|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ | | | | | | | | | | |
|  | | | |  | | |  | | | |
| ФГБОУ ВПО «Пермский государственный национальный исследовательский университет» | | | | | | | | | | |
|  | |  | | | | | | |  | |
|  | | ОТЧЕТ  по дисциплине «Формальные грамматики и методы трансляции» | | | | | | |  | |
|  | | |  | | | | |  | | |
|  | Работу выполнила  студентка гр. ПМИ-1,2  Калинина М.О. \_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2022 | | | |  | Проверил  ассистент кафедры МОВС  Пономарёв Ф.А. \_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2022 | | | |  |
|  | | Пермь 2022 | | | | | | |  | |

**Оглавление**

[Модуль ввода-вывода 3](#_Toc120817192)

[Описание 3](#_Toc120817193)

[Проектирование 3](#_Toc120817194)

[Реализация 3](#_Toc120817195)

[Тестирование 9](#_Toc120817196)

[Лексический анализатор 12](#_Toc120817197)

[Описание 12](#_Toc120817198)

[Проектирование 12](#_Toc120817199)

[Реализация программы 13](#_Toc120817200)

[Тестирование 17](#_Toc120817201)

[Синтаксический анализатор 21](#_Toc120817202)

[Описание 21](#_Toc120817203)

[Проектирование 21](#_Toc120817204)

[Реализация 23](#_Toc120817205)

[Тестирование 35](#_Toc120817206)

[Семантический анализатор 40](#_Toc120817207)

[Описание 40](#_Toc120817208)

[Проектирование 40](#_Toc120817209)

[Реализация 41](#_Toc120817210)

[Тестирование 46](#_Toc120817211)

# Модуль ввода-вывода

## Описание

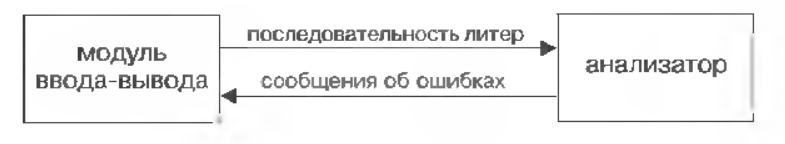
Модуль ввода-вывода считывает последовательность литер исходной программы с внешнего устройства и передает их анализатору.

## Проектирование

На данном этапе нам необходимо входной файл разбить на лексемы, в соответствии с синтаксисом языка, а именно его ключевыми словами и специальными символами, определив для каждой лексемы её тип и сохраняя их в определённый буфер.

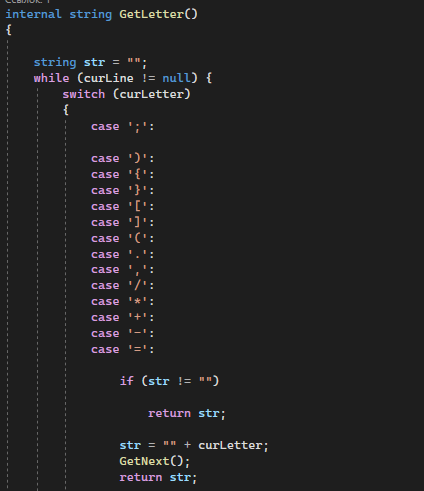
Концепция данного модуля заключается на обработке отдельных групп символов. Набор всех допустимых символов, был разбит на 3 подмножества.

1. Множество букв в нижнем и вернем регистре и нижнее подчёркивание
2. Множество цифр
3. Множество символов

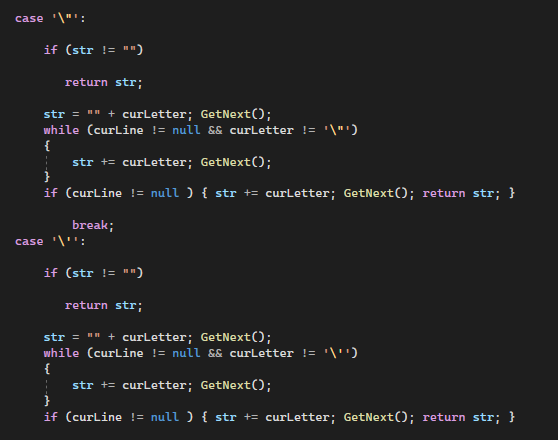


## Реализация

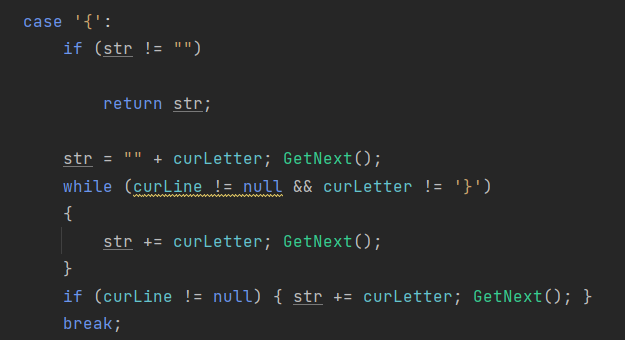
В первую очередь модуль ввода-вывода распознает операторы, которые могут быть разделителями.



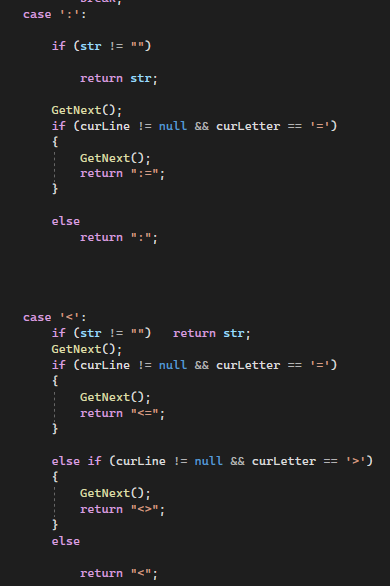
Всё что находится в кавычках он определяет, как одну лексему.

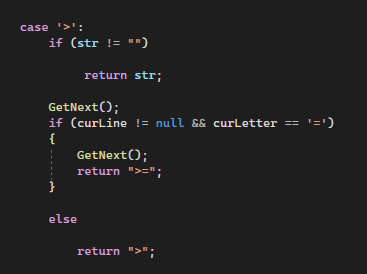


Комментарии он просто пропускает.

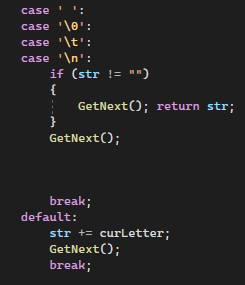


Также реализована проверка символов наперед для неодносимвольных операторов.

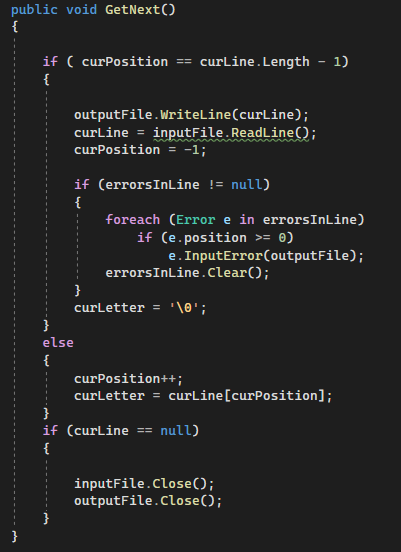




При встрече символов переноса строки, пробела, табуляции модуль вводы-вывода их просто пропускает. Если встретился другой символ, он прибавляет его в строку пока не встретился известный нам оператор.

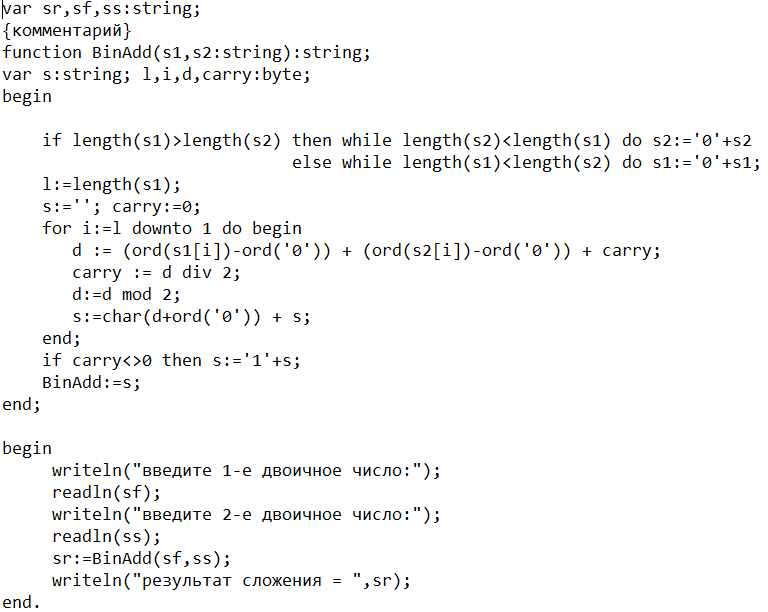


На модуль ввода-вывода положен также вывод ошибок каждой строки программы.

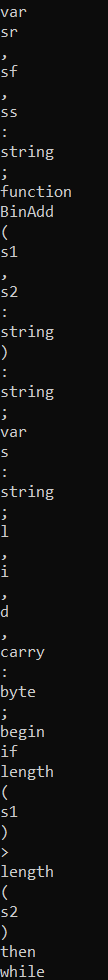


## Тестирование

Для данной программы



был выведен данный результат:



Был вставлен неполный скриншот, но из этой части уже можно понять, что модуль работает верно.

# Лексический анализатор

## Описание

Лексический анализатор формирует символы исходной программы и строит их внутреннее представление. Кроме того, сканер распознает и исключает комментарии, которые не нужны для дальнейшей трансляции.

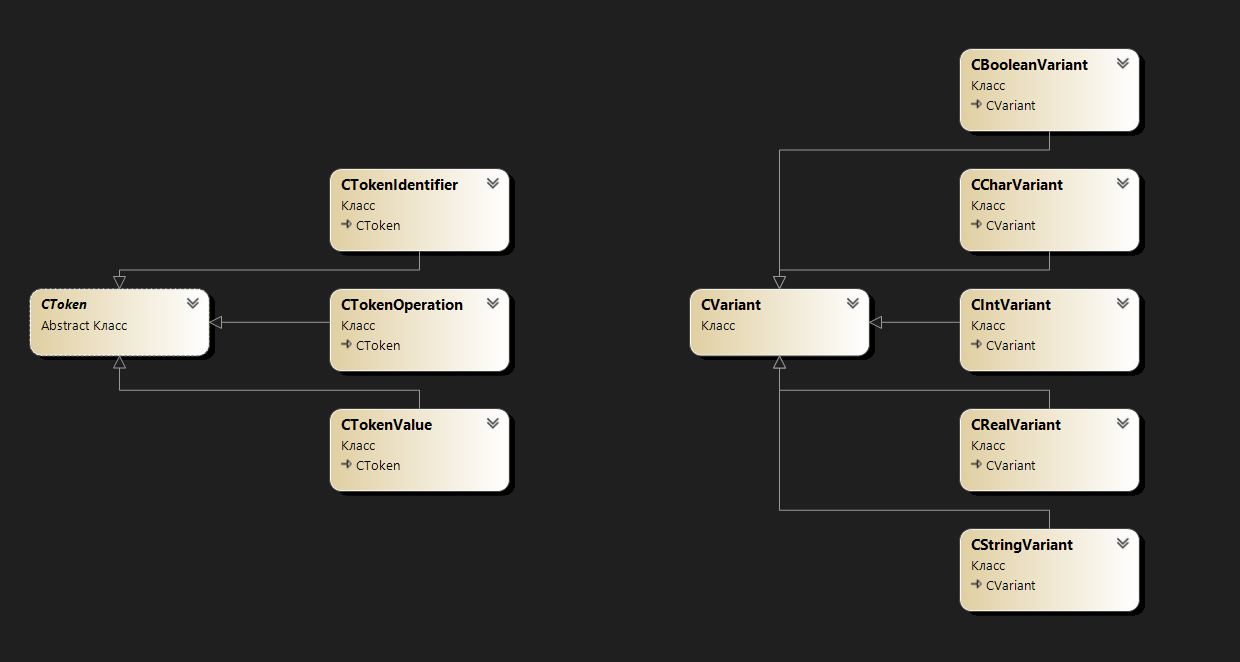
## Проектирование

Нам необходимо понять, как создать структуру классов и ошибок.

Для каждой группы символов нужно будет создать свой отдельный класс, который будет связан иерархически.

Остановимся на данной реализации, где есть класс родитель для всех остальных типов (идентификатора, спец символа, константы).

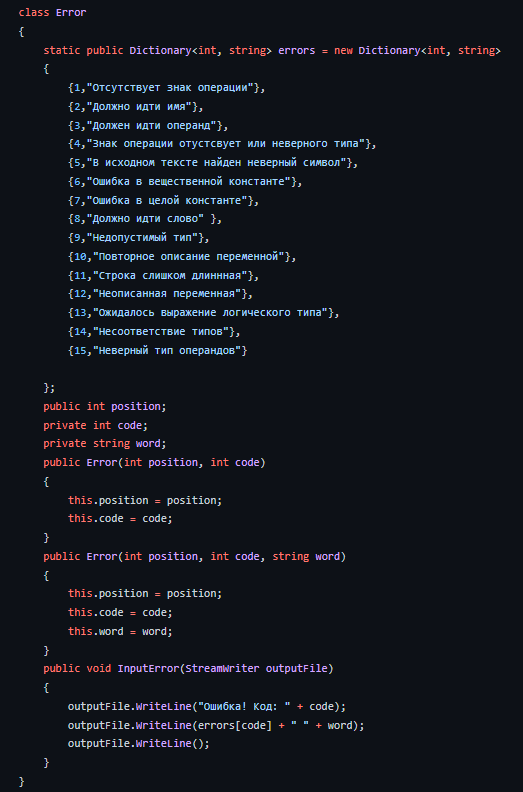
И для символа констант сделаем такой же родительский класс для типов констант (целого, логического, вещественного строкового)



## Реализация программы

Все спец слова должны хранится в специальном словаре, где ключ — это строка слова, а значение - его код, базовые типы были отнесены туда же.

Так же был создан класс для хранения ошибок, а внутри него словарь для ошибок и их описаний.

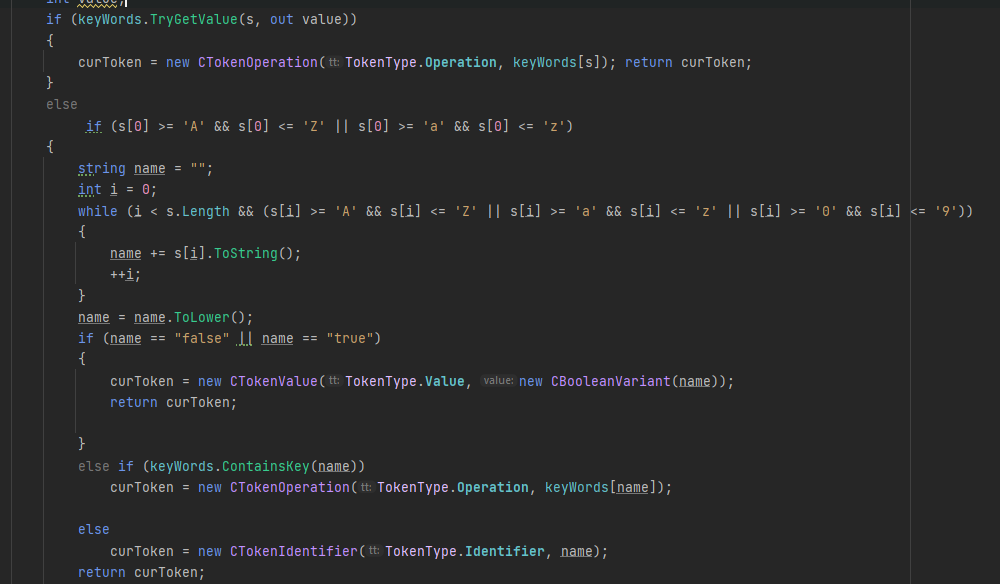


Ошибки хранились в листе ошибок и выводились после каждой строки программы.

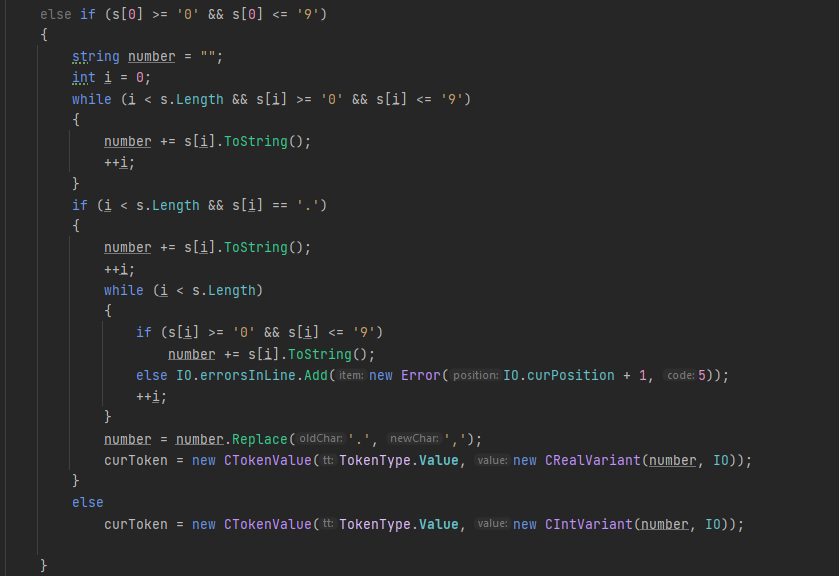
Ошибки, связанные с преобразованием типа, обрабатывались непосредственно в классе, связанном с этим типом.

Классификация лексем на символы

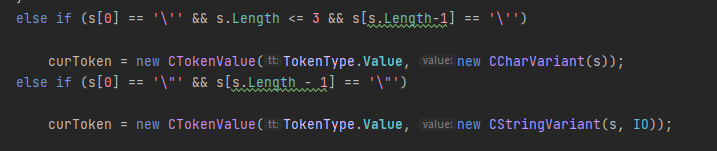
Распознавание ключевых слов или идентификаторов



Распознавание вещественных и целых чисел



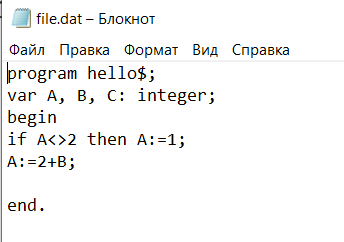
Распознавание стрингов и чаров



## Тестирование

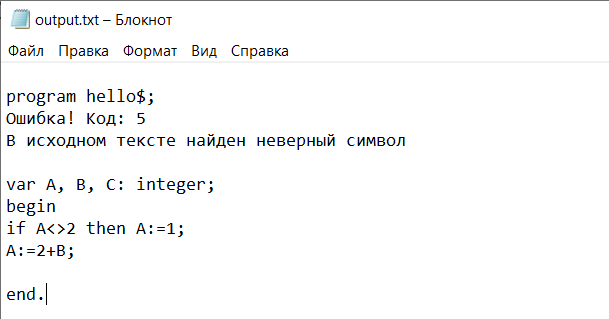
1. Входные данные:

Имеется ошибка, символ не входящий в область допустимых символов



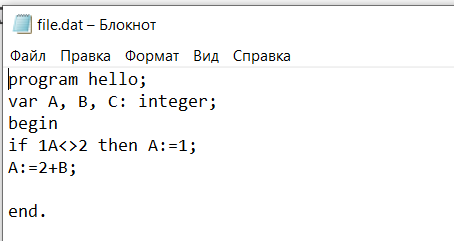
Ожидаемый результат: сообщение об ошибке

Выходной результат:



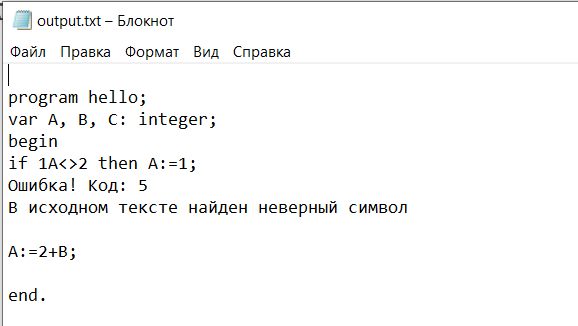
1. Исходный файл:

Имеется ошибка, символ не входящий в область допустимых символов



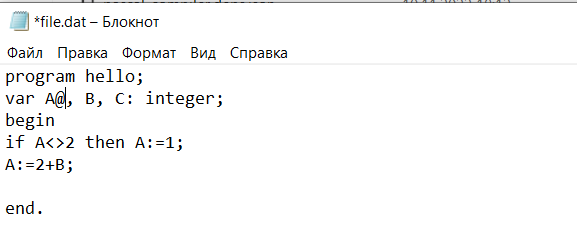
Ожидаемый результат: сообщение об ошибке

Выходной результат:



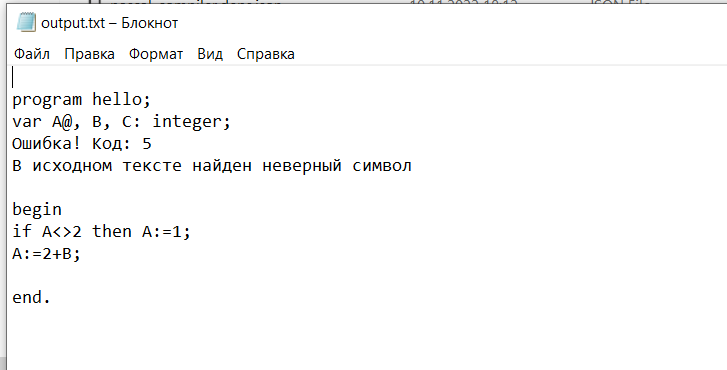
1. Исходный файл

Имеется ошибка, символ не входящий в область допустимых символов



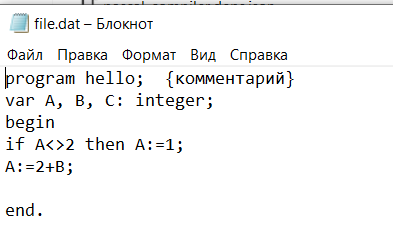
Ожидаемый результат: сообщение об ошибке

Выходной результат:



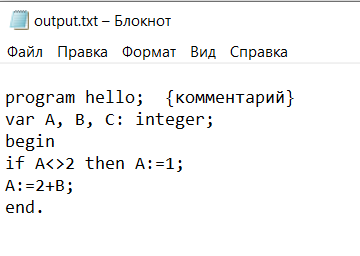
1. Исходный файл

Ошибок нет



Ожидаемый результат: нет сообщений об ошибке

Выходной результат:



# Синтаксический анализатор

## Описание

Синтаксический анализатор проверяет правильность и порядок использования символов языка.

## Проектирование

Синтаксический анализатор строится на правилах использования символов языка, в нашем случае используются БНФ.

Ниже приведены все использованные БНФ:

<программа>::=program <имя>;<блок>.

<имя>::=<буква>{<буква>|<цифра>}

<блок>::=<раздел констант><раздел переменных><оператор>

<раздел констант>::=<пусто>|const <определение константы>;{<определение константы>;}

<определение константы>::=<имя>=<константа>

<константа>::=<число без знака>|<знак><число без знака>|

<имя константы>|<знак><имя константы>|<строка>

<число без знака>::=<целое без знака>|<вещественное без знака>

<целое без знака>::=<цифра>{<цифра>}

<вещественное без знака>::=<целое без знака>.<цифра>{<цифра>}|<целое без знака>.<цифра>{<цифра>}E<порядок>|<целое без знака>E<порядок>

<порядок>::=<целое без знака>|<знак><целое без знака>

<знак>::=+|-

<имя константы>::=<имя>

<строка>::='<символ>{<символ>}'

<раздел переменных>::= var <описание однотипных переменных>;{<описание однотипных переменных>;}|<пусто>

<описание однотипных переменных>::=<имя>{,<имя>}:<тип>

<оператор>::=<простой оператор>|<сложный оператор>

<простой оператор>::=<оператор присваивания>|<пустой оператор>

<оператор присваивания>::=<имя>:=<выражение>

<выражение>::=<простое выражение>|<простое выражение><операция отношения><простое выражение>

<операция отношения>::==|<>|<|<=|>=|>|in

<простое выражение>::=<знак><слагаемое>{<аддитивная операция><слагаемое>}

<аддитивная операция>::=+|-|or

<слагаемое>::=<множитель>{<мультипликативная операция><множитель>}

<мультипликативная операция>::=\*|/|div|mod|and

<множитель>::=<имя>|<константа без знака>|(<выражение>)

<константа без знака>::=<число без знака>|<строка>|<имя константы>|nil

<пустой оператор>::=<пусто>

<пусто>::=

<сложный оператор>::=<составной оператор>|<выбирающий оператор>|<оператор цикла>

<составной оператор>::= begin <оператор>{;<оператор>} end

<выбирающий оператор>::= if <выражение> then <оператор>|if <выражение> then <оператор> else <оператор>

<оператор цикла>::= while <выражение> do <оператор>

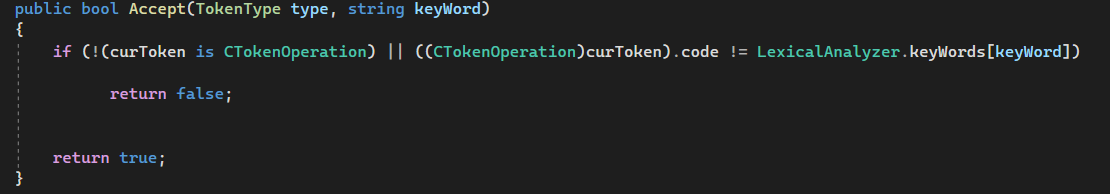
Для обработки событий, что наша программа не соответствует БНФ будем использовать лист ошибок и выводить ошибки после каждой строки программы.

Принятие решений при выборе альтернатив положим на этом же уровне БНФ, анализируя текущий символ.

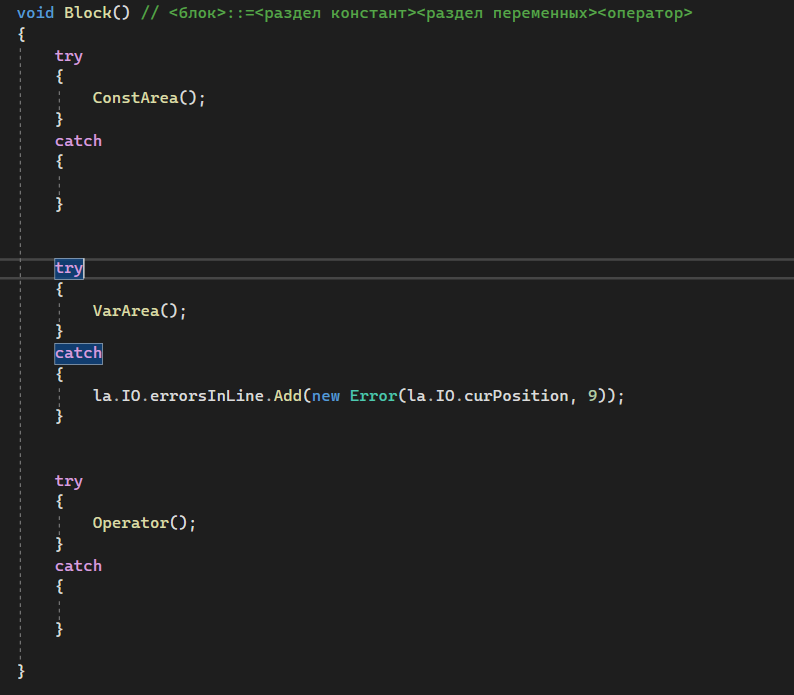
Так как синтаксический анализатор тесно взаимодействует с семантическим, мы будем рассматривать в этой части отчёта части семантического анализатора.

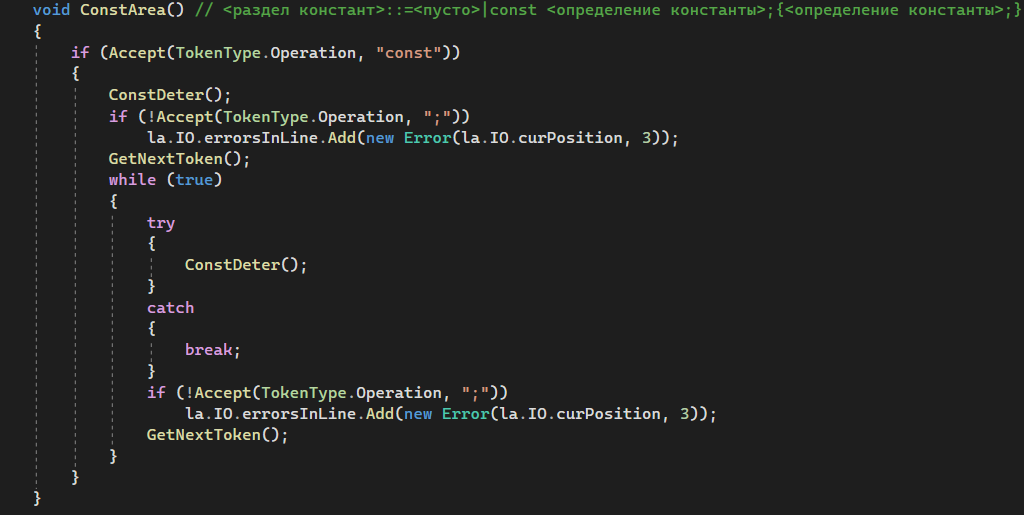
## Реализация

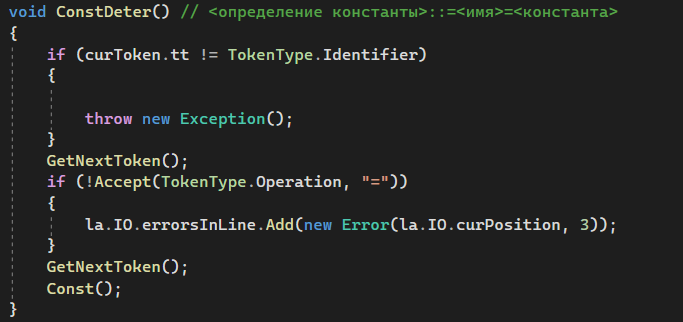
Проверка соответствия спец символа

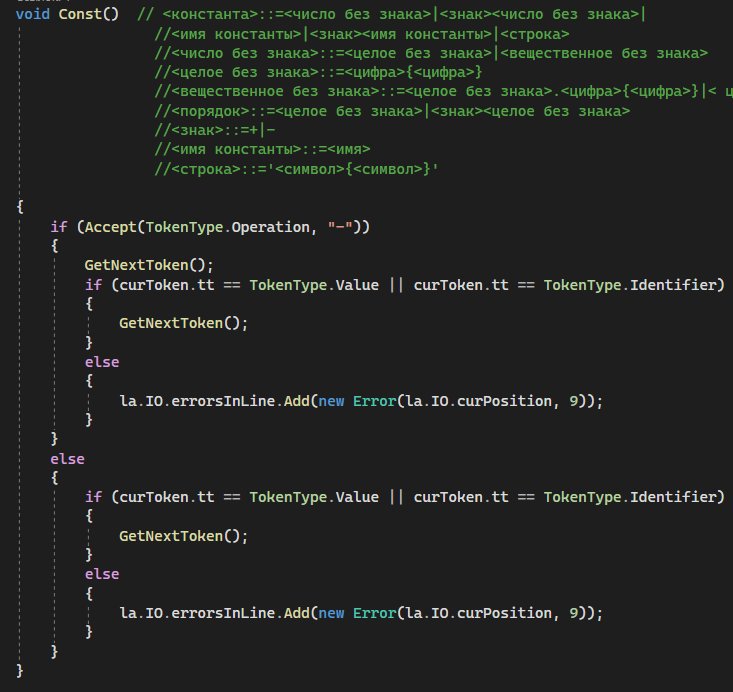


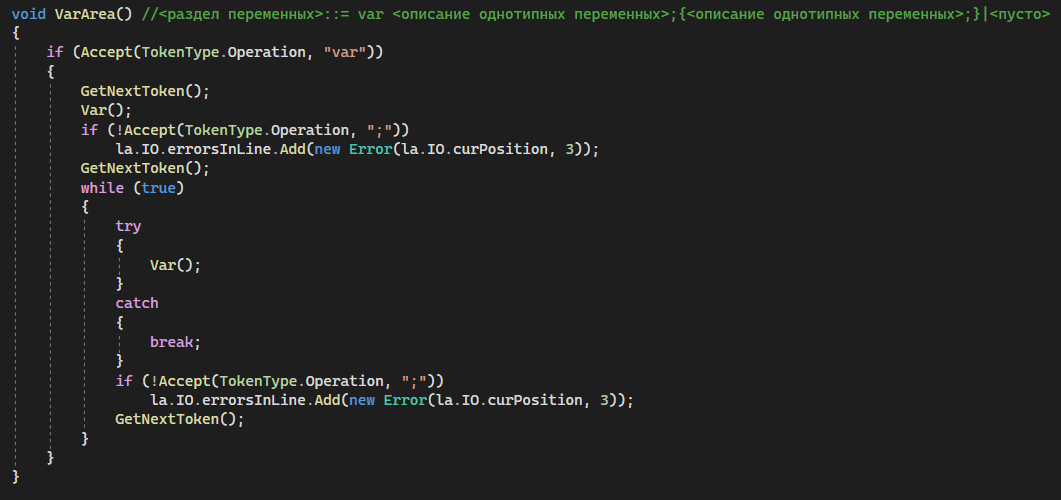
Все методы запрограммированы в соответствии БНФ





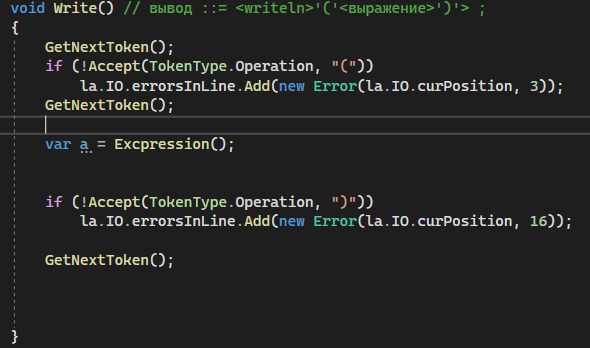


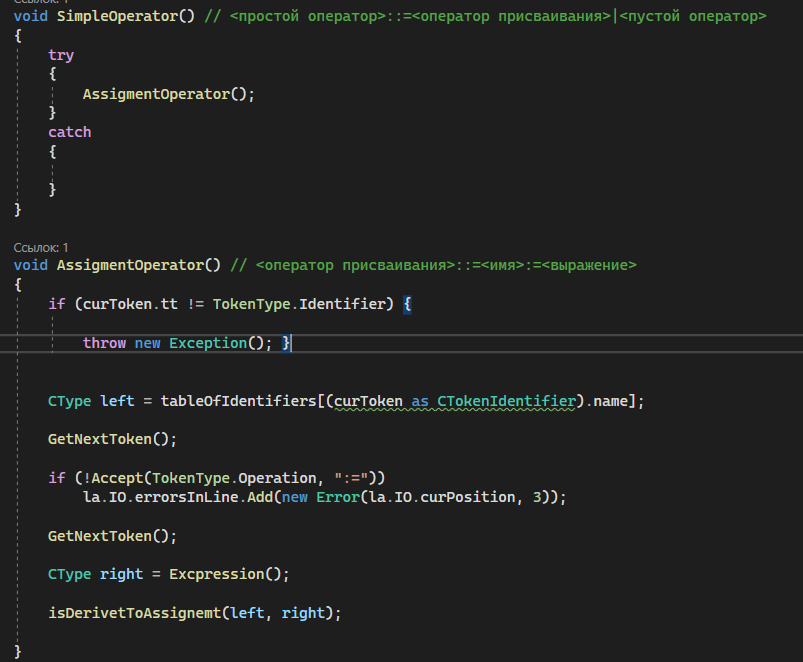


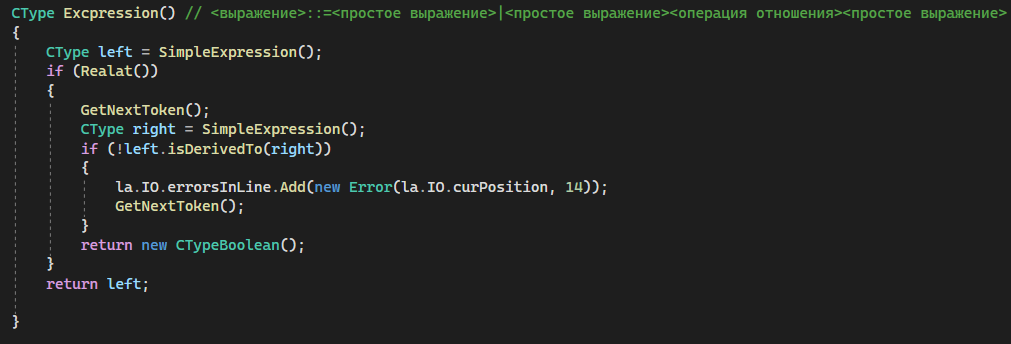


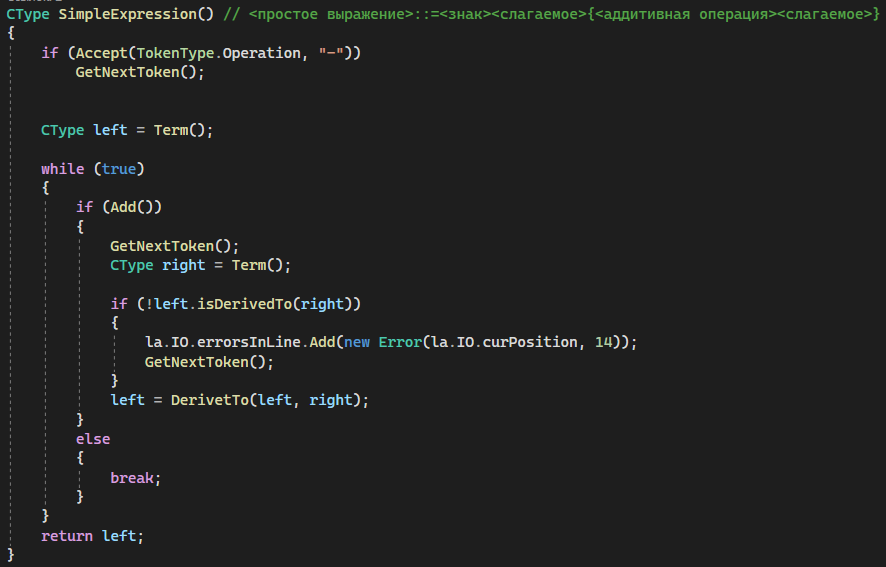


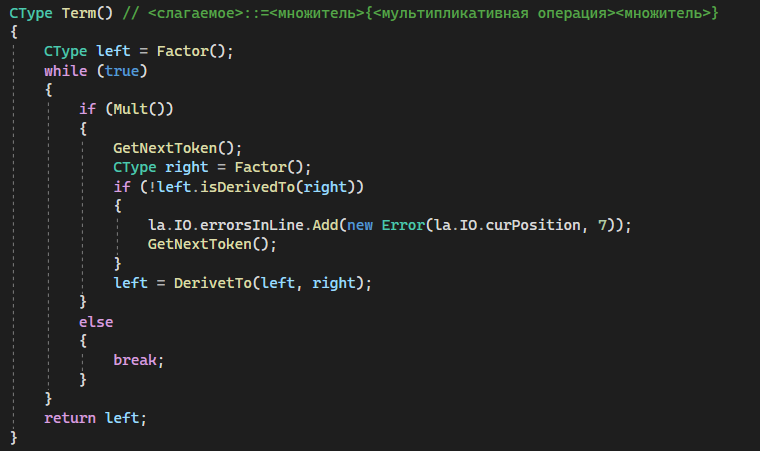


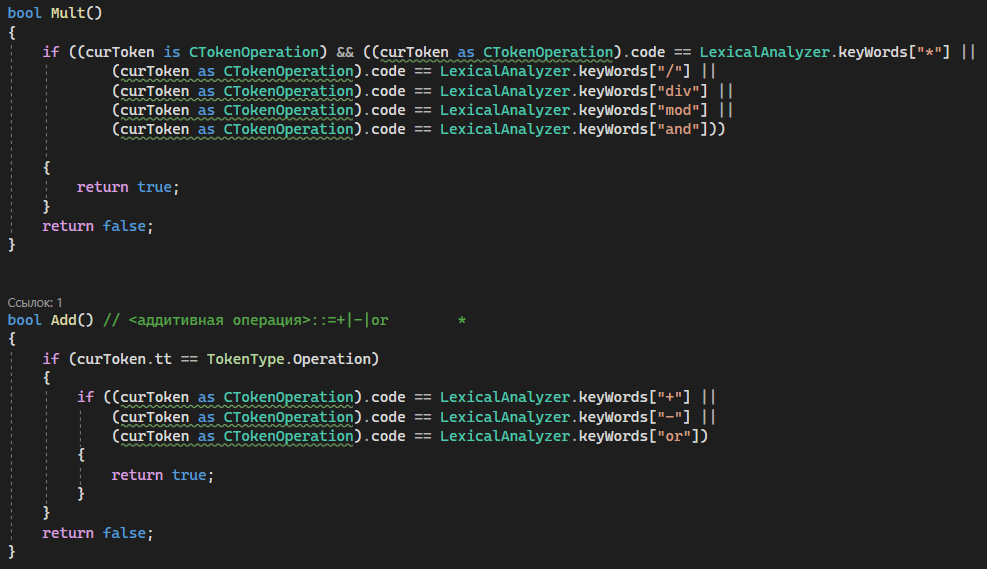


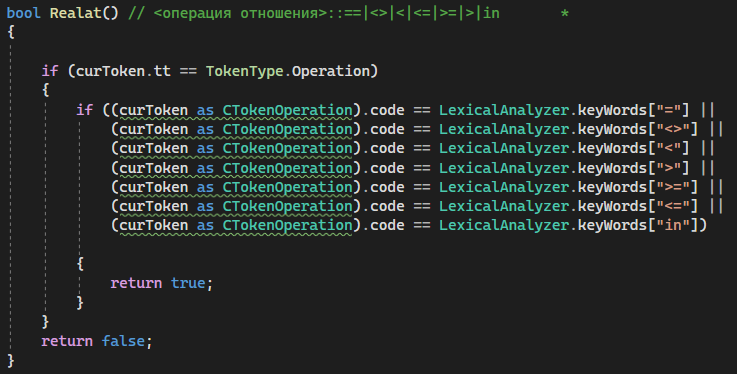


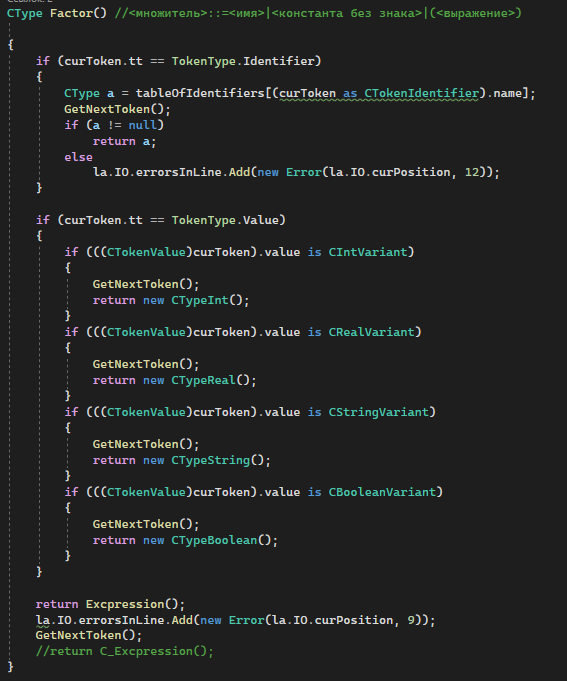


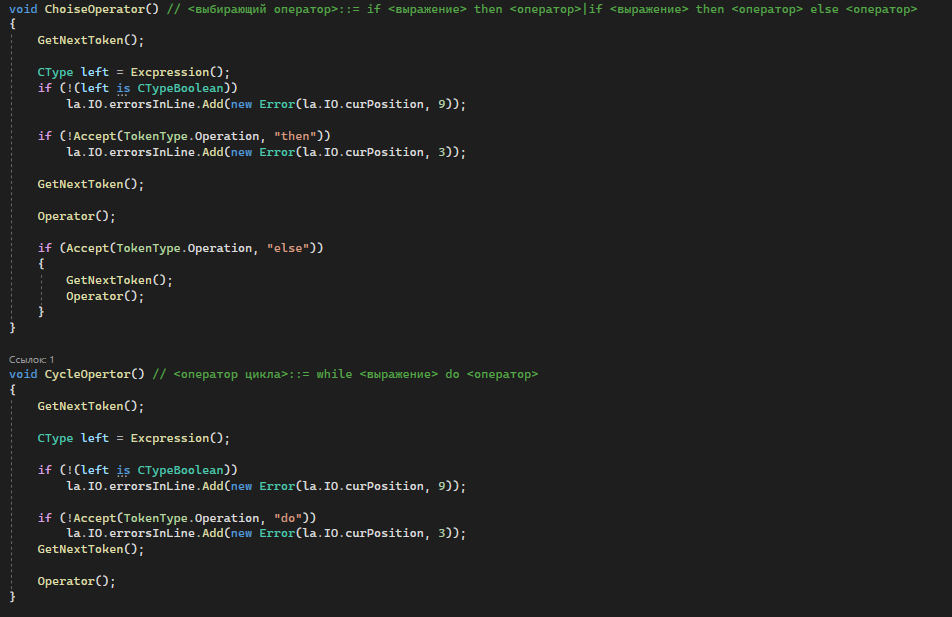










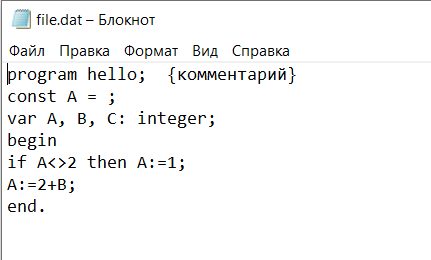


## Тестирование

Работа с простыми арифметическими операторами и сложными операторами

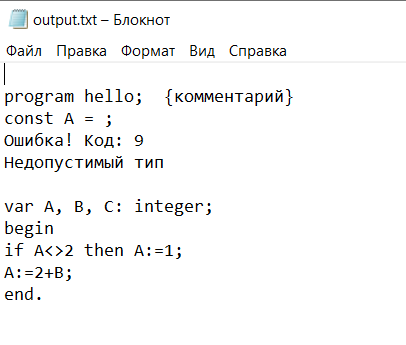
1. Входные данные:

Есть ошибка – присваиваем константе пустое значение



Ожидаемый результат: сообщение об ошибке

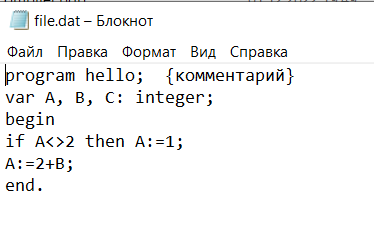
Выходной результат:



1. Входные данные:

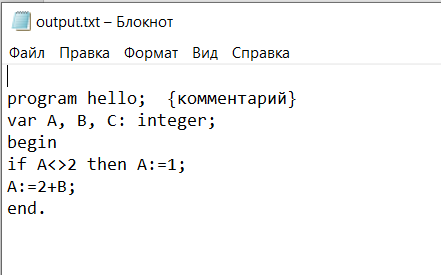
Работа с простыми арифметическими операторами

Работа с оператором ветвления



Ожидаемый результат: отсутствие ошибок и корректная работа программы

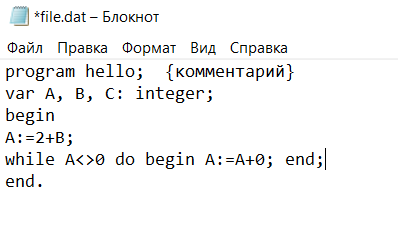
Выходной результат:



1. Входной результат:

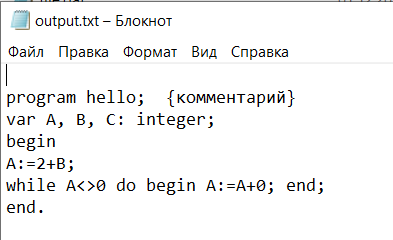
Работа с оператором цикла

Входной результат:



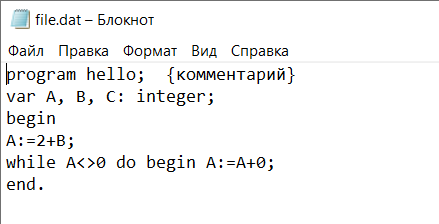
Ожидаемый результат: отсутствие ошибок и корректная работа программы

Выходной результат:



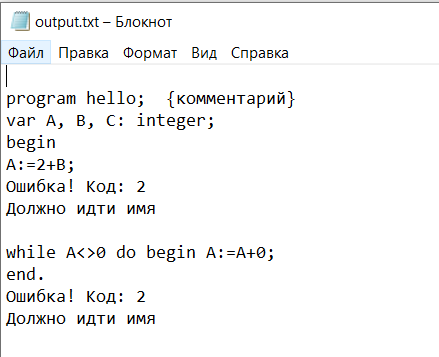
1. Входной результат:

Есть синтаксическая ошибка – не хватает «end;»



Ожидаемый результат: сообщение об ошибке

Выходной результат:



# Семантический анализатор

## Описание

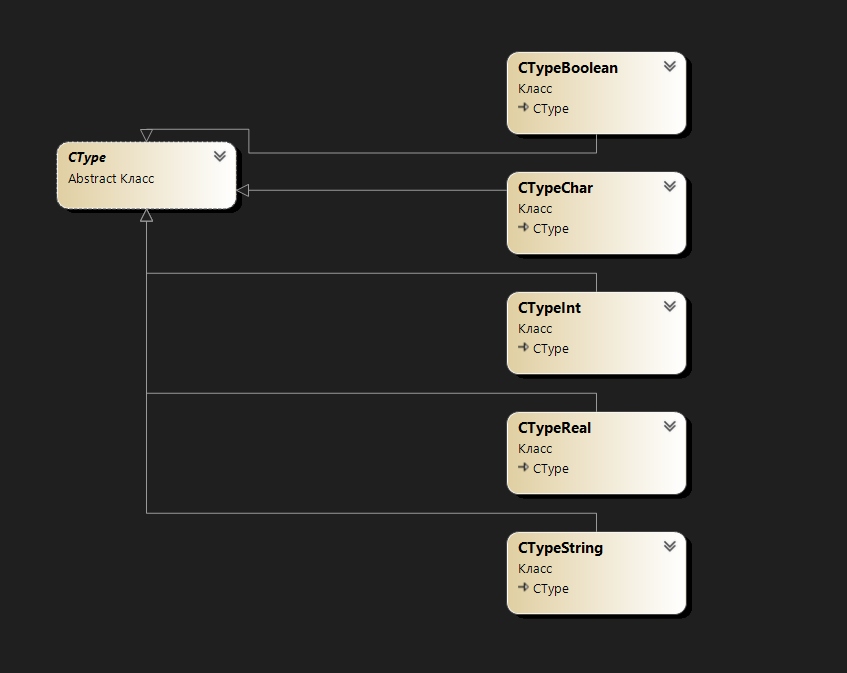
Семантический анализатор проверяет наличие соответствия типов, инициализацию всех переменных.

## Проектирование

Нам необходимо проверять приводимость типов и инициализацию переменных.

Для этого нам нужно понимать какой тип возвращает оператор.

Были определены 5 основных типов:



Так же необходимо создать таблицу идентификаторов для хранения идентификатора и его типа.

Таблицы будет две, для хранения неописанных и описанных переменных.

Таблицу идентификаторов мы заполняем в блоке описания переменных.

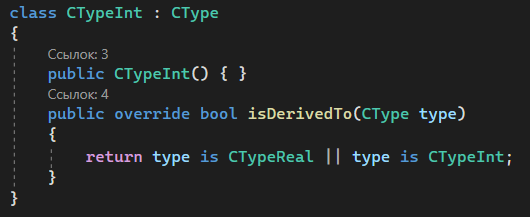
Правила приводимости типов

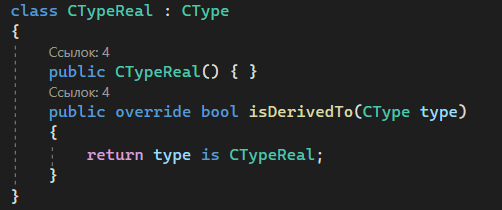
1. Присваивание
   1. Целому значению можно присвоить только целое значение
   2. Вещественному значению можно присвоить целое или вещественное значение
   3. Строковому типу можно присвоить только строковое значение
   4. Логическому типу можно присвоить только логическое значение
2. Операции арифметические и логические
   1. Проводить операции можно целыми и вещественным значениями
   2. Остальные только с такими же типами

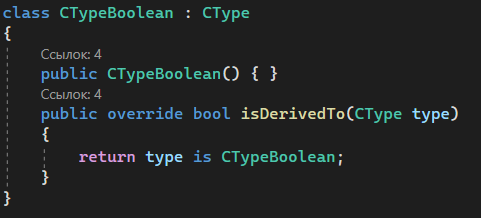
## Реализация

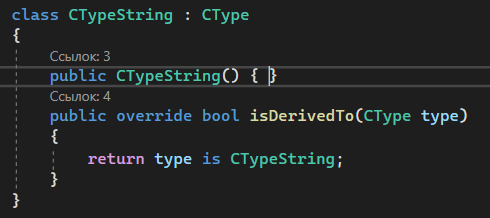
Приводимость типов мы определяем по 2 действиям: присваивание и арифметические и логические операции.

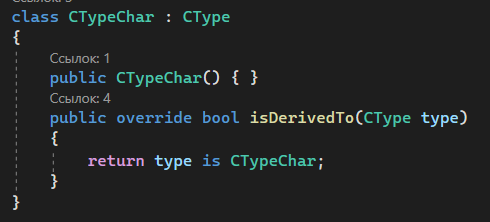
Анализатор для арифметических операций хранится как метод для каждого класса отдельно



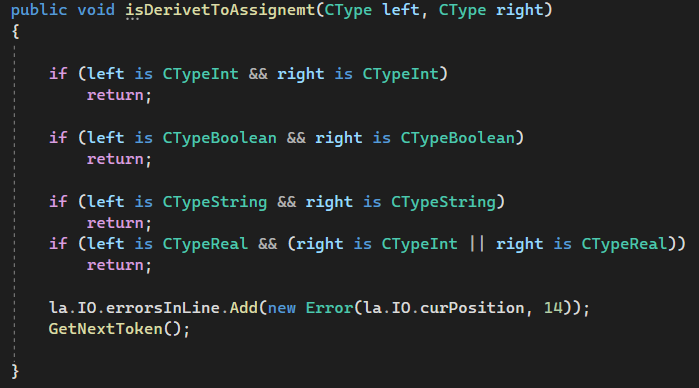




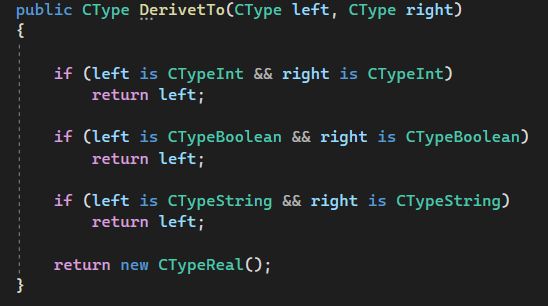




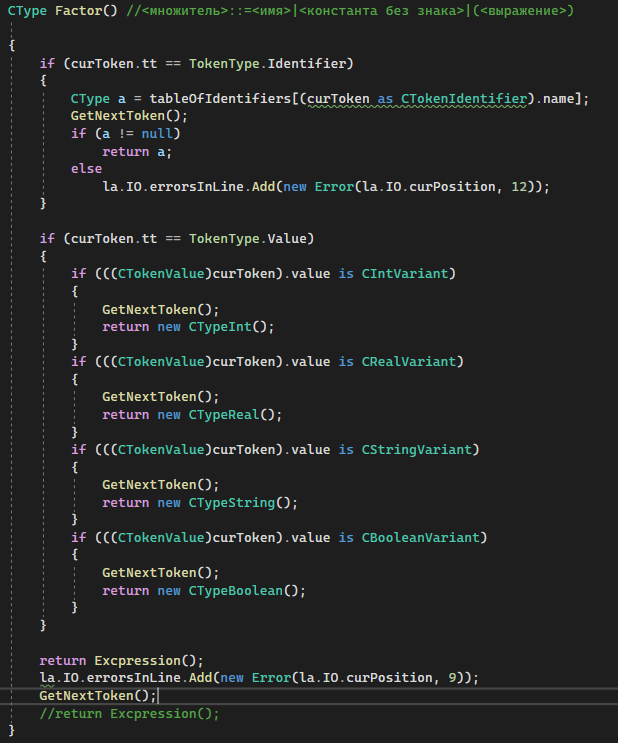
А приводимость для присваивания работает как метод анализатора



Так же есть метод для приведения типов

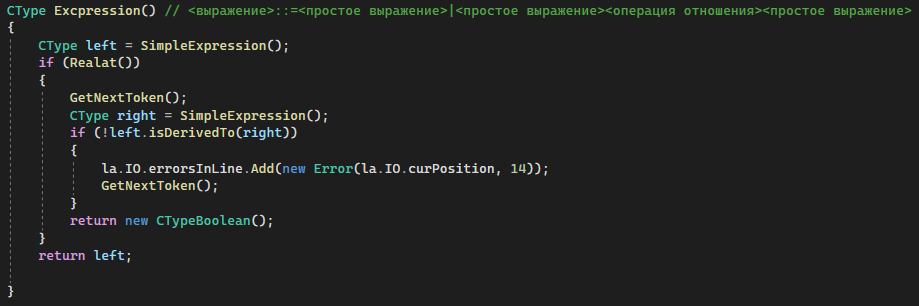


Так как все операции, связанные с типами, происходят в БНФ, связанных с операциями, они должны возвращать тип значения, с которым работают (<слагаемое>, <простое выражение>, <выражение>, <множитель>).







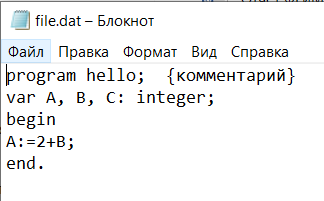


Проверка типа возвращенного нижней БНФ происходит в вызвавшей БНФ.

## Тестирование

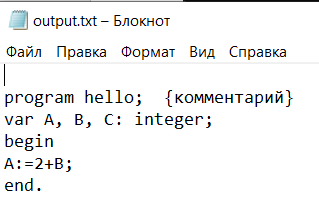
Проверка приводимости типов при арифметической операции и присваивания

1. Входные данные:



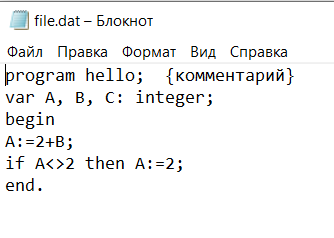
Ожидаемый результат: Отсутствие ошибок, нормальная работа программы без прерываний

Выходной результат:



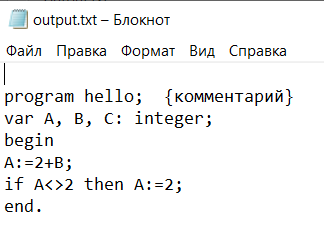
Проверка приводимости типов при арифметической операции и присваивания, и логической операции

1. Входные данные:



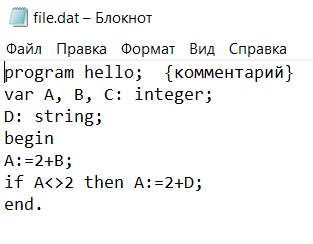
Ожидаемый результат: Отсутствие ошибок, нормальная работа программы без прерываний

Выходные данные:



Ошибка в привидении типа (проверка нейтрализации ошибок)

1. Входные данные:



Ожидаемый результат: ошибка не приводимости типов

Выходной результат:

