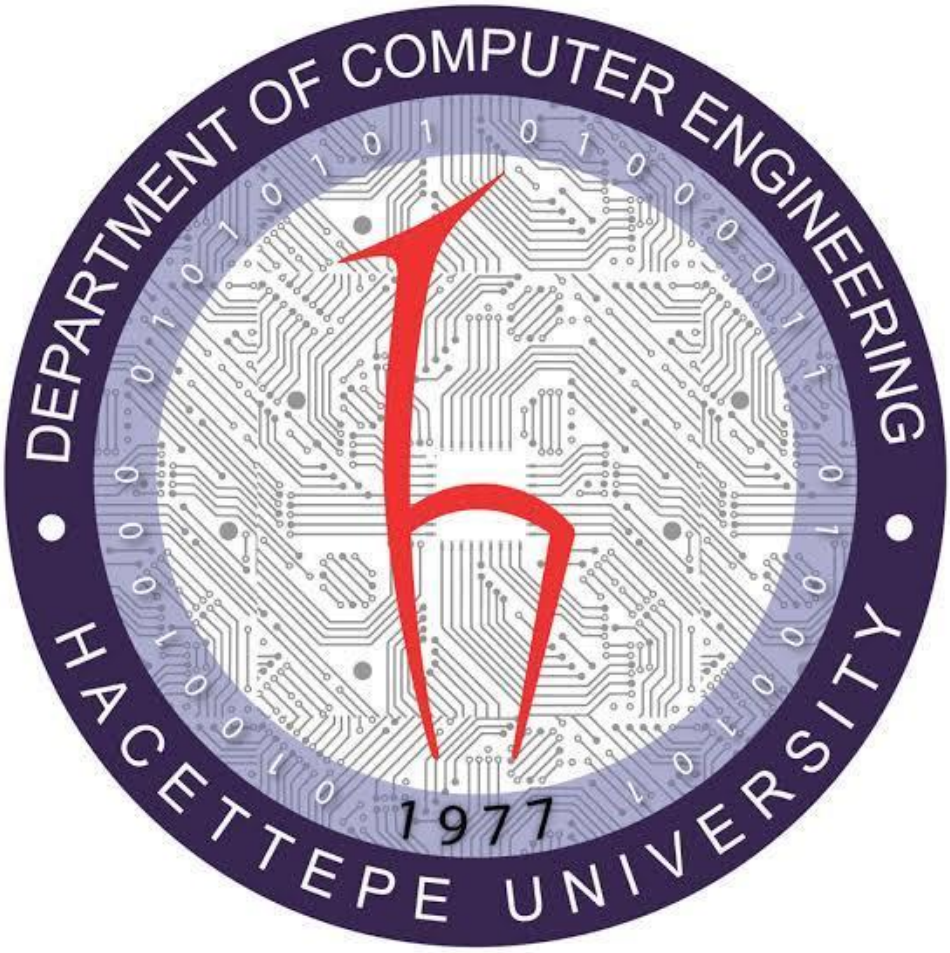


PROGRAMMING LANGUAGES PROJECT

Submission:18.11.2019

Due:17.12.2019

Project Group:

Ali Kayadibi - 21727432

Muhammed Said Kaya - 21627428

İsmet Okatar – 21727542

Contents

BNF DESCRIPTION OF THE LANGUAGE

```
%token OR_OPERATOR AND_OPERATOR NOT_OPERATOR EQUAL_OPERATOR
LESS_OPERATOR LESS_OR_EQUAL_OPERATOR GREATER_OPERATOR
GREATER_OR_EQUAL_OPERATOR LEFT_PARENTHESIS
%token RIGHT_PARENTHESIS LEFT_CURLY RIGHT_CURLY LEFT_BRACKET
RIGHT_BRACKET PLUS MINUS MULTIPLICATION DIVISION EXPONENTIAL MOD
TRUE FALSE FOR WHILE IF ELSE PUSH
%token PULL INT_TYPE FLOAT_TYPE STRING_TYPE VOID_TYPE BOOLEAN_TYPE
INTEGER FLOAT STRING NEW_LINE INVALID ASSIGNMENT_OPERATOR PROGRAM
COMMA RETURN COMMENT SEMICOLON
%token GRAPH_TYPE QUOTE POINT ARRAY

%start program

%%

types : INT_TYPE | STRING_TYPE | BOOLEAN_TYPE | FLOAT_TYPE |
GRAPH_TYPE
                ;
primitive_types : INTEGER| FLOAT| STRING | TRUE | FALSE | QUOTE
STRING QUOTE          ;
operators : EQUAL_OPERATOR | NOT_OPERATOR | LESS_OPERATOR |
LESS_OR_EQUAL_OPERATOR | GREATER_OPERATOR |
GREATER_OR_EQUAL_OPERATOR
                ;
array_declaration : types ARRAY LEFT_BRACKET INTEGER RIGHT_BRACKET

program : VOID_TYPE PROGRAM LEFT_PARENTHESIS RIGHT_PARENTHESIS
LEFT_CURLY statements RIGHT_CURLY
                ;
statements : statements SEMICOLON statement
                | statements NEW_LINE
                | statement SEMICOLON
                | NEW_LINE
                ;
statement :  push_statement
                | for
                | while
                | if_else
                | function_declaration
                | function_call
                | variable_declaration
                | assignment
                | return
                | array_declaration
                ;
push_statement : PUSH LEFT_PARENTHESIS expression RIGHT_PARENTHESIS
                ;
```

```
pull_statement : PULL LEFT_PARENTHESIS RIGHT_PARENTHESIS
                    ;
for : FOR LEFT_PARENTHESIS assignment SEMICOLON expression operators
expression SEMICOLON expression RIGHT_PARENTHESIS LEFT_CURLY
statements RIGHT_CURLY
       ;
while : WHILE LEFT_PARENTHESIS expression operators expression
RIGHT_PARENTHESIS LEFT_CURLY statements RIGHT_CURLY
       ;
if_else : matched
              | unmatched
              ;
matched : IF LEFT_PARENTHESIS expression operators expression
RIGHT_PARENTHESIS matched ELSE matched
              | LEFT_CURLY statements RIGHT_CURLY
              ;
unmatched : IF LEFT_PARENTHESIS expression operators expression
RIGHT_PARENTHESIS if_else
              | IF LEFT_PARENTHESIS expression matched ELSE
unmatched
              ;


expression : expression MINUS expression_term
              | expression PLUS expression_term
              | expression OR_OPERATOR expression_term
              | expression_term
              ;
expression_term : expression_term MULTIPLICATION expression_factor
              | expression_term DIVISION expression_factor
              | expression_term AND_OPERATOR expression_factor
              | expression_factor
              ;
expression_factor : expression_factor MOD expression_factor2
              | expression_factor2;
expression_factor2 : LEFT_PARENTHESIS expression RIGHT_PARENTHESIS
              | primitive_types
              ;

function_declaration : types STRING LEFT_PARENTHESIS
function_declaration_argument_list RIGHT_PARENTHESIS LEFT_CURLY
statements RIGHT_CURLY
              ;
function_declaration_argument_list : STRING
              | function_declaration_argument_list COMMA STRING
              | empty
              ;
function_call : STRING LEFT_PARENTHESIS function_call_argument_list
RIGHT_PARENTHESIS
              ;
function_call_argument_list : expression
              | function_call_argument_list COMMA expression
              | empty
              ;
empty : ;
```

```
variable_declaration : ASSIGNMENT_OPERATOR types
variable_declaration_argument_list
                ;
variable_declaration_argument_list : STRING
                | variable_declaration_argument_list COMMA STRING;
variable_accessing_value : STRING
                | variable_accessing_value POINT STRING
            ;
return : RETURN variable_accessing_value;
assignment : pull_statement ASSIGNMENT_OPERATOR types STRING
                | pull_statement ASSIGNMENT_OPERATOR STRING
                | expression ASSIGNMENT_OPERATOR types STRING
                | expression ASSIGNMENT_OPERATOR
variable_accessing_value
                | function_call ASSIGNMENT_OPERATOR types STRING
                | function_call ASSIGNMENT_OPERATOR STRING
                ;
%%
#include "lex.yy.c"
int lineno;
int main(void){
 return yyparse();
}
yyerror( char *s ) { fprintf( stderr, "%s in line: %d\n", s,
lineno); };
```

EXPLANATION OF BNF

In this yacc part, we are sending the tokens from lex file to yacc file.The tokens are defined in %token section. Our program will recognize these tokens and perform the rules we defined above. We write the rules together thinking how a rule can be writen using tokens. For example program starts with void type, followed by paranthesis, left curly and right curly and between these left and right curly, we have statements.Statements must be recursive because we will have more than one statement. Such that writing statements ; statement is correct recursive form. Statement can have if statement, while statement, pull, push statement function call, function decrelation and such. If can be matched and unmatched. Expressions must have multiple rules together because in expression, each weight of operator is different and we must consider these cases.(* has more weight than +,- operation).To do that first we write the operands that have the same weight than we define another rule.(Just like we learned in class).And we are making the most important operands lower in the tree so that they will be calculated first.By doing that we will have no ambiguity in our language and there will be only one tree in our language.Functions can have argument, argument list, empty we must also consider these cases.Variable decrelation also have same cases.We considered these cases in our language and defined our rules regarding to this.

ADDITIONAL FUNCTIONALITY IN OUR LANGUAGE

Our language supports variety of functions for GIS purposes.

```
graph showonmap(int longtitude, int altitude);
```

This function returns the location as a graph by using the longtitude and altitude

```
graph searchlocation(string adress);
```

This function returns the location as a graph by using the adress as a string

```
int getroadspeed(int road);
```

This function gets the road number and returns the max speed limit of it

```
graph getlocation(string user);
```

This function gets the users name and returns the location as a graph

```
int getCurrentRoad(graph userLocation);
```

This function gets the users location and returns the current roads number

```
int showtargetlongtitude(graph location);
```

This function returns the longtitude of the current location

```
int showtargetaltitude(graph location);
```

This function returns the altitude of the current location

```
int square(int a);
```

This function is used for getting square purposes

```
int abs(int a);
```

This function returns the absolute version of the given input

```
string display(string a);
```

This function imitates the print function. It only accepts string type

```
string typeof(string a);
```

This function returns the typeof the given input

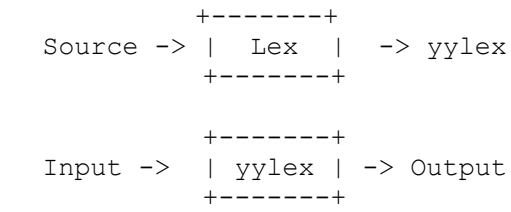
LEX DESCRIPTION OF THE LANGUAGE

```
%option yylineno
array                                   array
return                                  return
digit                                   [0-9]
sign                                    [+-]
letter                                  [A-Za-z]
newline                                 \n
lp                                      \(
rp                                      \)
lc                                      \{
rc                                      \}
lb                                      \[
rb                                      \]
quote                                   \~
alphanumeric                            ({letter}|{digit})
semicolon                               \;
whitespace                              [ \t\n]


true                                    #T
false                                   #F

assignment_operator                     \:\-\>
or_operator                             \|
and_operator                            \&
greater_operator                        \:\>
greater_or_equal_operator               \:\?\>
not_operator                            \:\<\>
equal_operator                          \:\?
less_or_equal_operator                  \:\?\<
less_operator                           \:\<

plus_operator                           \+
minus_operator                          \-
multiplication_operator                 \*
division_operator                       \/
mod_operator                            \^
exponential_operator                    \*\*
%%

{array} return(ARRAY);
{return} return(RETURN);
{or_operator} return(OR_OPERATOR );
{and_operator} return(AND_OPERATOR );
{not_operator} return(NOT_OPERATOR);
{equal_operator} return(EQUAL_OPERATOR);
{less_operator} return(LESS_OPERATOR);
{less_or_equal_operator} return(LESS_OR_EQUAL_OPERATOR);
{greater_operator} return(GREATER_OPERATOR);
{greater_or_equal_operator} return(GREATER_OR_EQUAL_OPERATOR);
```

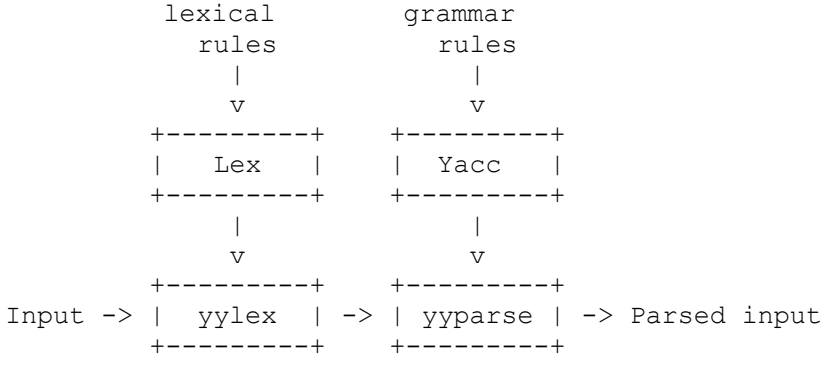
```
{assignment_operator} return(ASSIGNMENT_OPERATOR);


{lp} return(LEFT_PARENTHESIS);
{rp} return(RIGHT_PARENTHESIS);
{lc} return(LEFT_CURLY);
{rc} return(RIGHT_CURLY);
{lb} return(LEFT_BRACKET);
{rb} return(RIGHT_BRACKET);

{plus_operator} return(PLUS);
{minus_operator} return(MINUS);
{multiplication_operator} return(MULTIPLICATION);
{division_operator} return(DIVISION);
{exponential_operator} return(EXPONENTIAL);
{mod_operator} return(MOD);

{true} return(TRUE);
{false} return(FALSE);

program return(PROGRAM);
while return(WHILE);
for return(FOR);
if return(IF);
else return(ELSE);

push return(PUSH);
pull return(PULL);

int return(INT_TYPE);
float return(FLOAT_TYPE);
string return(STRING_TYPE);
void return(VOID_TYPE);
boolean return(BOOLEAN_TYPE);
graph  return(GRAPH_TYPE);

{sign}{digit}+ return(INTEGER);
{sign}{digit}+\.{digit}+ return(FLOAT);
{letter}+{alphanumeric}* return(STRING);

\, return(COMMA);
\. return(POINT);
{quote} return(QUOTE);
{semicolon} return(SEMICOLON);
\n { extern int lineno; lineno++;
                  return NEW_LINE;
                }
. return(INVALID);
%%
int yywrap(void){     return 1;}
```

EXPLANATION OF LEX

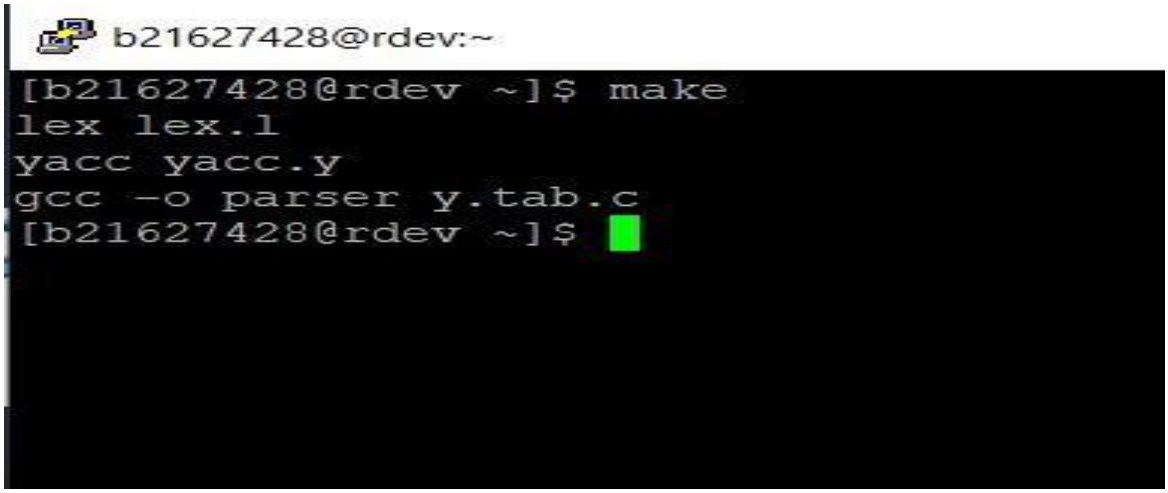
Lex is a program generator designed for lexical processing of character input. It accepts a high-level, problem oriented specification for character string matching, and produces a program in a general purpose language which recognizes regular expressions. The regular expressions are specified by the us in the source specifications given to Lex.It matches strings in given input and tokenizes them.We will use these tokens in the yacc part to define our rules.This is a really good explanation we have found in internet it explains how lex works:



Here is another explanation we have found it shows interaction between input, lex, yacc.



RUNNING THE LEX AND YACC



EXAMPLE PROGRAM TEST

```
void main(){

    graph showonmap(int longtitude, int altitude){
        :-> graph location;
        return (location);
    };

    graph searchlocation(string adress){
        :-> graph location;
        return (location);
    };
    int getroadspeed(int road){
        :-> int speed;
        return (speed);
    };
    graph getlocation(string user){
        :-> graph location;
        return (location);
    };
    int getCurrentRoad(graph userLocation){
```

```
        :-> int road;
        return (road);
    };
    int showtargetlongtitude(graph location){
        :-> int longtitude;
        return (longtitude);
    };
    int showtargetaltitude(graph location){
        :-> int altitude;
        return (altitude);
    };
    int square(int a){
        return (a*a);
    };
    int abs(int a){
        if(a :< 0){
            -1 * a :-> a;
        }
        return (a);
    };
string display(string a){
        if(:<>(typeof(a) :? ~string~)){
            return(~Element has to be string to display!~);
        };

    };
    string typeof(string a){
        return (~string~);
    };
    string typeof(int a){
        return (~int~);
    };
    string typeof(boolean a){
        return (~boolean~);
    };
    string typeof(graph a){
        return (~graph~);
    };

    :-> string destinationAdress;
    :-> int destinationLongtitude;
    :-> int destinationAltitude;
    :-> graph destinationLocation;

    :-> graph userLocation;
    :-> boolean approached;
    :-> string driverName;
```

```
    :-> int vechileSpeed;
    :-> int vechileLongtitude;
    :-> int vechileAltitude;

    :-> boolean driving;
    :-> string display;
    :-> int currentRoad;
    :-> int currentSpeed;

    approached :-> 0;

    ~home str. 163~ :-> destinationAdress;
    searchlocation(destinationAdress) :-> destinationLocation;
    showtargetlongtitude(destinationLocation):->
destinationLongtitude;
    showtargetaltitude(destinationLocation):->
destinationAltitude;
    showonmap(destinationLongtitude, destinationAltitude) :->
destinationLocation;
    ~ahmet~ :-> driverName;

    while( driving :? true){

        getlocation(driverName) :-> userLocation;
        if( destinationLocation :? userLocation){
            true :-> approached;
            false:-> driving;
            ~You have reached your destination!~ :-> display;
        };

        getCurrentSpeed(userLocation) :-> currentSpeed;
        getCurrentRoad(userLocation) :-> currentRoad;
        getroadspeed(currentRoad) :-> currentRoadSpeed;

        if(currentSpeed :> currentRoadSpeed){
            ~Warning your speed is spiking!~ :-> display;
        };
    };


}
```

RESULT OF THE TEST PROGRAM

```
RIGHT_CURLY SEMICOLON
INT_TYPE  STRING LEFT_PARENTHESIS INT_TYPE  STRING RIGHT_PARENTHESIS LEFT_CURLY
    IF LEFT_PARENTHESIS STRING  LESS_OPERATOR  0RIGHT_PARENTHESIS LEFT_CURLY
        STRING  ASSIGNMENT_OPERATOR  INTEGER  MULTIPLICATION  STRING
    RIGHT_CURLY
    RETURN  LEFT_PARENTHESIS STRING RIGHT_PARENTHESIS SEMICOLON
RIGHT_CURLY SEMICOLON
ASSIGNMENT_OPERATOR  STRING  STRING SEMICOLON
ASSIGNMENT_OPERATOR  INT_TYPE  STRING SEMICOLON
ASSIGNMENT_OPERATOR  INT_TYPE  STRING SEMICOLON
ASSIGNMENT_OPERATOR  GRAPH_TYPE  STRING SEMICOLON
ASSIGNMENT_OPERATOR  GRAPH_TYPE  STRING SEMICOLON
ASSIGNMENT_OPERATOR  BOOLEAN_TYPE  STRING SEMICOLON
ASSIGNMENT_OPERATOR  STRING_TYPE  STRING SEMICOLON
ASSIGNMENT_OPERATOR  INT_TYPE  STRING SEMICOLON
ASSIGNMENT_OPERATOR  INT_TYPE  STRING SEMICOLON
ASSIGNMENT_OPERATOR  INT_TYPE  STRING SEMICOLON
ASSIGNMENT_OPERATOR  BOOLEAN_TYPE  STRING SEMICOLON
ASSIGNMENT_OPERATOR  STRING_TYPE  STRING SEMICOLON
ASSIGNMENT_OPERATOR  INT_TYPE  STRING SEMICOLON
ASSIGNMENT_OPERATOR  INT_TYPE  STRING SEMICOLON
STRING  ASSIGNMENT_OPERATOR  QUOTE STRING QUOTE
STRING  ASSIGNMENT_OPERATOR  STRING LEFT_PARENTHESIS STRING RIGHT_PARENTHESIS SEMICOLON
STRING  ASSIGNMENT_OPERATOR  0SEMICOLON
STRING  ASSIGNMENT_OPERATOR  STRING LEFT_PARENTHESIS STRING RIGHT_PARENTHESIS
STRING  ASSIGNMENT_OPERATOR  STRING LEFT_PARENTHESIS STRING RIGHT_PARENTHESIS
STRING  ASSIGNMENT_OPERATOR  QUOTE STRING QUOTE
STRING  ASSIGNMENT_OPERATOR  STRING LEFT_PARENTHESIS STRING COMMA  STRING RIGHT_PARENTHESIS SEMICOLON
WHILE LEFT_PARENTHESIS  STRING  EQUAL_OPERATOR  STRING RIGHT_PARENTHESIS LEFT_CURLY
    STRING  ASSIGNMENT_OPERATOR  STRING LEFT_PARENTHESIS STRING RIGHT_PARENTHESIS SEMICOLON
    IF LEFT_PARENTHESIS  STRING  EQUAL_OPERATOR  STRING RIGHT_PARENTHESIS LEFT_CURLY
        STRING  ASSIGNMENT_OPERATOR  STRING SEMICOLON
        STRING  ASSIGNMENT_OPERATOR  STRING SEMICOLON
        STRING  ASSIGNMENT_OPERATOR  QUOTE STRING  STRING  STRING  STRING  STRING !QUOTE
    RIGHT_CURLY SEMICOLON

    STRING  ASSIGNMENT_OPERATOR  STRING LEFT_PARENTHESIS STRING RIGHT_PARENTHESIS SEMICOLON
    STRING  ASSIGNMENT_OPERATOR  STRING LEFT_PARENTHESIS STRING RIGHT_PARENTHESIS SEMICOLON
    STRING  ASSIGNMENT_OPERATOR  STRING LEFT_PARENTHESIS STRING RIGHT_PARENTHESIS
    IF LEFT_PARENTHESIS STRING  GREATER_OPERATOR  STRING RIGHT_PARENTHESIS LEFT_CURLY
        STRING  ASSIGNMENT_OPERATOR  QUOTE STRING  STRING  STRING  STRING  STRING !QUOTE SEMICOLON
    RIGHT_CURLY SEMICOLON
RIGHT_CURLY SEMICOLON
```

```
[b21727542@rdev v1]$ ./pls < test2.in
VOID_TYPE  PROGRAM LEFT_PARENTHESIS RIGHT_PARENTHESIS LEFT_CURLY

    INT_TYPE  STRING LEFT_PARENTHESIS INT_TYPE  STRING COMMA  INT_TYPE  STRING RIGHT_PARENTHESIS LEFT_CURLY
        ASSIGNMENT_OPERATOR  GRAPH_TYPE  STRING SEMICOLON
        RETURN  LEFT_PARENTHESIS STRING RIGHT_PARENTHESIS SEMICOLON
    RIGHT_CURLY SEMICOLON

    INT_TYPE  STRING LEFT_PARENTHESIS STRING_TYPE  STRING RIGHT_PARENTHESIS LEFT_CURLY
        ASSIGNMENT_OPERATOR  GRAPH_TYPE  STRING SEMICOLON
        RETURN  LEFT_PARENTHESIS STRING RIGHT_PARENTHESIS SEMICOLON
    RIGHT_CURLY SEMICOLON
    INT_TYPE  STRING LEFT_PARENTHESIS INT_TYPE  STRING RIGHT_PARENTHESIS LEFT_CURLY
        ASSIGNMENT_OPERATOR  INT_TYPE  STRING SEMICOLON
        RETURN  LEFT_PARENTHESIS STRING RIGHT_PARENTHESIS SEMICOLON
    RIGHT_CURLY SEMICOLON
    INT_TYPE  STRING LEFT_PARENTHESIS STRING_TYPE  STRING RIGHT_PARENTHESIS LEFT_CURLY
        ASSIGNMENT_OPERATOR  GRAPH_TYPE  STRING SEMICOLON
        RETURN  LEFT_PARENTHESIS STRING RIGHT_PARENTHESIS SEMICOLON
    RIGHT_CURLY SEMICOLON
    INT_TYPE  STRING LEFT_PARENTHESIS GRAPH_TYPE  STRING RIGHT_PARENTHESIS LEFT_CURLY
        ASSIGNMENT_OPERATOR  INT_TYPE  STRING SEMICOLON
        RETURN  LEFT_PARENTHESIS STRING RIGHT_PARENTHESIS SEMICOLON
    RIGHT_CURLY SEMICOLON
    INT_TYPE  STRING LEFT_PARENTHESIS GRAPH_TYPE  STRING RIGHT_PARENTHESIS LEFT_CURLY
        ASSIGNMENT_OPERATOR  INT_TYPE  STRING SEMICOLON
        RETURN  LEFT_PARENTHESIS STRING RIGHT_PARENTHESIS SEMICOLON
    RIGHT_CURLY SEMICOLON
    INT_TYPE  STRING LEFT_PARENTHESIS GRAPH_TYPE  STRING RIGHT_PARENTHESIS LEFT_CURLY
        ASSIGNMENT_OPERATOR  INT_TYPE  STRING SEMICOLON
        RETURN  LEFT_PARENTHESIS STRING RIGHT_PARENTHESIS SEMICOLON
    RIGHT_CURLY SEMICOLON
    INT_TYPE  STRING LEFT_PARENTHESIS INT_TYPE  STRING RIGHT_PARENTHESIS LEFT_CURLY
        RETURN  LEFT_PARENTHESIS STRING MULTIPLICATION STRING RIGHT_PARENTHESIS SEMICOLON
```

RESOLVING CONFLITCS AND PROBLEMS IN THE LANGUAGE

During this project, we got shift/reduce and reduce/reduce conflitcs. A shift-reduce conflict occurs in a state that requests both a shift action and a reduce action. A reduce-reduce conflict occurs in a state that requests two or more different reduce actions. After some searching, we understand that program obtains multiple parse trees from our code. The reason why this happens is there were operator presidences and operation assosiativities that are not defined very clear.

To solve the shift/reduce and reduce/reduce conflitcs, we used our knowledge that we learned in class, we defined the operation presidence and operation assosiativity between operators and operations of the language. We also divided arithmetic rules in to multiple rules.This helps making rules more readable, understandable and also gives less conflicts.

Making the language ambigious was the biggest problem of the language alongside with writing grammer to handle such operations. Ambiguous grammar is a context-free grammar for which there exists a string that can have more than one leftmost derivation or parse tree.

After that we have prepared a test input where we used rules, tokens we defined in YACC and LEX.And also we extended this with special data types, special functions. Our aim is with using lex and yacc, transfer that input to an output that generates no error.

As a summary we face with many problems during the definion phase and implementation state of the language. Due to fact that we have multiple and various operations, declarations , collection and primitive types , it's our main focus to keep the language maintained clear, therefore biggest effort goes to resolving conflicts that is generated by the YACC when we tried to compile the Language since there is no special tool to detect conflicts and errors, therefore this operation and resolving step done by manual debugging(by hand)

SHORT TUTORIAL ABOUT OUR LANGUAGE

Our language specializes around GIS. It has special functions, useful functions, useful operands what we can use in GIS applications. To make it global, easy to writable and easy to readable we used most common operands in our language.We have defined these operands, data types in our lex file and to make it understandable we have followed naming standarts that is followed by popular languages.

INTEGER,FLOAT,STRING,VOID

, :-> , :?, :?>, +,-,*,^,**,(,{[,],},), #T, #F etc…

The syntax of our language is similar to Java syntax. We are supporting decleration, assignment, object creation, loops, conditions, function declerations, function calls etc…These are very popular in GIS applications so people that use our language will not waste unnecessary time searching in the internet.

Our language supports extended version of comparison using logic statements, mathematical expression without creating ambiguity.Even if you make mistake writing mathematical equations(such as using different weight operands with no paranthesis) our language will take care of this ambigious state and will convert that equation to an not amgibious version.Our users will not make mistake in mathematical equations which is very common in GIS applications.Same for logical equations.In each case our program will convert to a tree which is defined higher weights in belove, lower weights in above so that lower operands will be calculated before the upper operands.

VARIABLE :-> INTEGER + INTEGER * INTEGER – INTEGER

VARIABLE :-> INTEGER +(INTEGER * INTEGER) - INTEGER

Our language also support multi if else block which is also very common in GIS application.User can write multiple if else condition in another if else

block.It increases user's writeability because it is very common using condition blocks in matmematical functions.

IF (VARIABLE_NAME OPERAND VARIABLE NAME){

EXPRESSION / EXPRESSION ; EXPRESSIONS

}

Our language also support function decleration anywhere in language, multi function decleration, function inside function, function with no argument, function with one argument or function with multiple argument.

Function is an another type for statement.You can call functions , assign return values of functions.

Exp:

TYPE FUNCTION_NAME(TYPE VARIABLE/VARIABLE_LIST/EMPTY){

EXPRESSION / EXPRESSIONS

}

We have also supported decleration of variables before assigning them, which adds more flexibility to our language.

Exp:

:-> TYPE VARIABLE_NAME

Also we have input and output in our language defined as pull, push.Users can give values to our language during execution, they can push values (similar to print).

Exp:

push(expression)

pull()

We also have graph data type which is unusual to see in other languages.They dont support graph data type. Because it is a GIS specialized language graph is very useful to have.In other languages you are expected to

implement your own graph type but we want our language to be used for users who are interested in GIS.Our users will have easy time to implement, use graph type for their personal problem. This is very important.GIS application writers will choose our language because we support graph type.

:-> GRAPH_TYPE VARIABLE_NAME

We also have error finder which finds the line which gives syntax error.It will give error and tell the line number to user so that our user will see which line gives error and user can find and debug their errors easily.If there is no error, our language returns The code is correct, compiled succesfuly to user.

This is a short tutorial to our language with the help of our lex file and rules above you can easily use our programming language for GIS applications.Hope this language is useful in your applications…