|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ | | | | | | | | | | |
|  | | |  | | | | |  | | |
| ФГБОУ ВО «Пермский государственный национальный  исследовательский университет» | | | | | | | | | | |
|  | | |  | | | | |  | | |
|  | | ОТЧЕТ  по дисциплине «Формальные грамматики и методы трансляции» | | | | | | |  | |
|  | | |  | | | | |  | | |
|  | Работу выполнил  студент гр. ПМИ-1,2  Жуков А.Е. \_\_\_\_\_\_\_  (подпись)  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2021 | | | |  | Работу проверил  ассистент кафедры МОВС  Пономарев Ф.А. \_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_» \_\_\_\_\_\_\_\_ 2021 | | | |  |
|  |  | | | |  |  | | | |  |
|  | | | | Пермь 2021 | | |  | | | |

# Техническая документация

1. Компиляторы: принципы, технологии и инструментарий, 2 е изд . : Пер . с англ. — М. : ООО “И.Д. Вильямс”, 2018. — 1184 с. : ил. — Парал. тит. Англ (на английском не осилил, много времени в перевод уходит☺)
2. Разработка Паскаль-компилятора / Л.А. Залогова.

# Лексический анализатор

## Проектирование

Я долго думал, много читал, чтобы как можно лучше спроектировать все сущности, обдумывая много идей и решений, но так и не пришел ни к чему, как мне кажется, нужному. Поэтому решил разобраться в самом процессе.

Сразу выделю класс IO, в котором будет метод getNextToken(), по поводу полей, пока не определился, но нужен, как минимум текущий токен, которым управляет класс CToken.

В CToken несколько типов полей: TokenType, EOperationKeyWords, String и CVariant. EOperationKeyWords содержит все ключевые слова, служебные символы и знаки операций. Вся структура следующая:

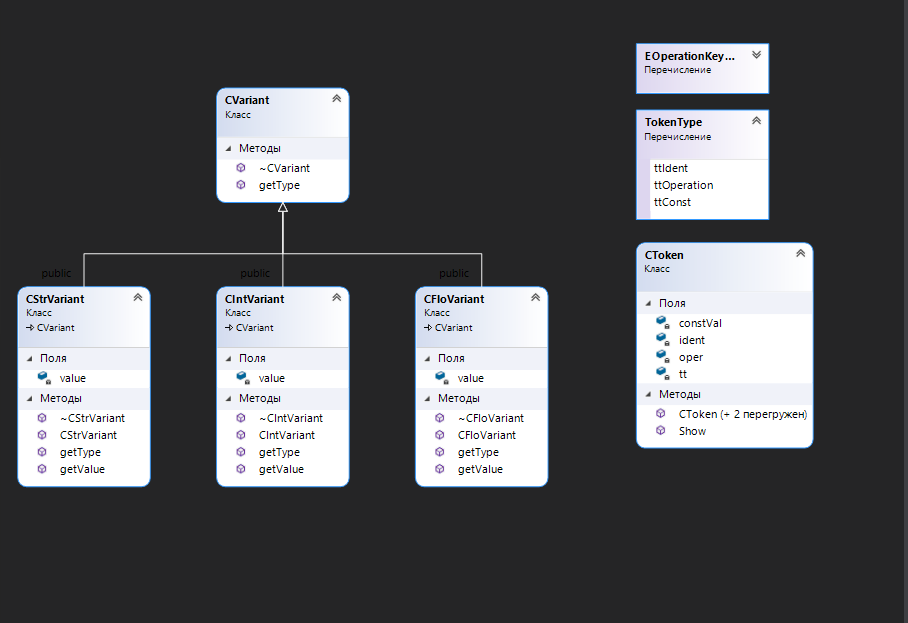


Рисунок 1 – Структуры CToken и CVariant.

Класс IO имеет следующую структуру:

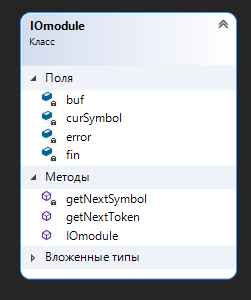


Рисунок 2 – Структура класса IO.

Для получения токенов в методе getNextToken() был реализован автомат. Я посчитал, что он отлично подходит для данной задачи, а также хорошо дополняем в будущем.

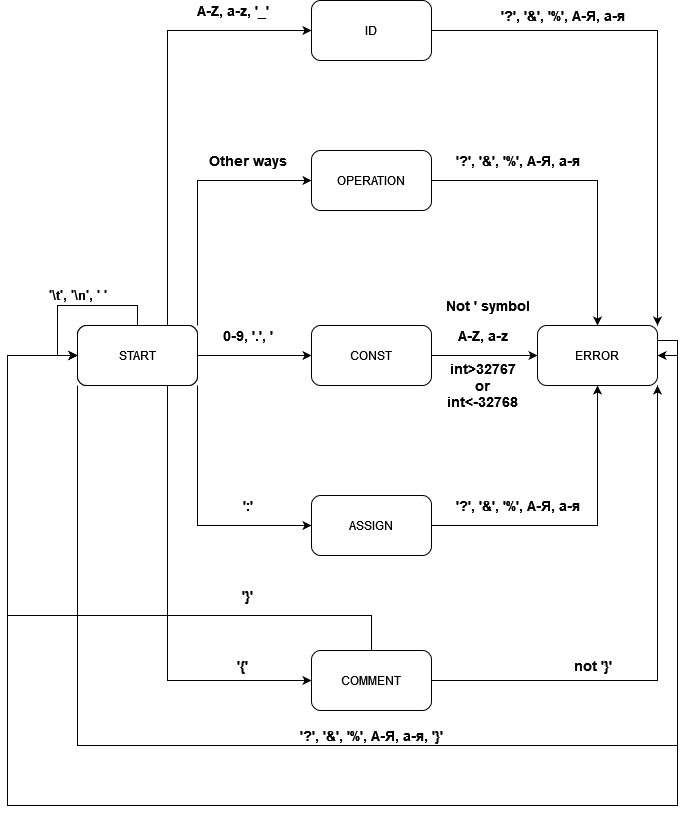


Рисунок 3 – Структура автомата

В модуль ввода-вывода добавил класс lexErrors, который будет хранить информацию о лексических ошибках. В этом классе есть перечисление с именами ошибок, метод show(), для вывода строки, в которой находится ошибка, а так же текст самой ошибки.

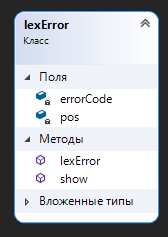


Рисунок 4 – Структура класса lexError.

## Разработка

Начал с описания CVariant. Все методы тут аналогичны, в конструкторе присваивается значение. Метод getValue() возвращает значение каждого варианта, getType() – тип. Так же прописал виртуальные деструкторы, чтобы память освободилась точно, все-таки си два креста, шутки с памятью плохи.

Далее идет CToken. В заголовочном файле имеется перечисление всех ключевых слов и операторов, закинул все в одно, чтобы не усложнять. Создал два map: в одном ключ перечисление, значение – само ключевое слово и операция; в другой поменяны местами ключ-значение. Вторую по факту не использовал в итоге, но может доведу до ума в конце, чтобы быстрее работало. По методам по факту 3 конструктора, принимающие тип токена, и в зависимости от типа значение: ключевое слово или операция, строку с идентификатором, константа, которую храню в ранее описанном классе CVariant. Метод Show() выводит информацию о токене: тип и значение. Для констант использую dynamic\_cast, чтобы привести базовый абстрактный класс к дочерним и вытянуть данные.

Сам модуль ввода-вывода содержит, кроме себя самого класс с лексическими ошибками, про них чуть позже. По факту самый существенный метод в модуле ввода-вывода – получение следующего токена. Конструктор получает на вход имя файла, открывает его и заполняет им строку. Я не думаю, что будет настолько большая программа, что std::string выдаст overflow, по крайней мере, в данной работе это точно сработает. Прописаны две вспомогательные функции: первая сдвигает указатель с текущего символа на следующий и возвращает символ; вторая ищет в map индекс оператора или ключевого слова, если не находит, то -1. Казалось бы, в map 58 элементов, искать должно быстро, но есть проблема, что идет сравнение строк, что не очень быстрая операция, но пока пусть будет так, опять же можно вернуться и замутить оптимизацию.

Получение токенов происходит через метод getNextToken(). Это автомат с семью состояниями. Я обернул его в while(true) в котором switch(state). Все программируется согласно спроектированному автомату. Есть строка для токена, которая может приводится к константам. Все состояния, за исключением ERROR и COMMENT возвращают новый токен.

Состояние ID для идентификаторов. Считываем до того момента, пока поступают допустимые символы идентификатора. Проверка на ключевое слово, если идентификатор ключевое слово – возвращаем, что токен ttOperation, иначе ttIdent.

Состояние OPERATION для операций. Сначала проверка на знаки сравнения, потому что есть <>, <=, >=. Проверка по первому символу, дальше в зависимости от второго возвращаем ту операцию, которая была. Если же это не оператор сравнения, то может быть ключевое слово. Считываем, пока на вход подаются буквы. Так, если пришел +, - либо что-то из похожего, то цикл просто скипнется. Ищем индекс ключевого слова в map, возвращаем токен.

Состояние CONST для константных значений. Первым делом проверка на символ 39 из ASCII, что является одинарной кавычкой, то есть в случае строковой константы. Строковая константа заканчивается так же одинарной кавычкой. Если же это не строковая константа, то значит, что числовая, считываем пока на входе цифры, при получении точки запоминаем, что это вещественная константа. Как только цифры заканчиваются возвращаем токен.

Состояние ASSIGN для присваивания. Проверяем следующий символ после «:», если это =, то возвращаем токен оператора присваивания, иначе просто «:».

Состояние COMMENT для пропуска комментариев. Просто скипаем все символы, пока не получим }.

Состояние ERROR для вывода ошибок.

Для хранения ошибок используется класс lexError. Конструктор получает на вход код ошибки и позицию. Я не храню список всех ошибок, а только одной. То есть получив ошибку, я создаю экземпляр этого класса, вывожу ошибку, удаляю созданный экземпляр. Выделил на этапе лексического анализа 7 ошибок: запрещенный символ, неправильное именование идентификатора, превышение максимальной длины идентификатора, неоткрытый комментарий, незакрытый комментарий, превышение максимального значения для integer, незакрытая строковая константа. Так, метод show() класса lexError выводит строку, в которой произошла ошибка, указывает на символ, в котором ошибка, выводит текст ошибки. Единственное, что мне не нравится в этом всем, что я использую глобальную переменную с текстом программы.

Почти в каждом состоянии идет проверка на запрещенные символы. START может вернуть ошибку, что комментарий не открыт, при получении символа }.

ID может вернуть ошибку о том, что превышена максимальная длина идентификатора.

CONST может вернуть несколько ошибок. Строка не закрыта, если достигнут конец текста программы, а это все еще строковая константа. Неправильное имя идентификатора: в это состояние проваливаемся, когда получаем число, идентификатор не может начинаться с цифры, таким образом, получая букву в этом состоянии, делаем вывод, что идентификатор имеет неверное имя. Здесь же идет проверка на превышение допустимого значения для integer.

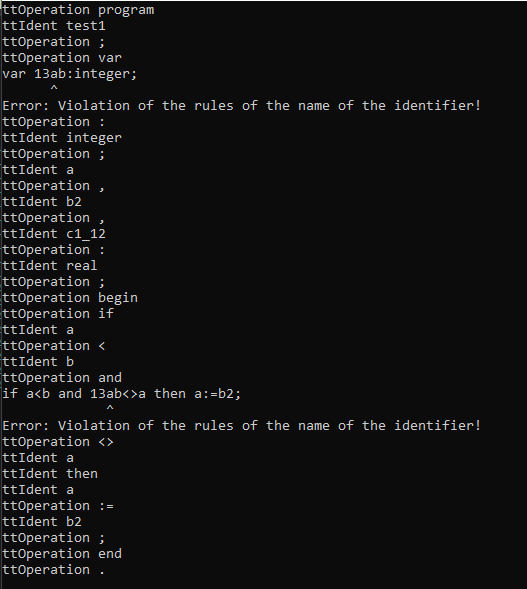
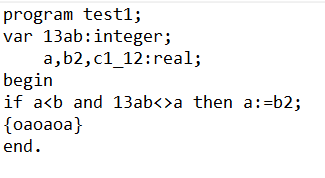
COMMENT возвращает только одну ошибку, когда комментарий не закрыт, то есть текст программы закончился, а символа } так и не поступило.

В состоянии ERROR происходит вывод информации об ошибке, удаление экземпляра класса и переход в стартовое состояние.

## Тестирование

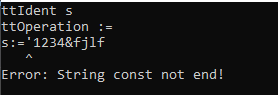
Я просто написал некоторые программки, которые все случаи охватывают, все ошибки, большинство ключевых слов.

1. Неверное именование идентификатора, операции сравнения, пропуск комментариев.

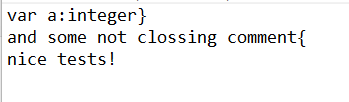


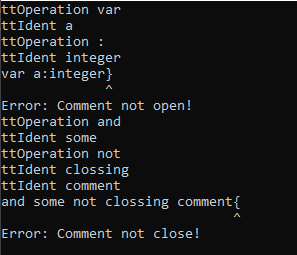
1. Не закрытая строка.

s:='1234&fjlf



1. Неоткрытый комментарий и незакрытый комментарий.





1. Превышение длины идентификатора, выход за пределы integer, запрещенные символы.

