COMPILER PROJECT - 9
BCSE-3rd Year

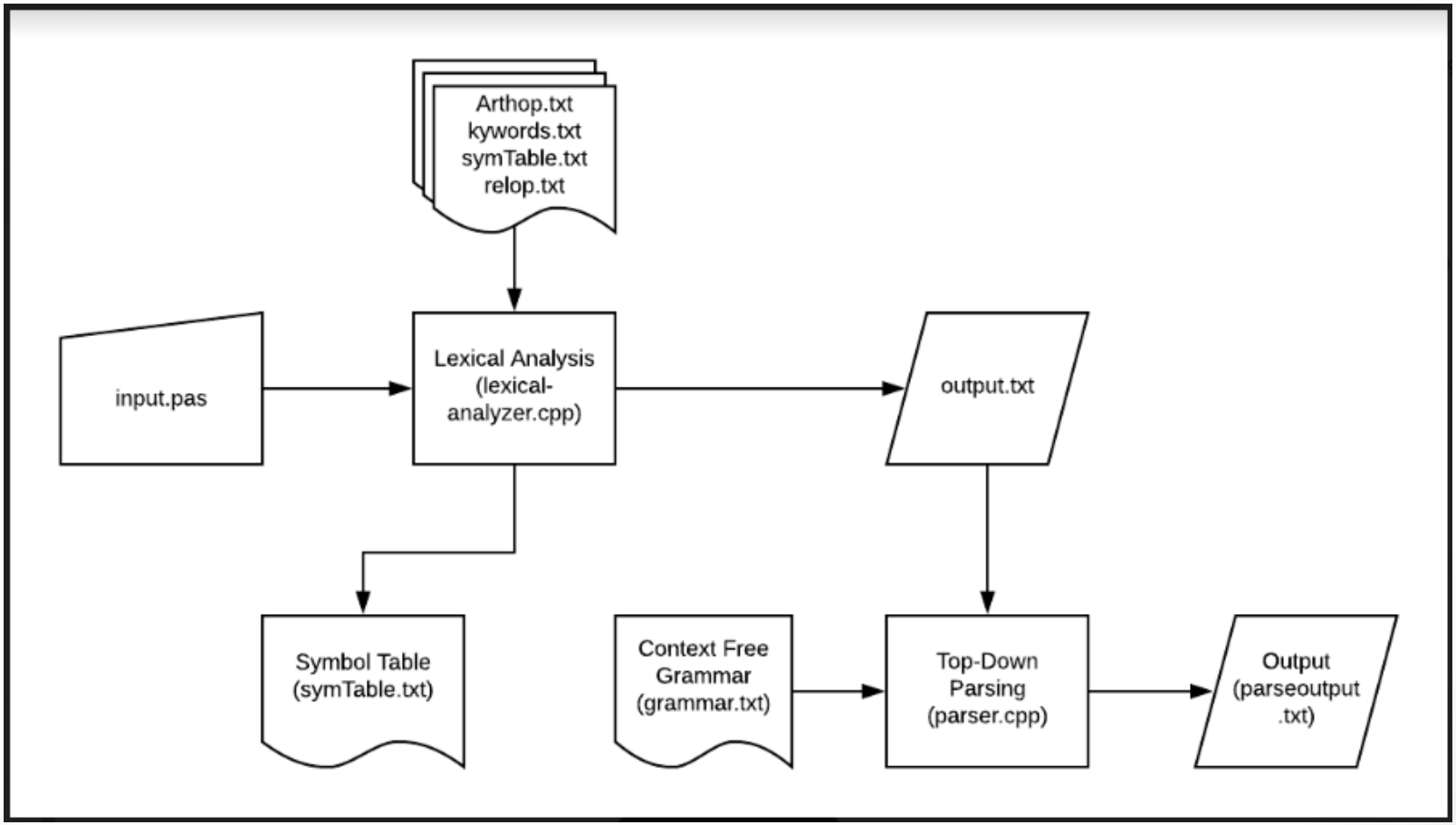
Sourav Dutta - 001610501076

Abhijeet Raj - 001610501080

Akash Mandal - 001610501084

Shreya - 001610501085

FLOWCHART


Arthop.txt
kywords.txt
symTable.txt
relop.txt
input.pas
Lexical Analysis
(lexical-
analyzer.cpp)
output.txt
Symbol Table
(symTable.txt)
Context Free
Grammar
(grammar.txt)
Top-Down
Parsing
(parser.cpp)
Output
(parseoutput
.txt)

PART-1: CFG FOR PASCAL LANGUAGE

```
P -> program ProgramName ; BlockBody
ProgramName -> id
BlockBody -> LibraryDef ConstantDef VariableDef FuncDef begin MainBody end .
LibraryDef -> epsilon
LibraryDef -> uses id LibraryDef'
LibraryDef' -> , id LibraryDef'
LibraryDef' -> ;
ConstantDef -> const Const_Def
ConstantDef -> epsilon
Const_Def -> id = Numeral ; C
C -> Const_Def
C -> epsilon
VariableDef -> var Var_Def V
VariableDef -> epsilon
V -> epsilon
V -> Var_Def V
Var_Def -> VarList : Id_Type ;
VarList -> id VarList'
VarList' -> , id VarList'
VarList' -> epsilon
Id_Type -> integer
Id_Type -> real
FuncDef -> epsilon
FuncDef -> function id ( FuncArgmt ) : Id_Type Local_Decl begin Statement end ; FuncDef
FuncArgmt -> epsilon
FuncArgmt -> VarList : Id_Type X
X -> epsilon
X -> , FuncArgmt
FuncArgmt -> epsilon
Local_Decl -> epsilon
Local_Decl -> VariableDef
Statement -> epsilon
Statement -> Input_Stmt ; Statement
Statement -> Output_Stmt ; Statement
Statement -> IfElse_Stmt
Statement -> Cond_Stmt
```

PART-1 (Continued…)

```
Statement -> Assign_Stmt ; Statement
Input_Stmt -> get id
Output_Stmt -> put id
Cond_Stmt -> ( Condition ) ? Statement : Statement
Assign_Stmt -> id = expression
expression -> expression addsubop term
expression -> term
term -> term * factor
term -> factor
factor -> ( expression )
factor -> id
factor -> Numeral
addsubop -> +
addsubop -> -
IfElse_Stmt -> if ( Condition ) then begin Statement end ElsePart Statement
ElsePart -> else begin Statement end
ElsePart -> epsilon
Condition -> expression relop expression
relop -> >
relop -> <
relop -> >=
relop -> <=
MainBody -> epsilon
MainBody -> Statement
```

PART-2: LEXICAL ANALYSER

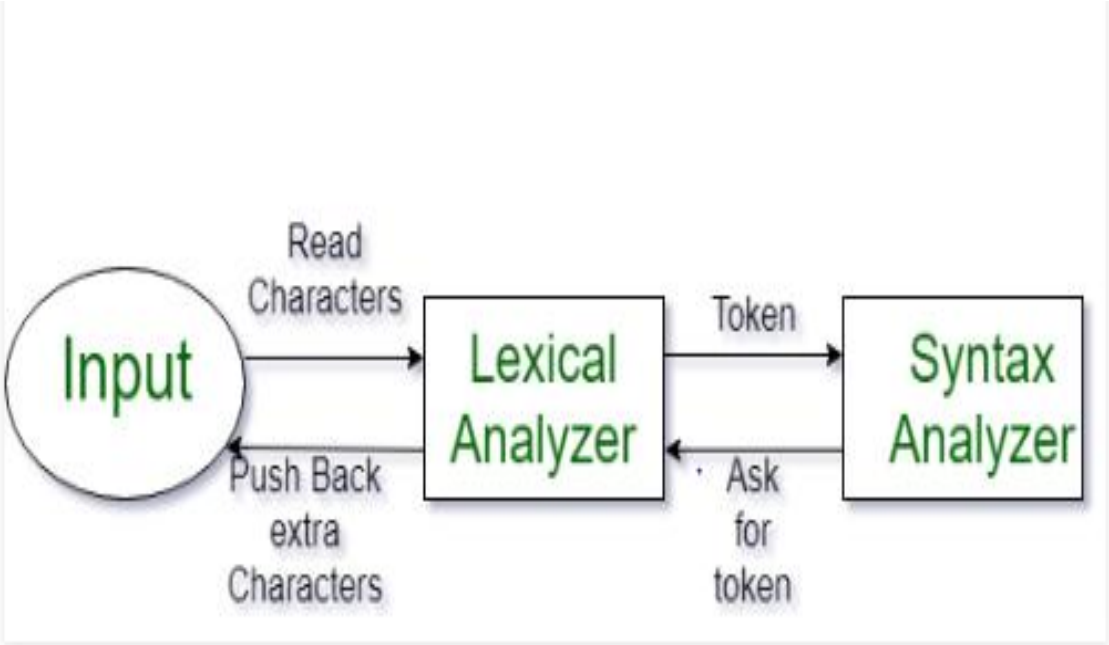
- **GLOBAL DATA STRUCTURE USED:**

```
#define p pair<string,string>
#define pp pair<p,int>
#define ppp pair<string,pp>
#define sett set<string>

sett kwords,relOp;
set<char> arthOp;
map<string,int> declare;
vector<ppp> symtable;
sett errormsg;
vector<string> line;
```

- **sett kwords, relOp;**
  - All the keywords and relational operators of Pascal language are stored in **kwords** and **relOp** respectively.
- **set<char> arthOp;**
  - Arthematic operators are stored in this set containing character.
- **Map<string, int> declare;**
  - This contains the list of variables that are defined in the input program.
- **Vector<ppp> symtable;**
  - This table consists of all the tokens present in the input program.
- **Sett errormsg;**
  - This includes messages for various syntax errors that could be present in the program.

- Lexical Analysis is the first phase of compiler also known as scanner.
- The output is a sequence of tokens that is sent to the parser for syntax analysis.



What the lexical analyser does?

- ✓ Tokenization that means it divides the program into valid tokens.
- ✓ It removes white space characters.
- ✓ Following error checkings have been taken care of:
  - Redeclaration of a variable
  - Using of undeclared variable
  - Invalid identifier name (starts with a numeric value)

PART 3:TOP-DOWN PARSER LL(1)

- **GLOBAL DATA STRUCTURE USED:**
  - **map<string, set<vector<string> > > productions**
    -- Stores the grammar of every non-terminals (key).
  - **map<int, vector<string> > getprodleft**
    -- Used to retain the ordering of the grammars. (since the above map stores it in lexicographical order)
  - **string startsymbol**
    -- To store Starting non-terminal symbol
  - **set<string> nonterminals, terminals**
    -- To store all the non-terminals and terminals.
  - **map<string, set<string> > first**
    -- Stores the first set (terminals) for every terminals and non-terminals (key).
  - **map<string, set<string>> follow**
    -- Stores the follow set (terminals) for every non-terminals (key).
  - **map<string,vector<vector<string>>> parsetable**
    -- Corresponding to each non-terminals (key), it stores the action of every terminals and $. Can be thought as a 2D-array of strings.

PART 3:TOP-DOWN PARSER LL(1)

- In the syntax analysis phase, a compiler verifies whether or not the tokens generated by the lexical analyser are grouped according to the syntactic rules of the language. This is done by a parser. The parser obtains a string of tokens from the lexical analyser and verifies that the string can be the grammar for the source language.
- **buildProductions**(string filename): This function takes as input a CFG defined for a language and stores it into a map, so that, for parsing, we need not refer the input file again and again.
- **removeLeftRecursion():** This function removes left recursion from the grammar. As we know, top-down parsing method cannot handle left recursive grammars. A grammar is left recursive if it has a non terminal (variable) S such that their is a derivation
  S -> Sα | β
  where α ∈(V+T)* and β ∈(V+T)* (sequence of terminals and non terminals that do not start with S)
  Due to the presence of left recursion some top down parsers enter into infinite loop so we have to eliminate left recursion.

- The algorithm used for left recursion removal is as follows:

- Arrange non-terminals in some order: $A_1 \ldots A_n$
- **for** i **from** 1 **to** n **do** {
  - **for** j **from** 1 **to** i-1 **do** {
    replace each production
    $A_i \rightarrow A_j \gamma$
    by
    $A_i \rightarrow \alpha_1 \gamma \mid \ldots \mid \alpha_k \gamma$
    where $A_j \rightarrow \alpha_1 \mid \ldots \mid \alpha_k$
  }
  - eliminate immediate left-recursions among A_i productions
}

- **printProductions():** This function is used for printing the production rules. Uses the “getprodleft” to retain the ordering of grammars as given.

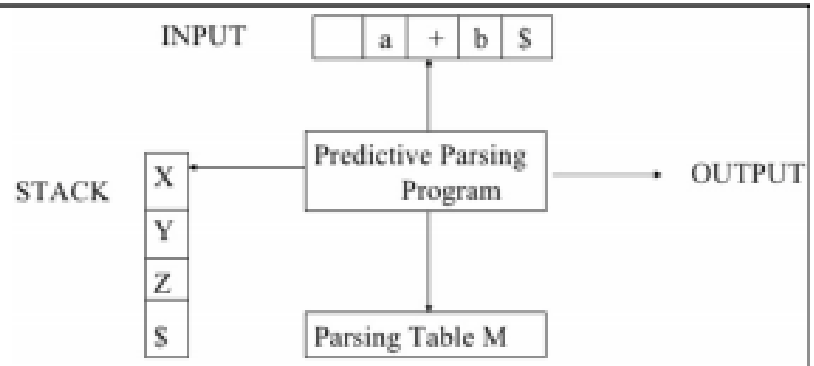
- **computeFirst():** This function computes First set for each terminal and non terminal present in the grammar. **First(X)** for a grammar symbol X is the set of terminals that begin the strings derivable from X.
- **Rules to compute First set:**
  - If x is a terminal, then FIRST(x) = { ‘x’ }
  - If x-> Є, is a production rule, then add Є to FIRST(x).
  - If X->Y1 Y2 Y3….Yn is a production,
    - FIRST(X) = FIRST(Y1)
    - If FIRST(Y1) contains Є then FIRST(X) = { FIRST(Y1) – Є } U { FIRST(Y2) }
    - If FIRST (Yi) contains Є for all i = 1 to n, then add Є to FIRST(X).

- **computeFollow():** This function computes Follow set for every non terminal present in the grammar. **Follow(X)** to be the set of terminals that can appear immediately to the right of Non-Terminal X in some sentential form.

- **buildParsingTable():** This function is used for constructing LL(1) parsing table.
- **Rules for calculating the Parsing table:**
  - ✓ For each production A→ α of thr grammar, do the following:
    - ➢ For each terminal a, in first of FIRST(α), add A→ α to M[A, α].
    - ➢ If ε is in FIRST(α), add A→ α to M[A, α] for each terminal b in the FOLLOW(A). If ε is in FIRST(α) and $ is in FOLLOW(A), add A→ α to M[A, $].
  - ✓ Make each undefined entry of M "error".
- **parseTokens("output.txt"):** This function takes as argument output.txt file which contains the stream of tokens, we get after the lexical analysis of Pascal program.

- **Algorithm for table driven predictive parsing:**

Table-Driven Parsing



- An input buffer
  - Contains the input string
  - The string can be followed by $, an end marker to indicate the end of the string
- A stack
  - Contains symbols with $ on the bottom, with the start symbol initially on the top
- A parsing table (2-dimensional array M[A, a])
- An output stream (production rules applied for derivation)

Continued…..

Input: a string w, a parsing table M for grammar G

Output: if w is in L(G), a leftmost derivation of w; otherwise, an error indication

Method:

```
set ip to point to the first symbol of w$
repeat
    let X be the top stack symbol and a the symbol pointed to by ip;
    if X is a terminal or $, then
        if X = a then
            pop X from the stack and advance ip
        else error()
    else                                /* X is a non-terminal */
        if M[X, a] = X->Y1Y2…Yk, then
            pop X from the stack
            push Yk, …, Y2, Y1 on to the stack
            output the production X->Y1Y2…Yk
        end
        else error()
until X = $
```

OUTPUT

```
C:\Users\SOURAV\Desktop\compiler-design\project>a
Input the filename which contains grammar : grammar.txt
After removing left recursion, The production rules are:
P -> program ProgramName ; BlockBody
ProgramName -> id
BlockBody -> LibraryDef ConstantDef VariableDef FuncDef begin MainBody end .
LibraryDef -> epsilon  | uses id LibraryDef'
LibraryDef' -> , id LibraryDef'  | ;
ConstantDef -> const Const_Def  | epsilon
Const_Def -> id = Numeral ; C
C -> epsilon  | id = Numeral ; C
VariableDef -> epsilon  | var Var_Def V
V -> Var_Def V  | epsilon
Var_Def -> VarList : Id_Type ;
VarList -> id VarList'
VarList' -> , id VarList'  | epsilon
Id_Type -> integer  | real
FuncDef -> epsilon  | function id ( FuncArgmt ) : Id_Type Local_Decl begin Statement end ; FuncDef
FuncArgmt -> epsilon  | id VarList' : Id_Type X
X -> , FuncArgmt  | epsilon
Local_Decl -> epsilon  | var Var_Def V
Statement -> Assign_Stmt ; Statement  | Cond_Stmt  | IfElse_Stmt  | Input_Stmt ; Statement  | Output_Stmt ; Statement  | epsilon
Input_Stmt -> get id
Output_Stmt -> put id
Cond_Stmt -> ( Condition ) ? Statement : Statement
Assign_Stmt -> id = expression
expression -> term expression'
expression' -> addsubop term expression'  | epsilon
term -> factor term'
term' -> * factor term'  | epsilon
factor -> ( expression )  | Numeral  | id
addsubop -> +  | -
IfElse_Stmt -> if ( Condition ) then begin Statement end ElsePart Statement
ElsePart -> else begin Statement end  | epsilon
Condition -> ( expression ) term' expression' relop expression  | Numeral term' expression' relop expression  | id term' expression' relop expression
relop -> <  | <=  | >  | >=
MainBody -> ( Condition ) ? Statement : Statement  | epsilon  | get id ; Statement  | id = expression ; Statement  | if ( Condition ) then begin Statement end ElsePart Statement  | put id ;
Statement

Non Terminals are: Assign_Stmt BlockBody C Cond_Stmt Condition Const_Def ConstantDef ElsePart FuncArgmt FuncDef Id_Type IfElse_Stmt Input_Stmt LibraryDef LibraryDef' Local_Decl MainBody
Output_Stmt P ProgramName Statement V VarList VarList' Var_Def VariableDef X addsubop expression expression' factor relop term term'

Terminals are: ( ) * + , - . : ; < <= = > >= ? Numeral begin const else end function get id if integer program put real then uses var
```

Symbol	FIRST Set
(	{ (}
)	{) }
*	{ * }
+	{ + }
,	{ , }
-	{ - }
.	{ . }
:	{ : }
;	{ ; }
<	{ < }
<=	{ <= }
=	{ = }
>	{ > }
>=	{ >= }
?	{ ? }
Assign_Stmt	{ id }
BlockBody	{ begin , const , function , uses , var }
C	{ epsilon , id }
Cond_Stmt	{ (}
Condition	{ (, Numeral , id }
Const_Def	{ id }
ConstantDef	{ const , epsilon }
ElsePart	{ else , epsilon }
FuncArgmt	{ epsilon , id }
FuncDef	{ epsilon , function }
Id_Type	{ integer , real }
IfElse_Stmt	{ if }
Input_Stmt	{ get }
LibraryDef	{ epsilon , uses }
LibraryDef'	{ , , ; }
Local_Decl	{ epsilon , var }
MainBody	{ (, epsilon , get , id , if , put }
Numeral	{ Numeral }
Output_Stmt	{ put }
P	{ program }
ProgramName	{ id }
Statement	{ (, epsilon , get , id , if , put }
V	{ epsilon , id }
VarList	{ id }
VarList'	{ , , epsilon }
Var_Def	{ id }
VariableDef	{ epsilon , var }
X	{ , , epsilon }
addsubop	{ + , - }
begin	{ begin }
const	{ const }
else	{ else }
end	{ end }
expression	{ (, Numeral , id }
expression'	{ + , - , epsilon }
factor	{ (, Numeral , id }
function	{ function }
get	{ get }
id	{ id }
if	{ if }
integer	{ integer }
program	{ program }
put	{ put }
real	{ real }
relop	{ < , <= , > , >= }
term	{ (, Numeral , id }
term'	{ * , epsilon }
then	{ then }
uses	{ uses }
var	{ var }

```
---------------------------------
Symbol          FOLLOW Set
---------------------------------
Assign_Stmt             { ; }
BlockBody               { $ }
C               { begin , function , var }
Cond_Stmt               { : , end }
Condition               { ) }
Const_Def               { begin , function , var }
ConstantDef             { begin , function , var }
ElsePart                { ( , : , end , get , id , if , put }
FuncArgmt               { ) }
FuncDef                 { begin }
Id_Type                 { ) , , , ; , begin , var }
IfElse_Stmt             { : , end }
Input_Stmt              { ; }
LibraryDef              { begin , const , function , var }
LibraryDef'             { begin , const , function , var }
Local_Decl              { begin }
MainBody                { end }
Output_Stmt             { ; }
P               { $ }
ProgramName             { ; }
Statement               { : , end }
V               { begin , function }
VarList                 { : }
VarList'                { : }
Var_Def                 { begin , function , id }
VariableDef             { begin , function }
X               { ) }
addsubop                { ( , Numeral , id }
expression              { ) , ; }
expression'             { ) , ; , < , <= , > , >= }
factor          { ) , * , + , - , ; , < , <= , > , >= }
relop           { ( , Numeral , id }
term            { ) , + , - , ; , < , <= , > , >= }
term'           { ) , + , - , ; , < , <= , > , >= }
---------------------------------
```

Input the filename which contains tokens to be parsed : output.txt
Token number: 1, Token: program, Message: Parsed successfully
Token number: 2, Token: id, Message: Parsed successfully
Token number: 3, Token: ;, Message: Parsed successfully
Token number: 4, Token: uses, Message: Parsed successfully
Token number: 5, Token: id, Message: Parsed successfully
Token number: 6, Token: ,, Message: Parsed successfully
Token number: 7, Token: id, Message: Parsed successfully
Token number: 8, Token: ;, Message: Parsed successfully
Token number: 9, Token: const, Message: Parsed successfully
Token number: 10, Token: id, Message: Parsed successfully
Token number: 11, Token: =, Message: Parsed successfully
Token number: 12, Token: Numeral, Message: Parsed successfully
Token number: 13, Token: ;, Message: Parsed successfully
Token number: 14, Token: id, Message: Parsed successfully
Token number: 15, Token: =, Message: Parsed successfully
Token number: 16, Token: Numeral, Message: Parsed successfully
Token number: 17, Token: ;, Message: Parsed successfully
Token number: 18, Token: var, Message: Parsed successfully
Token number: 19, Token: id, Message: Parsed successfully
Token number: 20, Token: ,, Message: Parsed successfully
Token number: 21, Token: id, Message: Parsed successfully
Token number: 22, Token: ,, Message: Parsed successfully
Token number: 23, Token: id, Message: Parsed successfully
Token number: 24, Token: :, Message: Parsed successfully
Token number: 25, Token: integer, Message: Parsed successfully
Token number: 26, Token: ;, Message: Parsed successfully
Token number: 27, Token: id, Message: Parsed successfully
Token number: 28, Token: ,, Message: Parsed successfully
Token number: 29, Token: id, Message: Parsed successfully
Token number: 30, Token: :, Message: Parsed successfully
Token number: 31, Token: real, Message: Parsed successfully
Token number: 32, Token: ;, Message: Parsed successfully
Token number: 33, Token: function, Message: Parsed successfully
Token number: 34, Token: id, Message: Parsed successfully
Token number: 35, Token: (, Message: Parsed successfully
Token number: 36, Token: id, Message: Parsed successfully
Token number: 37, Token: ,, Message: Parsed successfully
Token number: 38, Token: id, Message: Parsed successfully
Token number: 39, Token: :, Message: Parsed successfully
Token number: 40, Token: real, Message: Parsed successfully
Token number: 41, Token: ,, Message: Parsed successfully
Token number: 42, Token: id, Message: Parsed successfully
Token number: 43, Token: :, Message: Parsed successfully
Token number: 44, Token: integer, Message: Parsed successfully
Token number: 45, Token:), Message: Parsed successfully
Token number: 46, Token: :, Message: Parsed successfully
Token number: 47, Token: real, Message: Parsed successfully
Token number: 48, Token: var, Message: Parsed successfully
Token number: 49, Token: id, Message: Parsed successfully
Token number: 50, Token: ,, Message: Parsed successfully
Token number: 51, Token: id, Message: Parsed successfully
Token number: 52, Token: :, Message: Parsed successfully
Token number: 53, Token: integer, Message: Parsed successfully
Token number: 54, Token: ;, Message: Parsed successfully
Token number: 55, Token: id, Message: Parsed successfully
Token number: 56, Token: :, Message: Parsed successfully
Token number: 57, Token: real, Message: Parsed successfully
Token number: 58, Token: ;, Message: Parsed successfully
Token number: 59, Token: begin, Message: Parsed successfully
Token number: 60, Token: id, Message: Parsed successfully
Token number: 61, Token: =, Message: Parsed successfully
Token number: 62, Token: Numeral, Message: Parsed successfully
Token number: 63, Token: *, Message: Parsed successfully
Token number: 64, Token: id, Message: Parsed successfully
Token number: 65, Token: ;, Message: Parsed successfully
Token number: 66, Token: put, Message: Parsed successfully
Token number: 67, Token: id, Message: Parsed successfully
Token number: 68, Token: ;, Message: Parsed successfully
Token number: 69, Token: end, Message: Parsed successfully
Token number: 70, Token: ;, Message: Parsed successfully
Token number: 71, Token: begin, Message: Parsed successfully
Token number: 72, Token: get, Message: Parsed successfully
Token number: 73, Token: id, Message: Parsed successfully
Token number: 74, Token: ;, Message: Parsed successfully
Token number: 75, Token: put, Message: Parsed successfully
Token number: 76, Token: id, Message: Parsed successfully
Token number: 77, Token: ;, Message: Parsed successfully
Token number: 78, Token: id, Message: Parsed successfully
Token number: 79, Token: =, Message: Parsed successfully
Token number: 80, Token: id, Message: Parsed successfully
Token number: 81, Token: *, Message: Parsed successfully
Token number: 82, Token: (, Message: Parsed successfully
Token number: 83, Token: id, Message: Parsed successfully
Token number: 84, Token: *, Message: Parsed successfully
Token number: 85, Token: (, Message: Parsed successfully
Token number: 86, Token: id, Message: Parsed successfully
Token number: 87, Token: -, Message: Parsed successfully
Token number: 88, Token: Numeral, Message: Parsed successfully
Token number: 89, Token:), Message: Parsed successfully
Token number: 90, Token:), Message: Parsed successfully
Token number: 91, Token: ;, Message: Parsed successfully
Token number: 92, Token: if, Message: Parsed successfully
Token number: 93, Token: (, Message: Parsed successfully
Token number: 94, Token: id, Message: Parsed successfully
Token number: 95, Token: >, Message: Parsed successfully
Token number: 96, Token: id, Message: Parsed successfully
Token number: 97, Token:), Message: Parsed successfully
Token number: 98, Token: then, Message: Parsed successfully
Token number: 99, Token: begin, Message: Parsed successfully
Token number: 100, Token: id, Message: Parsed successfully
Token number: 101, Token: =, Message: Parsed successfully
Token number: 102, Token: id, Message: Parsed successfully
Token number: 103, Token: ;, Message: Parsed successfully
Token number: 104, Token: get, Message: Parsed successfully
Token number: 105, Token: id, Message: Parsed successfully
Token number: 106, Token: ;, Message: Parsed successfully
Token number: 107, Token: end, Message: Parsed successfully
Token number: 108, Token: else, Message: Parsed successfully
Token number: 109, Token: begin, Message: Parsed successfully
Token number: 110, Token: id, Message: Parsed successfully
Token number: 111, Token: =, Message: Parsed successfully
Token number: 112, Token: id, Message: Parsed successfully
Token number: 113, Token: ;, Message: Parsed successfully
Token number: 114, Token: if, Message: Parsed successfully
Token number: 115, Token: (, Message: Parsed successfully
Token number: 116, Token: id, Message: Parsed successfully
Token number: 117, Token: <=, Message: Parsed successfully
Token number: 118, Token: Numeral, Message: Parsed successfully
Token number: 119, Token:), Message: Parsed successfully
Token number: 120, Token: then, Message: Parsed successfully
Token number: 121, Token: begin, Message: Parsed successfully
Token number: 122, Token: get, Message: Parsed successfully
Token number: 123, Token: id, Message: Parsed successfully
Token number: 124, Token: ;, Message: Parsed successfully
Token number: 125, Token: put, Message: Parsed successfully
Token number: 126, Token: id, Message: Parsed successfully
Token number: 127, Token: ;, Message: Parsed successfully
Token number: 128, Token: end, Message: Parsed successfully
Token number: 129, Token: end, Message: Parsed successfully
Token number: 130, Token: (, Message: Parsed successfully
Token number: 131, Token: id, Message: Parsed successfully
Token number: 132, Token: >, Message: Parsed successfully
Token number: 133, Token: id, Message: Parsed successfully
Token number: 134, Token:), Message: Parsed successfully
Token number: 135, Token: ?, Message: Parsed successfully
Token number: 136, Token: id, Message: Parsed successfully
Token number: 137, Token: =, Message: Parsed successfully
Token number: 138, Token: id, Message: Parsed successfully
Token number: 139, Token: ;, Message: Parsed successfully
Token number: 140, Token: :, Message: Parsed successfully
Token number: 141, Token: id, Message: Parsed successfully
Token number: 142, Token: =, Message: Parsed successfully
Token number: 143, Token: id, Message: Parsed successfully
Token number: 144, Token: ;, Message: Parsed successfully
Token number: 145, Token: put, Message: Parsed successfully
Token number: 146, Token: id, Message: Parsed successfully
Token number: 147, Token: ;, Message: Parsed successfully
Token number: 148, Token: end, Message: Parsed successfully
Token number: 149, Token: ., Message: Parsed successfully
Token number: 150, Token: $, Message: Parsed successfully
- -
RESULT : Parsed successfully! No Error is found.

THANK YOU