БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

Факультет НиДО

Специальность ИиТП

Практическая работа № 2

по дисциплине «Методы трансляции»

Выполнил студент: Дегтярев А.А.

группа 393551

Зачетная книжка № 902021-26

Минск 2017

### Вариант 7

### ИПР №2

**Цель:** Нисходящие и восходящие методы синтаксического анализа**.**

**Общая постановка задачи:** Создать модель синтаксического анализатора для анализа выражений небольшого языка программирования типа Паскаль, С.

**Методические указания**:

Разработать лексический анализатор в виде класса. Требуется обратить внимание на набор методов класса (интерфейс). От того, насколько удачно будет проработан интерфейс класса, в дальнейшем от этого будут зависеть функциональные возможности лексического анализатора и удобство работы с ним. Основная функциональность, которую должен обеспечить класс синтаксического анализатора:

открытие файла с данными;

чтение строки из файла;

разбиение строки на числовые, символьные или строковые лексемы;

возможность задания символов-разделителей для чтения текстовых лексем;

возможность пропуска пробелов и символов-разделителей при лексическом анализе;

распознавание идентификаторов и ключевых слов;

распознавание констант;

распознавание разделителей и знаков операций.

Для тестирования созданного класса написать небольшое приложение.

В данном приложении класс лексического анализатора использовать для получения информации об объектах и их параметрах из файла данных.

*выполнить анализ следующей исходной программы:*

int m,f=100;

begin

for j := 2 to 9 do

begin

m := 7 ;

f := f + 1 ;

end

Порядок использования программы:



-f указывает файл из которого производится чтение

-l указывает на использования лексического анализатора

-v позволяет выводить промежуточные логи

-h показывает помощь по вводным параметрам

Ниже представлен файл, лексический анализ которого я выполнял:

|  |
| --- |
| int m,f:=100;  begin  for j:=2 to 9 do  begin  m:= 7 ;  f :=f +1 ;  end |

Для анализа я использовал встроенный в D обработчик регулярных выражений, который позволяет разделить читаемые строки сперва по переносам строк, а далее по указанным разделителям. (в данном случае это были пробельные символы, операторы, запятые;

Метод возвращает поток токенов, содержащие текст и тип, который потом может быть обработан анализатором грамматики и семантики и впоследствии представлен в виде исполняемого байт-кода или программы на транслируемом языке.

Ниже представлен скриншот работы программы:



Исходный код класса

|  |
| --- |
| //////////////////////  //CHARACTER SET:  //Any unicode character  //////////////////////  public class Token{  static string[string] lexerRegExp;  static string[] lexerKeyOrder = [  "OPERATOR2","OPERATOR1","COMMA","KEYWORD","SEMICOLON","TYPE","INTEGERLITERAL","ID"  ];  public static string splitterRegExp;  public string content;  public string type;  this(string input){  content = input;  //maximal munch technic  foreach(tokenType; lexerKeyOrder){  auto pattern = regex(lexerRegExp[tokenType]);  if(!matchFirst(content,pattern).empty){  type = tokenType;  break;  }  }  }  static this(){  lexerRegExp = [  "OPERATOR2" : r":=|==|<|>|!=|=|\+|-",  "OPERATOR1" : r"\+\+|--", /\*"\*/  "COMMA" : ",",  "KEYWORD" : "begin|end|for|to|do",  "SEMICOLON" : ";",  "TYPE" : "int|float",  "INTEGERLITERAL" : "[0-9]+",  "ID" : "[a-zA-Z][a-zA-Z0-9]\*"  ];  splitterRegExp =  "\u0020+|\u0009+|\u000B+|\u000C+|" ~ //WHITESPACE  "\u0000+|\u001A+|" ~ //EOF  "\u000D+|\u000A+|\u000D \u000A|\u2028+\u2029+|" ~ //EOL  lexerRegExp["OPERATOR2"] ~ "|" ~  lexerRegExp["OPERATOR1"] ~ "|" ~  lexerRegExp["COMMA"] ~ "|" ~  lexerRegExp["SEMICOLON"];  }  public override string toString(){  return "token: " ~ content ~ " is: " ~ type;  }  }  public Token[] LexicalAnalysis(){  auto inputFile = File(inputFilePath,"r");  auto splitPattern = regex(Token.splitterRegExp);  auto nonemptyChunkPattern = regex(r"\S");  /\*  "<-linterfix  \*/  string line;  Token[] tokenChain;  if(verbose) writeln("--Start Reading File--");  while(!inputFile.eof()){  line = inputFile.readln();  if(verbose) writeln("line: ",line);  foreach(chunk; line.splitter!(Yes.keepSeparators)(splitPattern)){  if(!matchFirst(chunk,nonemptyChunkPattern).empty){  tokenChain ~= new Token(chunk);  if(verbose) writeln(tokenChain[$-1]);  }  }  }  if(verbose) writeln("--Ended Reading File--");  return tokenChain;  } |