|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Министерство науки и высшего образования РФ | | | | | | | | |
|  | | |  | | |  | | |
| Пермский государственный национальный  исследовательский университет | | | | | | | | |
|  | | |  | | |  | | |
|  | | ОТЧЁТ  по лабораторной работе “Разработка компилятора языка Pascal»  по дисциплине «Формальные грамматики и методы трансляции” | | | | |  | |
|  | | |  | | |  | | |
|  | Работу выполнил  студент гр. ПМИ-3,4  Валеев Р. Р. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_» \_\_\_\_\_\_\_\_ 2021 | | |  | Проверил  асис.кафедры МОВС  Пономарев Ф.А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_» \_\_\_\_\_\_\_ 2021 | | |  |
|  |  | | |  |  | | |  |
| Пермь 2021 | | | | | | | | |

# Анализ

Для решения глобальной задачи – написание компилятора для подмножества языка Pascal, требуется решить несколько подзадач, а именно, спроектировать и разработать модуль ввода-вывода, лексический, синтаксический и семантический анализаторы, а также генератор кода. Первым этапом станет начальная реализация модуля ввода-вывода. Его задача заключается в переводе исходного кода программы в последовательность литер. Также в зоне его ответственности вывод сообщений об ошибках. Но на данном этапе выполнения лабораторный работы мы не будем уделять этому внимание. Перед проектированием важно более детально узнать информацию о структуре компилятора. Начальные сведения были получены из первых лекций по этому предмету, а также после знакомства первыми главами книги Залоговой «Разработка Паскаль-компилятора». Ответы на вопросы, возникающие в ходе этапов планирования и разработки, удавалось находить на следующих источниках: <https://docs.microsoft.com/ru-ru/>, [stackoverflow.com](https://stackoverflow.com/), <https://habr.com> и [www.cyberforum.ru](http://www.cyberforum.ru). В основном это был поиск идей для решения той или иной проблемы или прочтение документации о неизвестных мне возможностях языка C#.

# Проектирование

Перед началом разработки было проведено начальное проектирование. На рисунке ниже представлена спроектированная диаграмма классов

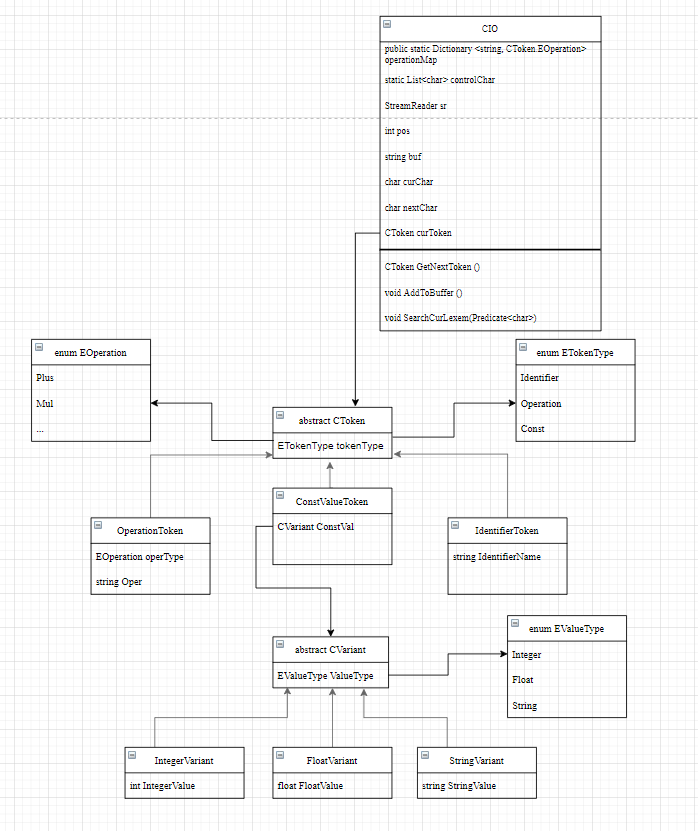


Рисунок 1 – Диаграмма классов

Каждому типу токена соответствует собственный класс, который наследуется от абстрактного класса CToken. Информация о токене включает в себя тип токена (идентификатор, операция, константное значение) и в зависимости от типа токена либо тип операции, либо значение константы, либо название идентификатора. Для хранения значения константы был спроектирован класс CVariant. Он является абстрактным и хранит в себе допустимые типы данных. Классы IntegerVariant, FloatVariant, StringVariant унаследованы от CVariant и хранят значение типов int, float и string соответственно.

Модуль ввода-вывода вынесен в отдельный класс. В данном классе будет хранится информация о последней считанной лексеме, о просматриваемом и следующем элементах, а также текущая позиция в строке текста программы. Также в нём хранится словарь операций operationMap, в котором последовательность символов сопоставляется с соответствующим токеном, и список управляющих символов ('\n', '\t', '\r', ' ', '\0'), элементы которого нужно игнорировать при считывании лексем. Для удобства реализации было принято решение хранить токены ключевых слов и операций в одном enum’е EOperation. Метод AddToBuffer() ответственен за получение символов из текста. При необходимости получения следующей лексемы происходит посимвольное считывание. При считывании символов исходного текста программы запоминается текущий и следующий после него элементы. Метод GetNextToken() возвращает следующую лексему из текста программы. В ходе её вызова сначала в виде строки лексема формируется в буфере buf. После того как лексема была сформирована, то создаётся её токен, который будет возвращён в результате вызова метода. В случае успешного формирования лексемы буфер очищается. На текущем этапе, ошибки, возникшие при считывании лексем, не фиксируются и, следовательно, информация о них выводится не будет.

# Разработка

После окончания этапа проектирования начался этап разработки. Сначала был реализован класс CToken и все его дочерние классы, связанные с хранением различных типов лексем. После чего началась разработка модуля ввода-вывода. Он построен в виде класса с названием CIO. Данный класс ответственен за перевод последовательности символов в лексемы. Так как в языке Pascal допускаются составные операторы, то для корректной интерпретации считанной последовательности может также потребоваться информация о следующем элементе. Поэтому в качестве полей класса CIO определены переменные типа char - curChar и nextChar, которые хранят в себе информацию о текущем и следующем символах. Класс CIO также имеет методы, такие как, AddToBuffer(), SearchCurLexem(), GetNextToken(). Основное время было уделено реализации последнего метода GetNextToken(). Это основой метод, выполняющий работу по определению лексемы и формированию токена. При вызове метода первым делом осуществляется проверка того, что не был достигнут конец файла. Если этого не произошло, то мы считываем следующий символ. Метод AddToBuffer() ответственен за считывание нового символа из текста программы. В ходе его выполнения следующий символ после текущего (nextChar) становится текущим (curChar) и из файла считывается ещё один символ, который будет сохранён как следующий символ (nextChar). Предыдущий считанный символ добавляется в строковую переменную buf. Реализация данного метода представлена ниже:

private void AddToBuffer()

{

pos++;

curChar = nextChar;

int readChar = sr.Read();

nextChar = readChar >= 0? (char)readChar : '\0' ;

buf += curChar.ToString().ToLower();

}

Далее приведен порядок разбора константных значений, идентификаторов, ключевых слов и операций. Распознавание числовой константы происходит следующим образом:

1. если текущий символ curChar является цифрой, то перейти к пункту 2;
2. пока nextChar является цифрой или равен символу «.», но при этом curChar является цифрой, то считать новый символ;
3. полученная последовательность символов в буфере анализируется:

* если символы curChar и nextChar совпадают и равны «.», то вернуть токен целочисленной константы для строки buf, не включающей в себя последний добавленный элемент и оставить в буфере только последний символ;
* иначе, если буфер содержит символ «.», то вернуть токен вещественной константы, иначе - токен целочисленной константы, и очистить буфер.

Распознавание идентификатора или ключевого слова:

1. если текущий символ curChar является буквой или символом «\_», то перейти к следующему пункту;
2. пока nextChar есть буква или символ «\_», то считать новый символ;
3. полученная последовательность символов в буфере анализируется:

* если последовательность символов в буфере найдена в словаре operationMap, т.е. является ключевым словом, то вернуть токен операции;
* иначе, последовательность символов является идентификатором, поэтому вернуть токен идентификатора.

Распознавание оператора или операции:

1. проверить, что curChar является ключом в словаре operationMap;
2. пока добавление к буферу следующего символа приводит к образованию составного оператора, то считать новый символ;
3. вернуть OperationToken с типом операции operationMap[buf].

Также стоит уделить внимание распознаванию строковой константы. В Pascal она начинается и заканчивается с символа «’». Если считанный символ равен «’», то начиная со следующего элемента проверять следующее: пока текущий элемент curChar не станет равен символу «’» или не будет достигнут конец файла, то считать новый символ. Далее вернуть токен строковой константы.

Таким образом, на данном этапе разработки, в результате вызова метода GetNextToken() возвращается либо считанный токен или null, когда был достигнут конец исходного файла.

Для нахождения конца текущей лексемы используется метод SearchCurLexem(), которому в качестве входного параметра передаётся предикат для объекта char. Переданный предикат определяет условие остановки поиска текущей лексемы. Пока условие остановки не наступило вызывается метод AddToBufer(), который меняет текущий элемент на следующий и считывает новый символ из кода переданной программы. Например, критерий остановки поиска идентификатора есть условие (c => !controlChar.Contains(c) && (char.IsLetterOrDigit(c) || c == '\_')), где вместо c будет подставляться значение элемента nextChar. Реализация данного метода представлена ниже:

private void SearchCurLexem(Predicate<char> condition)

{

while (condition(nextChar))

AddToBuffer();

}

# Тестирование

В этом разделе протестирована работа модуля ввода-вывода на различных входных данных.

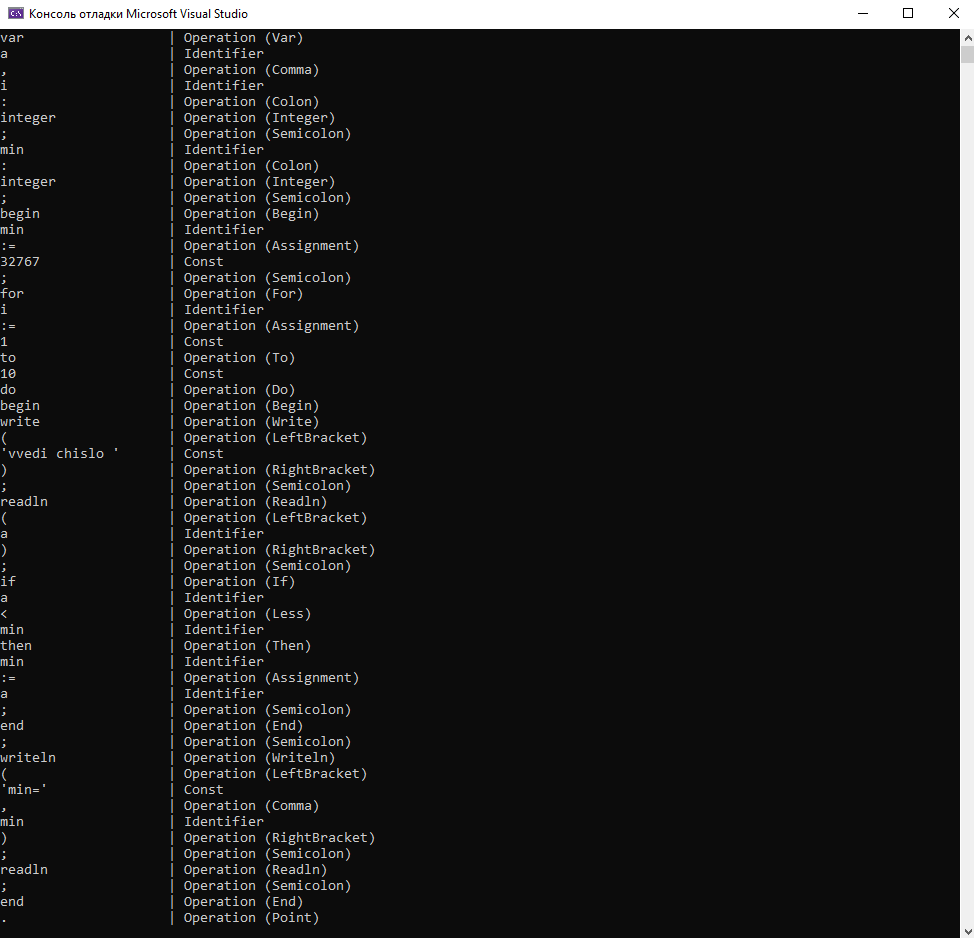
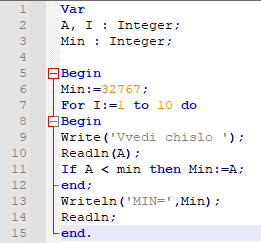


Рисунок 2 – Код №1 для тестирования

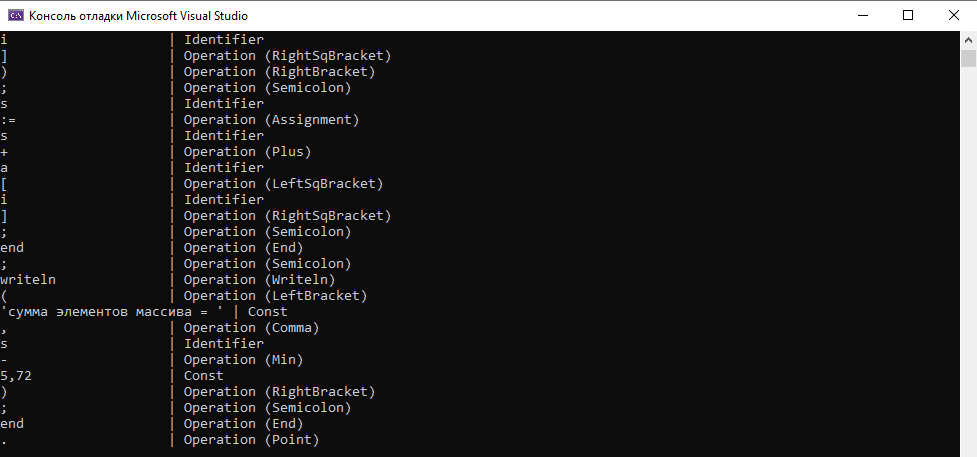
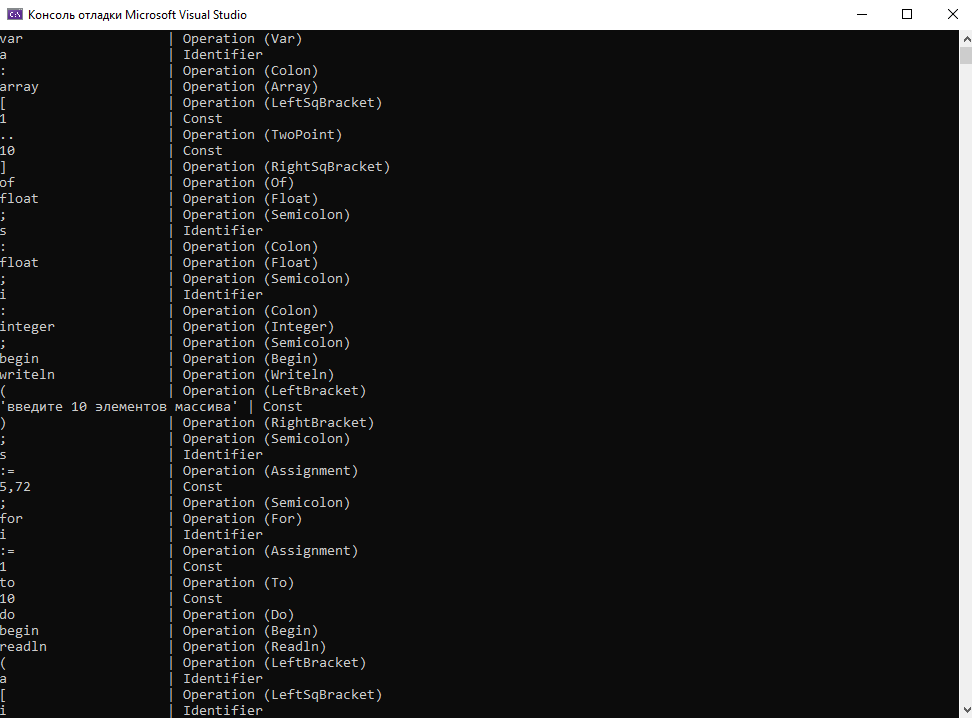
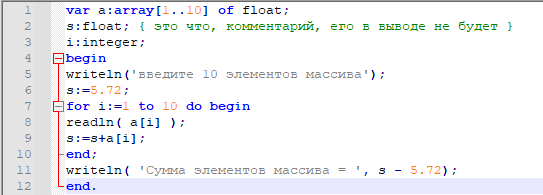


Рисунок 3 – Код №2 для тестирования