления математического выражения, написанного в Обратной Польской нотации. Программа считывает каждый отдельный символ с входного потока, который был записан в соответствующий массив **stream**. Далее идет проверка на то, является ли текущий символ цифрой (**IsDigit()**). Если да, символ преобразуется в вещественный тип и записывается в переменную. Далее идет проверка на то, являются ли последующие символы цифрами, так как возможен случай, что пользователь ввел многозначное число. Если это так, то запускается цикл, считывающий подряд идущие цифры, и записывающий с конца к уже имеющемуся числу. Далее после завершения цикла полученное число записывается в массив **numbers** и считывается следующий массив. Если текущий символ оказался не цифрой, идет проверка на то, является ли он символом операции (**IsOperation()**). Если да, то данный символ сразу берется как текущее действие над числами, а также из массива **numbers** берутся два последних записанных ранее числа и производится вычисление с помощью функции **Calculate()**, и результат перезаписывает крайний элемент массива **numbers**. После завершения считывания символов с входного потока результат вычислений находится в первой ячейке массива **numbers**, он выводится на консоль и выполнение функции завершается.

**Вспомогательные функции:**

1. **Help()**

Вызывается с помощью соответствующего флага **–help**, выводит на консоль инструкцию по использованию данной программы.

1. **IsDigit()**

Получает на вход символ и проверяет путем сравнения по позиции в таблице кодировки, является ли данный символ цифрой. Если да, возвращает 1, если нет – 0.

1. **IsOperation()**

Получает на вход символ и проверяет путем сравнения, является ли данный символ математической операцией над числами. Если да, возвращает 1, если нет – 0.

1. **GetPriority()**

Получает на вход символ и проверяет путем конструкции **switch-case**, какая математическая операция была получена на вход, и в зависимости от этого возвращает приоритет данной операции. Нужно это для того, чтобы верно вычислить математическое выражение в Инфиксной нотации, где ход действий зависит от приоритета той или иной операции.

1. **Calculate()**

Получает на вход два числа вещественного типа и символ операции, и проверяет путем конструкции **switch-case**, какую математическую операцию требуется выполнить. В зависимости от символа возвращается соответствующий результат вычисления. Дополнительно для операции деления идет проверка на то, является ли делитель нулем, и возвращает ошибку в случае истины.

1. **ReadingFromConsole()**

С помощью метода **getline()** осуществляется запись входного потока в символьный массив **stream**, с которого в последующем будут считываться символы и производиться вычисления.

1. **ReadingFromFile()**

На вход функция получает путь к файлу с математическими выражениями, а также флаг, обозначающий соответствующую нотацию, в которой были записаны математические выражения в это файле. Создается объект класса **ifstream**, который относится к файлу. Далее идет проверка на корректность ввода пути к файлу с помощью метода **is\_open()**. Если файл удалось открыть, запускается цикл, который будет работать до тех пор, пока не будет достигнут конец файла, соответствующая проверка осуществляется с помощью метода **eof()**. Далее считывается строка с файла с помощью метода **getline()** и записывается в соответствующий массив **stream**. Далее, в зависимости от введенного флага вызывается функция **InfNotation()** или **PolNotation()**. Результаты нумеруются, так что в случае, если выражений в файле несколько, это крайне удобно.

**Вывод**

В ходе выполнения лабораторной работы я узнал о реализации обработки ошибок в случае некорректного ввода пользователем данных; познакомился с функционалом заголовочных файлом <cstring>, <fstream>, <stdlib.h>; попрактивался в написании make-файла, в работе с Git, а также лучше узнал о реализации чтения данных с файлов.