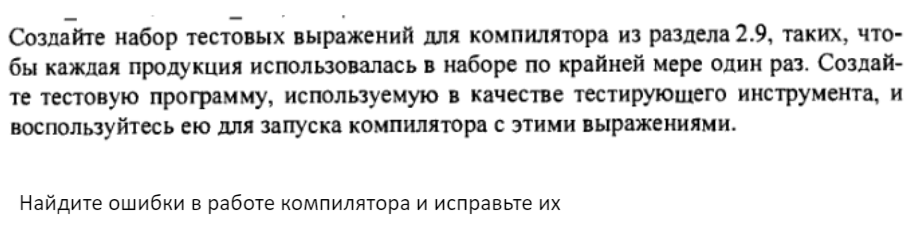
Жуковский Павел, 12 группа

Вариант 1

**Задание**

**Условие:**

****

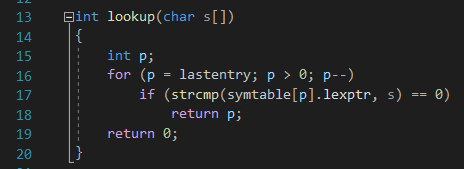
**Ход выполнения**

В данном компиляторе мною были обнаружены 3 ошибки, из-за которых входные строки обрабатывались некорректно.

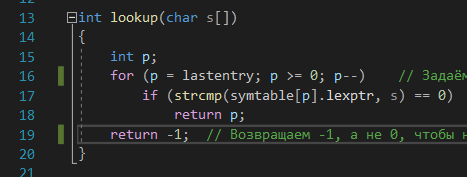
Для удобства я переименовал расширения исходных файлов с \*.cpp на \*.c, ведь они написаны на языке C, а не C++ (почему бы и нет).

В этом отчёте я разберу, где именно были допущены ошибки, а также, как исправить эти ошибки, а в конце предоставлю результаты тестирования компилятора на своих строках.

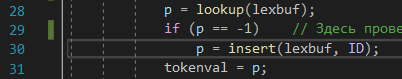
**Первая ошибка:** функция **lookup()**, файл **symbol.c**



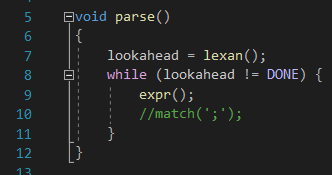
Если посмотреть на эту функцию, то можно заметить, что внутри цикла никогда не проверяется нулевой индекс (цикл доходит до нуля, но сам нуль не смотрит), поэтому самый первый элемент вовсе не рассматривается. Соответственно пришлось исправить ограничение в цикле с «p > 0» на «p >= 0», и тогда первый символ станет просматриваться. Однако возникает ещё одна проблема: при вставке символов в таблицу, **DIV** попадал на этот самый нулевой индекс (т.е. эта операция по сути даже не распознавалась), который также использовался как признак отсутствия записи. Чтобы исправить это, нужно вернуть любое отрицательное число (чтобы оно точно не совпало с каким-нибудь p), поэтому вернём **-1**. В итоге, функция стала выглядеть таким образом:



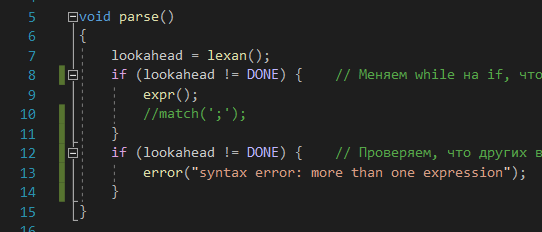
Также нужно соответственно исправить **0** на **-1** ещё в файле **lexer.c** в функции **lexan()**, чтобы программа корректно понимала признак отсутствия записи:



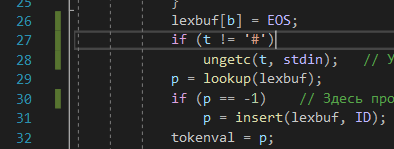
**Вторая ошибка:** функция **parse()**, файл **parser.c**



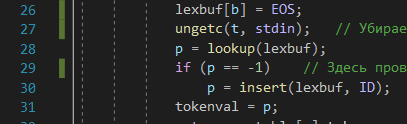
В данной функции можно заметить, что программа внутри цикла выполняет абсолютно все команды, которые встречает, пока не встретит символ «**#**». Это, вообще говоря, некорректно, т.к. наша программа работает только с одним арифметическим выражением. Соответственно, нужно сделать 2 вещи: 1) заменить цикл **while** на оператор **if**, чтобы обработать лишь одно арифметическое выражение; 2) по-хорошему надо также сделать проверку, что больше арифметических выражений нету, т.е. что нету синтаксической ошибки и что после первого выражения идёт непосредственно символ «**#**». В итоге функция станет выглядеть таким образом:



**Третья ошибка**: функция **lexan()**, файл **lexer.c**



Суть третьей ошибки заключается в том, что внутри функции **lexan()** почему-то символ «**#**» не добавлялся во входной поток (функция ungetc(t, stdin) добавляет символ t в поток stdin). На самом деле, для корректной работы символ «**#**» также должен добавляться в поток, поэтому эту проверку вовсе нужно убрать:



Таким образом, мы исправили все ошибки компилятора. Теперь можно его протестировать на строках. Тестировать программу мы будем на разные арифметические выражения, более того, мы также должны протестировать ситуации, когда на вход подаётся некорректное выражение. Для этого я написал 10 тестов.

1) **Тест №1**:

Вход: **3-5+7#**

Ожидаемый результат:

3

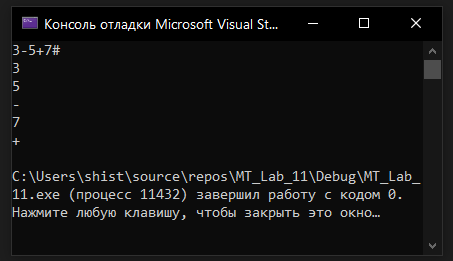
5

-

7

+

**Результат работы программы:**

****

**Результат теста: пройден**

2) **Тест №2**:

Вход: **4+5/7 #**

Ожидаемый результат:

4

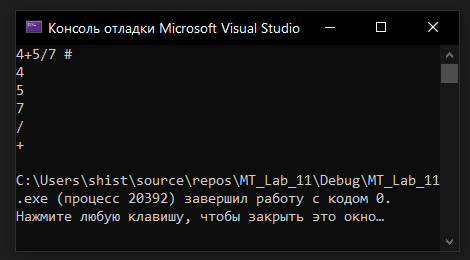
5

7

/

+

**Результат работы программы:**



**Результат теста: пройден**

3) **Тест №3**:

Вход: **2\*3+(x div y) mod z#**

Ожидаемый результат:

2

3

\*

x

y

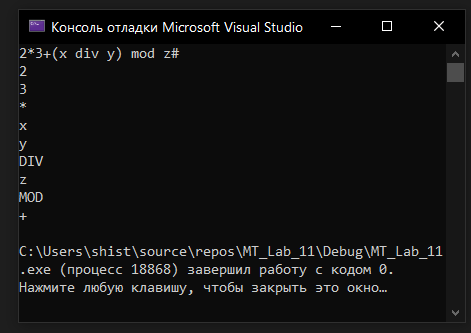
DIV

z

MOD

+

**Результат работы программы:**



**Результат теста: пройден**

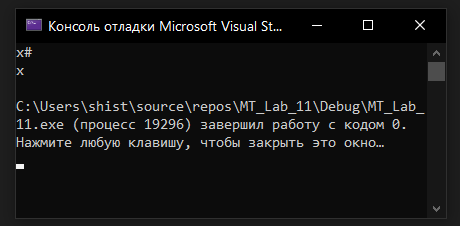
4) **Тест №4**:

Вход: **x#**

Ожидаемый результат:

x

**Результат работы программы:**



**Результат теста: пройден**

5) **Тест №5**:

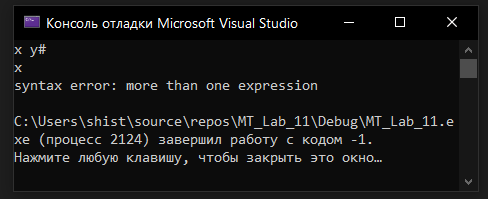
Вход: **x y#**

Ожидаемый результат:

x

syntax error: more than one expression

**Результат работы программы:**



**Результат теста: пройден**

6) **Тест №6**:

Вход: **3 mod 4 -#**

Ожидаемый результат:

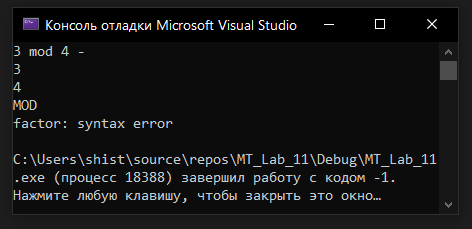
3

4

MOD

factor: syntax error

**Результат работы программы:**



**Результат теста: пройден**

7) **Тест №7**:

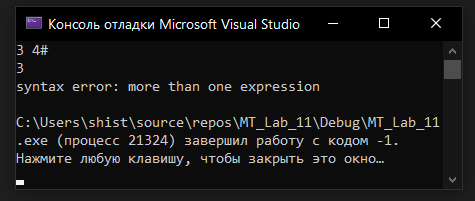
Вход: **3 4#**

Ожидаемый результат:

3

syntax error: more than one expression

**Результат работы программы:**



**Результат теста: пройден**

8) **Тест №8**:

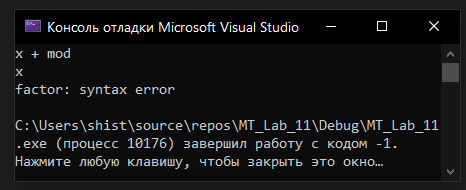
Вход: **x + mod**

Ожидаемый результат:

x

factor: syntax error

**Результат работы программы:**



**Результат теста: пройден**

9) **Тест №9**:

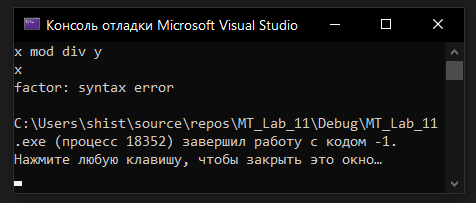
Вход: **x mod div y**

Ожидаемый результат:

x

factor: syntax error

**Результат работы программы:**



**Результат теста: пройден**

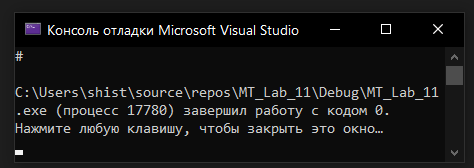
10) **Тест №10**:

Вход: **#**

Ожидаемый результат:

(ничего не должно вывестись)

**Результат работы программы:**



**Результат теста: пройден**

Таким образом, исправленная программа прошла 10 моих тестов и некорректных обработок строк обнаружено не было.

Все обновлённые исходные файлы программы я помещу в zip-архив и приложу в письме вместе с этим отчётом.