Лексический анализатор

Определим следующие типы распознаваемых токенов:

1. Ключевые слова. Список слов: if, else, then, while, begin, end, do, const, for, function, var, to, downto, +, -, \*, /, =, <, <=, >, >=, <>, :=, and, or, not, (, ).
2. Константы. Выделим три типа констант: строковые, целочисленные, вещественные.
3. Идентификаторы. Представляют собой строку однозначно определяющий идентификатор. Например: “x”, “y” и так далее.

Лексический анализатор – программа, принимающая на вход текст (последовательность символов из алфавита) и разбивающая его на подстроки (лексемы) в соответствии с некоторым набором правил.

Далее определим основные классы, необходимые для реализации лексического анализатора.

***CLexer.*** Представляет собой класс лексического анализатора. Хранит строковую константу разделителей токенов (separators), текст программы, строку, хранящую текущий токен, а также позицию текущего читаемого символа в тексте паскаль-программы.

Класс имеет следующие методы: GetTokenType() - определяет тип текущего токена и возвращает умный указатель на созданный объект класса (CIdentToken/ CKeywordToken/CConstToken).

GetNextToken() - выделяет следующий токен, начиная с позиции currentPosition. После выделения токена возвращает умный указатель на него.

IsReal (string s) - проверка строки s на соответствие типу данных real. Для того, чтобы строка соответствовала типу данных real, необходимо, чтобы в числе присутствовала одна точка. Возвращает bool.

IsInteger (string s) - проверка строки s на соответствие типу данных Integer. Чтобы строка соответствовала типу данных integer, необходимо, чтобы в строке присутствовали исключительно цифры.

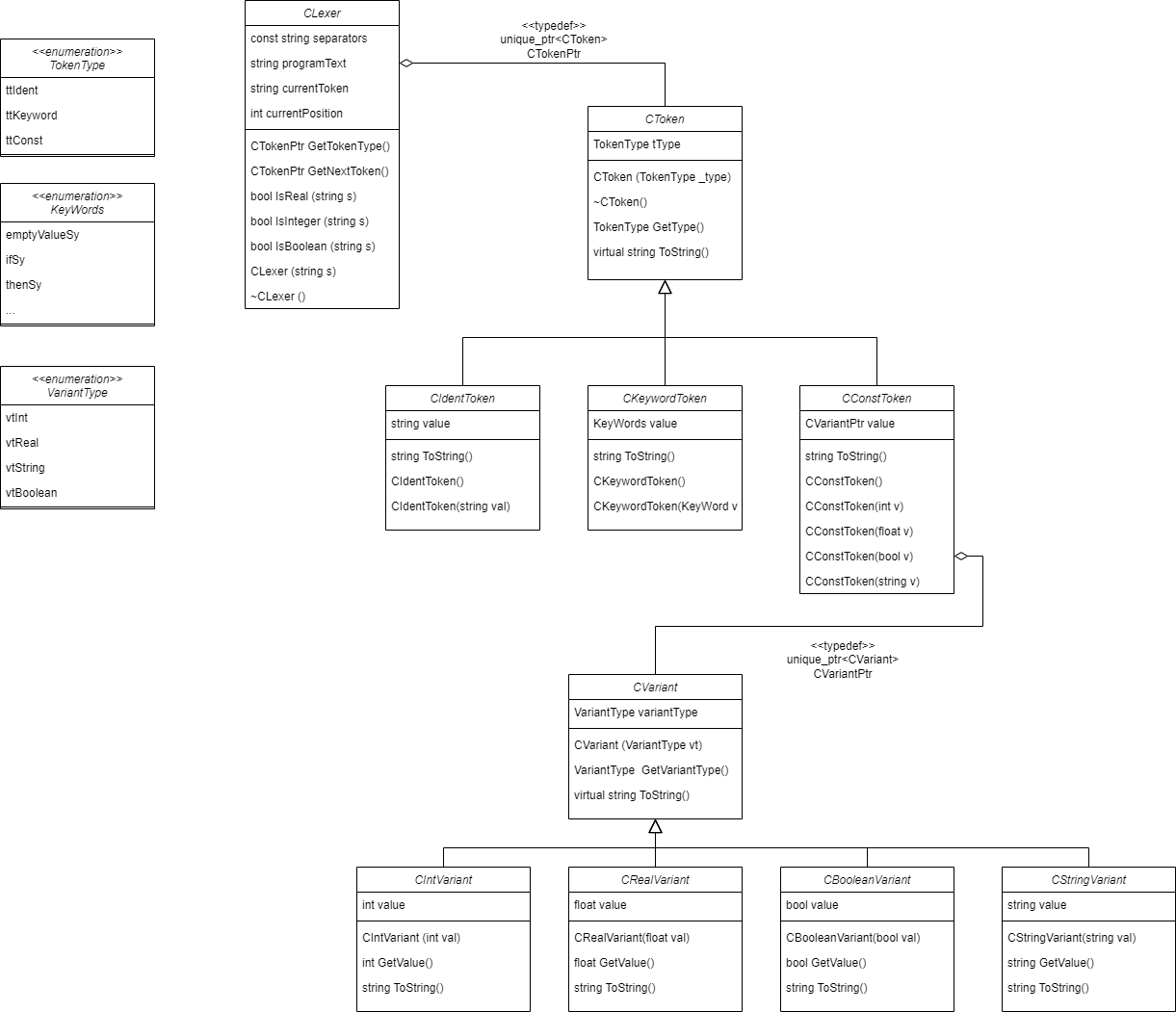
IsBoolean (string s) - проверка строки s на соответствие типу данных boolean. Строка s должна принимать одно из значений: “true”/”false”.

***CToken.*** Абстрактный класс. Наследуемые классы имеют поле типа TokenType, характеризующее один из трех видов токенов. Метод GetType() возвращает значение приватной переменной типа TokenType. Метод ToString() возвращает строковое представление токена.

***CIdentToken / CKeywordToken / CConstToken.*** Наследуются от CToken. Представляются как три разных вида токенов - идентификаторы, ключевые слова, константы. Все три класса переопределяют метод ToString(), а также имеют различные поля. CIdentToken имеет поле string, т.к. идентификатор представляет собой строку. CKeywordToken имеет поле KeyWords, где хранит enum значение одного из ключевых слов. CConstToken имеет поле CVariantPtr, предоставляющее умный указатель на объект класса CVariant, хранящий значение константы.

***CVariant.*** Абстрактный класс. Наследуемые классы имеют поля типа VariantType - enum значение, характеризующее тип константы (string/real/integer/boolean), а также методы GetVaraintType(), возвращающее тип константы и ToString(), преобразующий константу в строку.

***CIntVariant/CRealVariant/CBooleanVariant/CStringVariant.*** Наследуются от CVariant. Представляются как 4 разных вида констант - целочисленные, вещественные, строковые, логические. В каждом классе есть метод GetValue(), возвращающий значение константы, а также переопределенный метод ToString(), возвращающий значение константы в виде строки.



Умные указатели. В качестве указателей на объекты СToken и CVariant использовались умные указатели. Благодаря умным указателям возможно избежать утечек памяти в программе и обеспечить безопасное ее использование.

Тестирование лексического анализатора.

Синтаксический анализатор.

Для реализации синтаксического анализатора необходимо изначально определить все используемые конструкции языка Pascal. Приведем эти конструкции ниже в БНФ нотации.

1. <программа>::=**program** <имя>(<имя файла>{,<имя файла>});<блок>.
2. <имя файла>::=<имя>
3. <имя>::=<буква>{<буква>|<цифра>}
4. <блок>::=<раздел типов><раздел переменных><раздел процедур и функций><раздел операторов>
5. <раздел типов>::=<пусто>|**type** <определение типа>; {<определение типа>;}
6. <определение типа>::=<имя>=<тип>
7. <тип>::=<простой тип>
8. <простой тип>::=<перечислимый тип>|<ограниченный тип>|<имя типа>
9. <перечислимый тип>::=(<имя>{,<имя>})
10. <ограниченный тип>::=<константа>..<константа>
11. <имя типа>::=<имя>
12. <раздел переменных>::= **var** <описание однотипных переменных>;{<описание однотипных переменных>;}|<пусто>
13. <описание однотипных переменных>::=<имя>{,<имя>}:<тип>
14. <раздел процедур и функций>::={<описание процедуры или функции>;}
15. <описание процедуры или функции>::=<описание процедуры>
16. <описание процедуры>::=<заголовок процедуры><блок>
17. <заголовок процедуры>::= **procedure** <имя>;| **procedure** <имя> (<раздел формальных параметров>{;<раздел формальных параметров>});
18. <раздел формальных параметров>::=<группа параметров>| **var** <группа параметров>| **procedure** <имя>{,<имя>}
19. <группа параметров>::=<имя>{,<имя>}:<имя типа>
20. <раздел операторов>::=<составной оператор>
21. <оператор>::=<непомеченный оператор>
22. <непомеченный оператор>::=<простой оператор>|<сложный оператор>
23. <простой оператор>::=<оператор присваивания>|<оператор процедуры>
24. <оператор присваивания>::=<переменная>:=<выражение>
25. <переменная>::=<полная переменная>| <компонента переменной>|<указанная переменная>
26. <полная переменная>::=<имя переменной>
27. <имя переменной>::=<имя>
28. <выражение>::=<простое выражение>|<простое выражение> <операция отношения><простое выражение>
29. <операция отношения>::==|<>|<|<=|>=|>|**in**
30. <простое выражение>::=<знак><слагаемое>{<аддитивная операция><слагаемое>}
31. <аддитивная операция>::=+|-|**or**
32. <слагаемое>::=<множитель>{<мультипликативная операция> <множитель>}
33. <мультипликативная операция>::=\*|/|**div**|**mod**|**and**
34. <множитель>::=<переменная>|<константа без знака>| (<выражение>)| not <множитель>
35. <константа без знака>::=<число без знака>|<строка>|<имя константы>|nil
36. <оператор процедуры>::=<имя процедуры>|<имя процедуры>(<фактический параметр>{,<фактический параметр>})
37. <имя процедуры>::=<имя>
38. <фактический параметр>::=<выражение>|<переменная>|<имя процедуры>
39. <пустой оператор>::=<пусто>
40. <пусто>::=
41. <сложный оператор>::=<составной оператор>|<выбирающий оператор>|<оператор цикла>|<оператор присоединения>
42. <составной оператор>::= **begin** <оператор>{;<оператор>} **end**
43. <выбирающий оператор>::=<условный оператор>| <оператор варианта>
44. <условный оператор>::= **if** <выражение> **then** <оператор>| **if** <выражение> **then** <оператор> **else** <оператор>
45. <оператор варианта>::= **case** <выражение> **of** <элемент списка вариантов>{;<элемент списка вариантов>} **end**
46. <элемент списка вариантов>::=<список меток варианта>: <оператор>|<пусто>
47. <список меток варианта>::=<метка варианта>{,<метка варианта>}
48. <оператор цикла>::=<цикл с предусловием>|<цикл с постусловием>|<цикл с параметром>
49. <оператор цикла>::=<цикл с предусловием>
50. <цикл с предусловием>::= while <выражение> do <оператор>