Отчет по лабораторной работе №1

“Формальные грамматики и методы трансляции”

Выполнил: Шестаков Андрей, ПМИ-1

Оглавление

[Задача 3](#_Toc84201240)

[Анализ 4](#_Toc84201241)

[Проектирование 5](#_Toc84201242)

[Разработка и тестирование 7](#_Toc84201243)

# Задача

Спроектировать и запрограммировать модуль ввода/вывода для выдачи токенов из файла. Изначальный текст разбить на лексемы и в зависимости от того что получили, вернуть токены. Также необходимо выводить информацию о лексических ошибках.

# Анализ

Для выполнения данной лабораторной работы, я ознакомился со следующими книгами и статьями:

1) Разработка паскаль-компилятора Л.Залогова.

2) Компиляторы. Принципы, технологии и инструментарий. Альфред В. Ахо,

Моника С. Лам, Рави Сети, Джеффри Д. Ульман.

3) <http://citforum.ru/programming/theory/serebryakov/1.shtml>

# Проектирование

Для решения исходного задания было принято использовать следующий классы:

Class CToken; - Класс токена, где хранится информация.

Class CIO; - Модуль ввода/вывода

Class CVariant; - Для констант int, double, string.

Class Error; - Для ошибок

Рассмотрим каждый класс по отдельности:

1) CToken

Внутри будут следующие поля и методы:

ПОЛЯ

-Тип токена

-Тип операции

-Описание операции

-Значение константы

-Значение лексемы

МЕТОДЫ

-Инициализация описания операций

-Получение типа токена

-Вывод токена

2) CVariant

Абстрактный класс, который содержит виртуальный метод получение типа класса, так как от CVariant будет 3 производных класса:

-CIntVariant

- CDoubleVariant

- CStringVariant

Для каждого типа констант соответственно:

Производные классы будут содержать следующие поля и методы

ПОЛЯ

-Значение константы

МЕТОДЫ

-Получить значение

-Получить тип класса

3) CIO

Внутри будут следующие поля и методы:

ПОЛЯ

-Список ошибок

-Поток для чтения файла

-Ключевые слова языка

-Получить символ операции

-Координаты в программе

-Строка программы

МЕТОДЫ

-Получение следующего символа

-Получение следующего токена

-Проверка константы на переполнение

-Изменение координат токена

-Вывод ошибок

3)Error

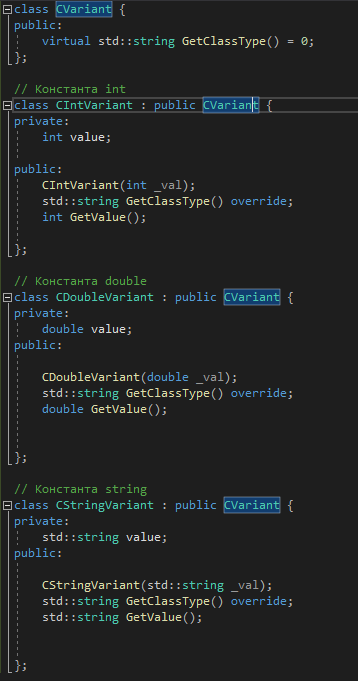
Абстрактный класс, который содержит виртуальный метод получение типа ошибки, также позицию токена, код ошибки, расшифровка ошибок. От него будут наследоваться Лексические ошибки, с

Синтаксические и т.д.

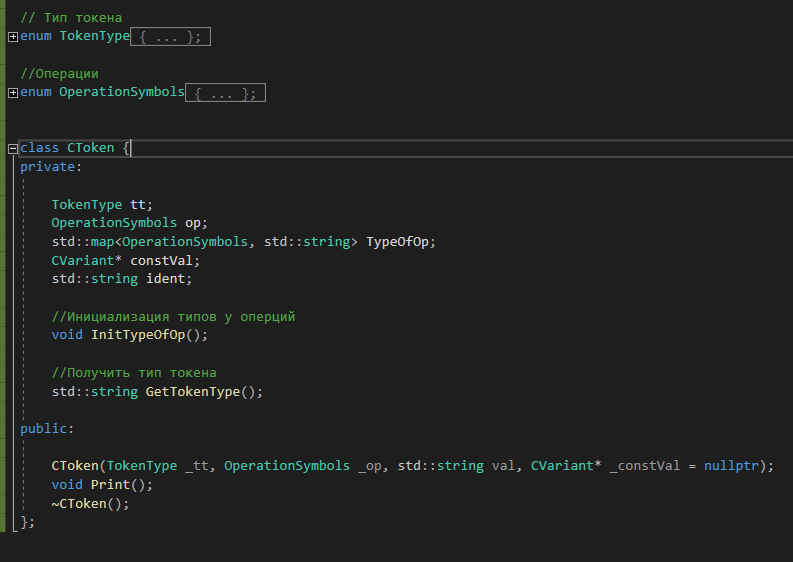
# Разработка и тестирование

Для начала я решил в заголовочных файлах расписать спроектированные классы, пока без реализации:

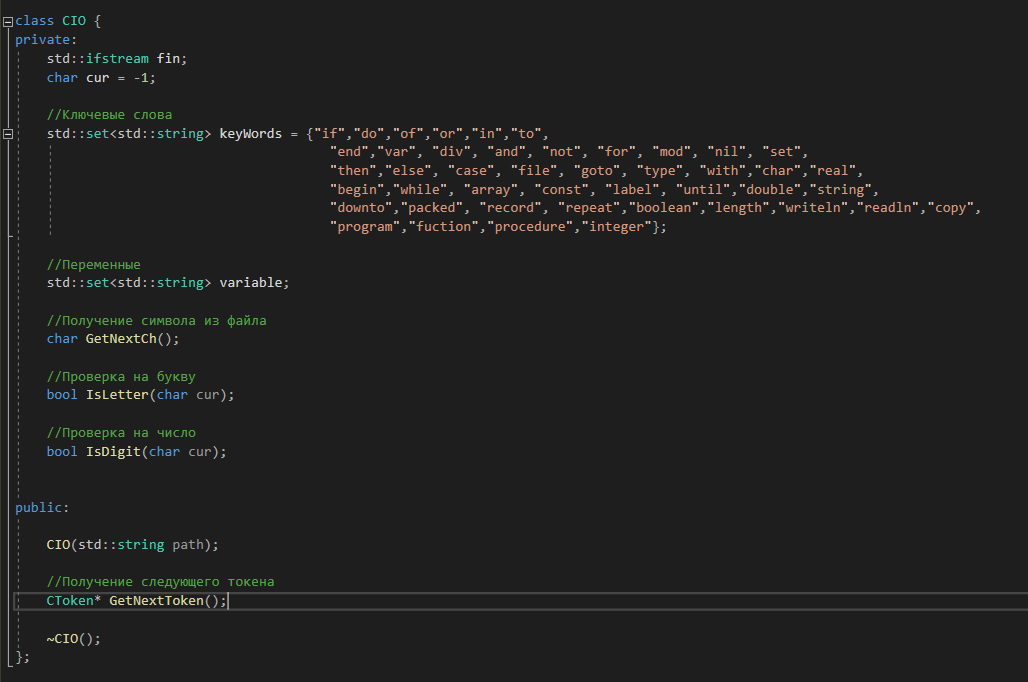
1)CVariant



2)CToken



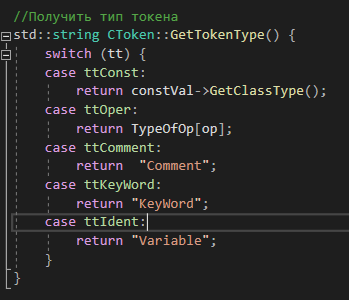
3)CIO



Дальше я постепенно приступил к реализации всех методов. Для начала, я написал несложные методы, которые в дальнейшем будут применяться в других более сложных методах (с точки зрения реализации).

В такие первые методы попали:

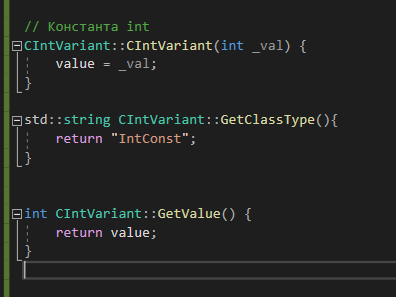
GetTokenType()



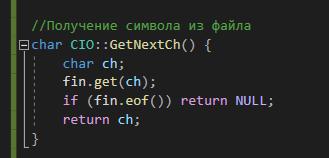
Инициализация типов у операции и конструктор



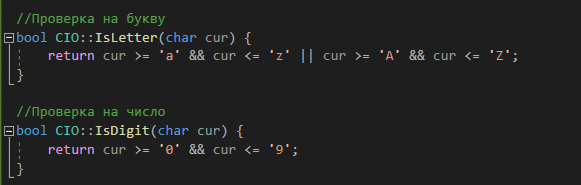
Методы производных классов CVariant(пример для int)



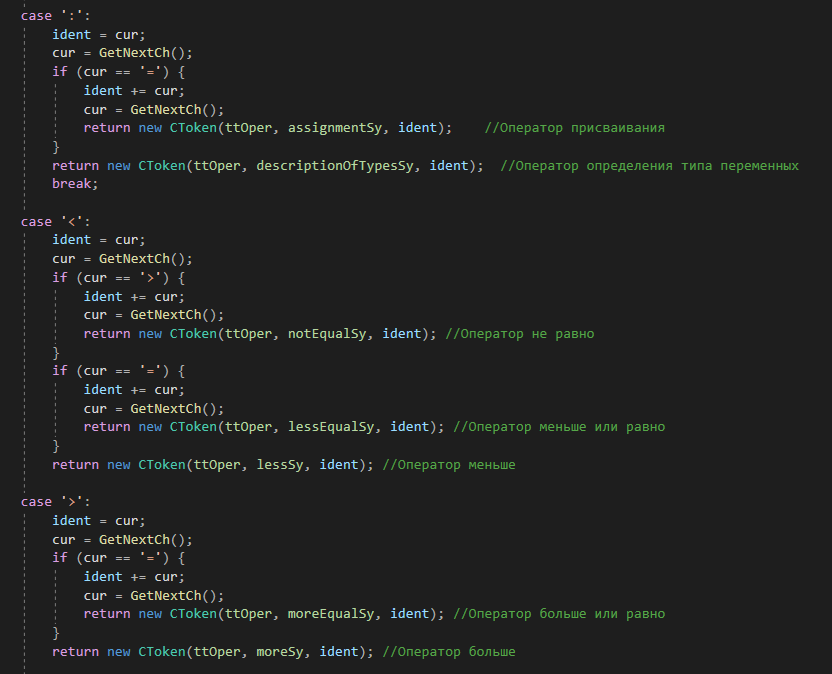
Получение символов из файла



И некоторые проверки



Дальше для тестирования я решил начать реализовывать метод GetNextToken() и написал парсер для некоторых символов.

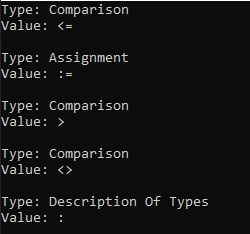


Протестируем:

Содержимое файла

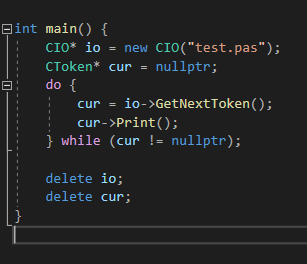


Результат:



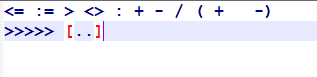
Вроде походит на правду.

Код основной программы

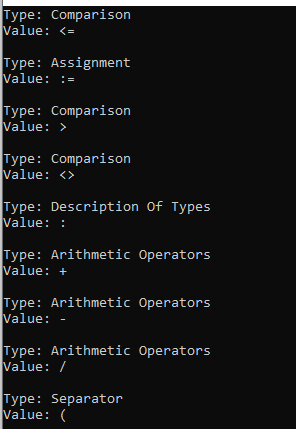


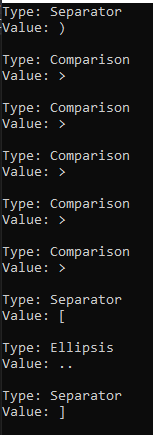
Дальше были дописаны парсеры для ещё некоторых символов

Содержимой файлика



Результат





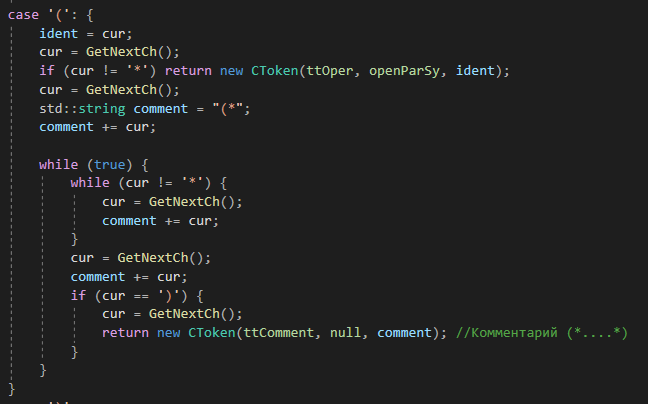
Также результат ожидаем

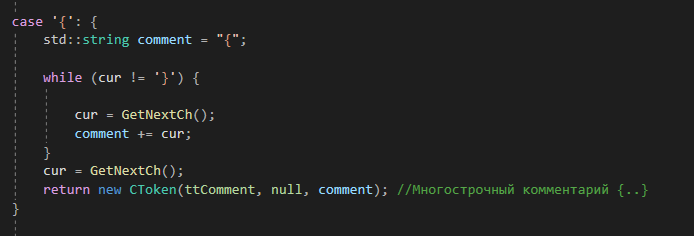
После этого я начал реализовывать комментарии

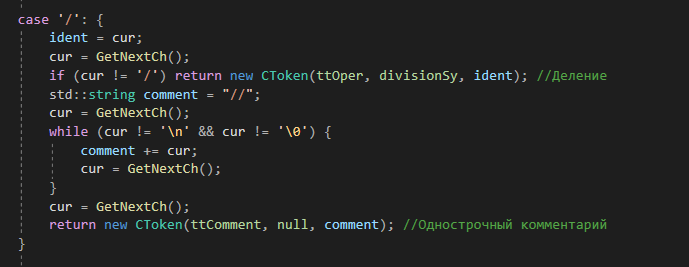
// -однострочные

{..} – многострочные

(\*....\*)

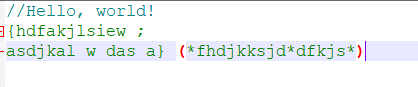




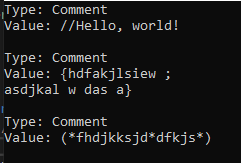


Тестируем

Файлик

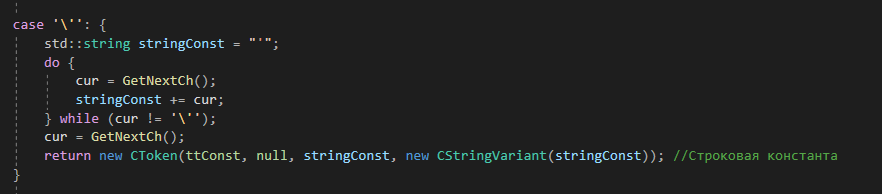


Результат

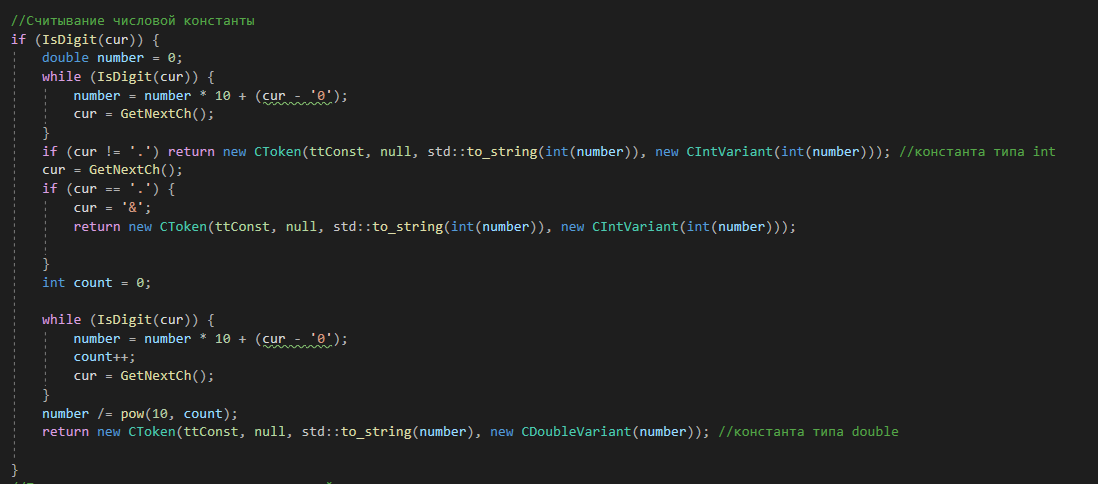


Дальше были разработаны считывание констант

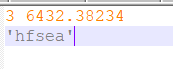
Строковая константа

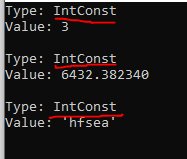


Числовая константа



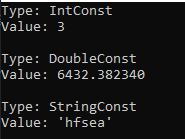
Тесты





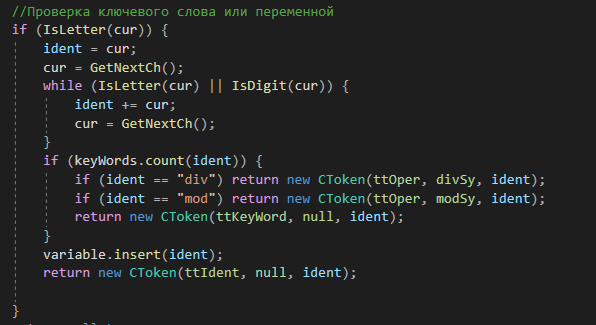
Как мы видим, везде определилась, как интовая константа. В итоге криво реализовал наследование.

Правим и проверяем

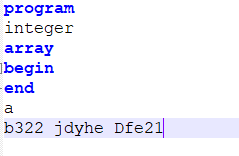


Теперь всё хорошо.

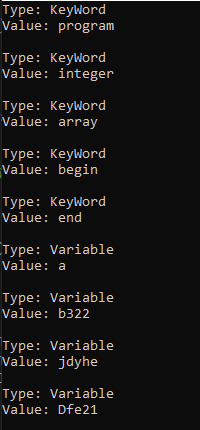
Дальше уже идёт проверка ключевых слов и переменных



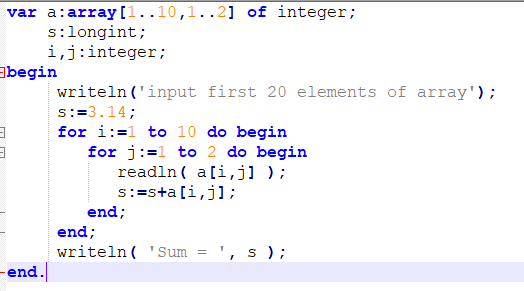
Тестируем



Результат



Теперь проведем комплексное тестирование.



Поскольку результат большой, выведу его текстом

Type: KeyWord

Value: var

Type: Variable

Value: a

Type: Description Of Types

Value: :

Type: KeyWord

Value: array

Type: Separator

Value: [

Type: IntConst

Value: 1

Type: Ellipsis

Value: ..

Type: IntConst

Value: 10

Type: Separator

Value: ,

Type: IntConst

Value: 1

Type: Ellipsis

Value: ..

Type: IntConst

Value: 2

Type: Separator

Value: ]

Type: KeyWord

Value: of

Type: KeyWord

Value: integer

Type: Separator

Value: ;

Type: Variable

Value: s

Type: Description Of Types

Value: :

Type: KeyWord

Value: longint

Type: Separator

Value: ;

Type: Variable

Value: i

Type: Separator

Value: ,

Type: Variable

Value: j

Type: Description Of Types

Value: :

Type: KeyWord

Value: integer

Type: Separator

Value: ;

Type: KeyWord

Value: begin

Type: KeyWord

Value: writeln

Type: Separator

Value: (

Type: StringConst

Value: 'input first 20 elements of array'

Type: Separator

Value: )

Type: Separator

Value: ;

Type: Variable

Value: s

Type: Assignment

Value: :=

Type: DoubleConst

Value: 3.140000

Type: Separator

Value: ;

Type: KeyWord

Value: for

Type: Variable

Value: i

Type: Assignment

Value: :=

Type: IntConst

Value: 1

Type: KeyWord

Value: to

Type: IntConst

Value: 10

Type: KeyWord

Value: do

Type: KeyWord

Value: begin

Type: KeyWord

Value: for

Type: Variable

Value: j

Type: Assignment

Value: :=

Type: IntConst

Value: 1

Type: KeyWord

Value: to

Type: IntConst

Value: 2

Type: KeyWord

Value: do

Type: KeyWord

Value: begin

Type: KeyWord

Value: readln

Type: Separator

Value: (

Type: Variable

Value: a

Type: Separator

Value: [

Type: Variable

Value: i

Type: Separator

Value: ,

Type: Variable

Value: j

Type: Separator

Value: ]

Type: Separator

Value: )

Type: Separator

Value: ;

Type: Variable

Value: s

Type: Assignment

Value: :=

Type: Variable

Value: s

Type: Arithmetic Operators

Value: +

Type: Variable

Value: a

Type: Separator

Value: [

Type: Variable

Value: i

Type: Separator

Value: ,

Type: Variable

Value: j

Type: Separator

Value: ]

Type: Separator

Value: ;

Type: KeyWord

Value: end

Type: Separator

Value: ;

Type: KeyWord

Value: end

Type: Separator

Value: ;

Type: KeyWord

Value: writeln

Type: Separator

Value: (

Type: StringConst

Value: 'Sum = '

Type: Separator

Value: ,

Type: Variable

Value: s

Type: Separator

Value: )

Type: Separator

Value: ;

Type: KeyWord

Value: end

Type: Separator

Value: .

Для начала мною были исправлены некоторые ошибке в получении следующего токена.

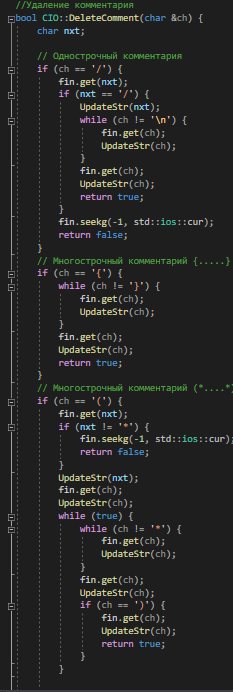
Для начала я оставил только 3 типа токена:

1)Операция

2)Идентификатор

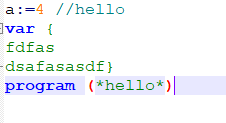
3)Константа

Также в получении следующего символа для пропускаю все комментарии.

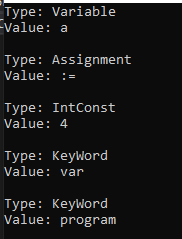


Тестирование

Исходный файл



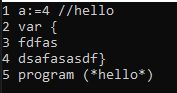
Результат программы:



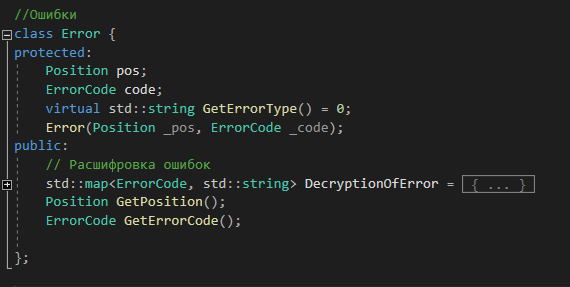
Далее я решил выводить строку программы, а затем все ошибки в ней.

Для начала просто выведем всю программу.

Получаем что-то такое



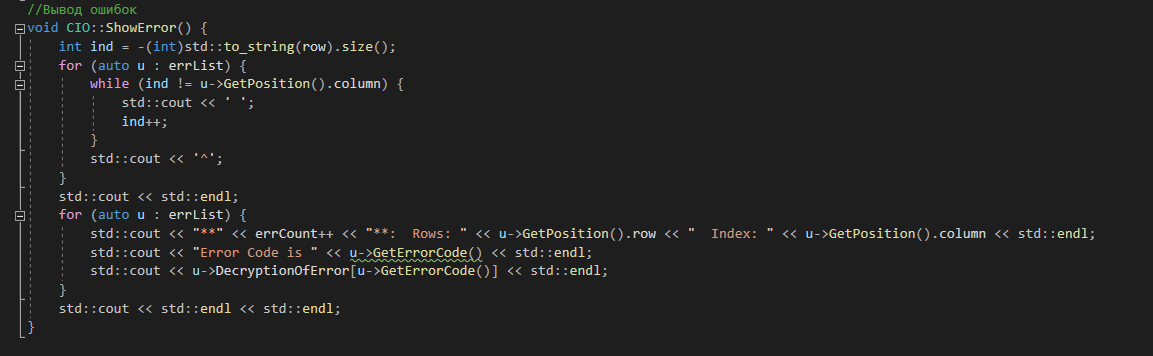
Теперь опишем класс ошибок



Теперь в момент получения следующего токена можно проверять корректность лексемы, начнём с переполнения типа

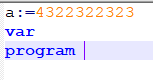


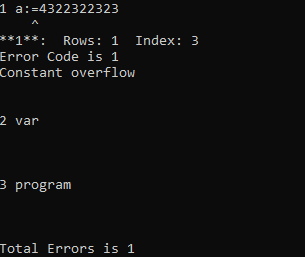
Дальше реализуем вывод ошибок



И протестируем

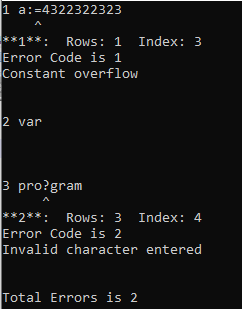
Исходный файл:





В результате получили ошибку о переполнении константы

Также в получении следующего токена я прохожу все проверки и символ никуда не подходит, значит некорректный символ.



Также проверка на неправильное название переменной

