Course no : CSE 3212

Course Title : Compile design laboratory

Report no : 02

Report on : Writing a program in flex to tokenize the source file containing the source code of my language.

Submission date: 05/10/2023

Name: Mahbubur Rahman

Roll : 1907022

Department of CSE

KUET, Khulna

Objectives:

1. To design a new language with the help of flex.
2. To design Regular Expression for analyzing the designed language.
3. To create lexical analyzer for the language.

Introduction:

Documentation for the designed Language:

1. **Identifiers:**

   Identifier name can be started with any letters or underscore(_) followed by letters, digits and underscore. Such as

   dataType variableName, _variable0

   Data Types:

   The language has 4 data types. They are ->

   I. Integer: It represents data of integer type. The syntax is as follows -

      int x = 10 , y

   II. Float: It represents data of float type. The syntax is as follows -

      float x = 10.10, y = -2.00, z

   III. Character: It represents data of character type. It can be any single character enclosed with single quotation. The syntax is as follows -

      char x = 'n' , y=' ' , z

   IV. String: It represents an array of characters. It can be any collection of characters (except new line) enclosed with double quotation.

      string x = "hello world"

2. **Operators:**

   This language includes different types of operators.

Arithmetic Operators:			
No.	Operator	Symbol	Example
1	Addition	+	x = a + b
2	Subtraction	-	y = b – c
3	Multiplication	*	z = m * n
4	Division	/	x = a / b
5	Modulus	%	y = a % 5

Logical Operators:			
No.	Operator	Symbol	Example
1	And	&&	x > 1 && x < 9
2	Or	||	5 || 0
3	Not	!	!(x < 2)

Relational Operators:			
No.	Operator	Symbol	Example
1	Equal	==	a == 1
2	Greater	>	a > b
3	Less	<	b < 6
4	Greater than	>=	3 >= 2
5	Less than	<=	2 <= 3

Assignment Operators:			
No.	Operator	Symbol	Example
1	Assign	=	a = 1
2	Add and assign	+=	a += 7
3	Subtract and assign	-=	b -= 1
4	Multiply and assign	*=	x *= y
5	Division and assign	/=	a /= b

Bitwise Operators:			
No.	Operator	Symbol	Example
1	Bitwise Not	~	~ a
2	Bitwise And	&	a & b
3	Bitwise Or	|	b | 6
4	Bitwise Ex-or	^	3 ^ 4

3. **Loop:**

Currently, the language has only one loop i.e. “for” loop. Structure is as follows –

for (initialization; condition; increment/decrement)

[

// statements

]

Example: loop (integer z = 10; z < 10 && z < 20 || x < 10; z++)

[

```
        Int a =10 , b=5, c
        c = a + b
        write(a ,b ,c)
    ]
```

4. **Conditional Statements:**

Two types of conditional statements are : if and if…else

1. **If structure :**

```
If (condition)
[
    // statements
]
```

Example:

```
int a= 6 ,b =4
if (a>b)
[
   write(a)
]
```

2. **If…else structure:**

```
If (condition)
[
// statements
]
else
[
    // statements
]
```

Example:

```
int x=6 , b=2
if (x>b)
[
   write( x )
]
else
```

[
write(b)
]

5. **Comments**
   I. **Single line comment:** Single line comment is represented by single #
      # this is single line comment
   II. **Multi line comment:** Multi line comment is enclosed by // … //

Example:
```
//
This is
multi line
comment
##
```

6. **Starting and Ending of a program:**
   The first line of the file is the beginning and the end of file represents the ending of the source program.
7. **Input and Output:**
   I. **Input:**

   read(variable1, variable2, variable3)

   II. **Output:**

   write (variable1, variable2)

Source file (test.rm):

```
#Single line Commment
//
Multiple
line
Comment
//
int a,b=4
char c='6',B
float C=2.5,d=-2.00
string n="This is a string",m

read(B,m)
a+=b^2
write(m,a)
if a<10
[ a++ ]
if (! (b+c))
[ read(c,d)
write(n,m) ]
```

```
if b>10 [ ++c ]

if b>10 [a++] else [ b++ ]
if b>10 [a++]
else
[ b++ ]

for ( int a ; a<7 ;++a)
[ write(a) ]
```

Flex file (language.l):

```
%{
   #include<stdio.h>
   #include<stdlib.h>
   #include<string.h>

   typedef struct
   {
      char name[20];
      char type[20];
      char value[20];
   } variableInformation;

   variableInformation variable[20];
   char text[1000];
   int line=0,error=0,statement=0,variableCount=0;

   int isVariable(char s)
   {
      if(s>='a' && s<= 'z' || s>='A' && s<='Z' || s>='0' && s<='9')
      return 1;
   }

   int isOperator(char s)
   {
      if(s=='=')
      return 1;
   }

   void variableType(char *strText , int i, char *data , int length )
   {
      for(;i<length;i++)
      {
```

```
        int j=0;
        while(i<length && strText[i] == ' ')
        {
           ++i;
        }
        while(i<length && isVariable(strText[i]))
        {
           variable[variableCount].name[j++]=strText[i++];
           if(strText[i]==' ' | strText[i]==',' | strText[i]== '=' | strText[i]== 10 | i == length)
           {
              strcpy(variable[variableCount++].type,data);
              break;
           }
        }
        while(i<length && strText[i]!=',')
        {
           ++i;
        }
     }
  }

%}

Keyword (for|if|else|elsif|switch|catch|while|{Type})

SingleComment #[^#\n]+[\n]?
MultipleComment [/][/]([^/]*|[\n]*)+[/][/][\n]?
Comment {SingleComment}|{MultipleComment}

NewLine [\n]
Error .*

Digits [0-9]+
Number -?{Digits}
Float {Number}[.]{Digits}
String ["].*["]
Character ['].[']
Id [a-zA-Z_][a-zA-Z0-9_]*
Type (int|float|char|string)
VariableDeclare ([ ]*[=][ ]*({Number}|{Float}|{Character}|{String}))?([ ]*[,][ ]*{Id}([ ]*[=][
]*({Number}|{Float}|{Character}|{String}))?)*[ ]*[\n]?
Variables {Type}[ ]*{Id}{VariableDeclare}

Relational ([<]|[>]|[<][=]|[>][=]|[<][>]|[=][=])
```

```
Arithmetic [-+*/%]
Assignment ([=]|[+][=]|[-][=]|[*][=]|[/][=]|[%][=])
Logical ([&][&]|[|][|]|[!])
Bitwise [~^|&]
Unary ([+][+]|[-][-])
Operator ({Arithmetic}|{Relational}|{Logical}|{Unary}|{Bitwise})
Prefix {Unary}{Id}
Postfix {Id}{Unary}
Expression (({Id}[ ]*{Assignment}[ ]*({Digits}|{Id}))(([ ]*{Operator}[
]*({Digits}|{Id}))?)*)|{Prefix}|{Postfix})[ ]*[\n]?

Input read[(]{Id}((,{Id})?)*[)][\n]?
Output write[(]{Id}((,{Id})?)*[)][\n]?
Statement {Variables}|{Expression}|{Input}|{Output}

Operation (({Id}[ ]*{Operator}[ ]*({Digits}|{Id}))(([ ]*{Operator}[
]*({Digits}|{Id}))?)*)|{Prefix}|{Postfix})
OperationIf (({Bitwise}|[!])[ ]*(({Id}|{Digits})|[(][ ]*{Operation}[ ]*[)]))|{Operation}
ExpressionIf {OperationIf}|[(][ ]*{OperationIf}[ ]*[)]
ConditionIf if[ ]*{ExpressionIf}[ ]*
StatementIf [ ]*{ConditionIf}[\n]?[ ]*[[][ ]*([\n]*[ ]*{Statement}[ ]*[\n]*)*[ ]*[]][ ]*[\n]?
IfElse {StatementIf}[ ]*else[ ]*[\n]?[ ]*[[][ ]*([\n]*[ ]*{Statement}[ ]*[\n]*)*[ ]*[]][\n]?

Seperator ";"
Initial {Variables}|({Id}{VariableDeclare}(([ ]*[,][ ]*{Id}{VariableDeclare})?)*)
Condition {Id}[ ]*({Relational}|{Logical})[ ]*{Id}(([ ]*({Relational}|{Logical})[ ]*{Id})?)*
Iteration {Prefix}|{Postfix}
Block [[][ ]*([\n]*[ ]*{Statement}[ ]*[\n]*)*[ ]*[]]
LoopFor "for"[ ]*[(][ ]*{Initial}?[ ]*{Seperator}[ ]*{Condition}?[ ]*{Seperator}[ ]*{Iteration}?[
]*[)][ ]*[\n]*{Block}[\n]?

%%


{Input} {++line;++statement;printf("Input\n");printf("Line number : %d\n",line);}
{Output} {++line;++statement;printf("Output\n");printf("Line number : %d\n",line);}

{LoopFor} {++line;++statement;printf("For Loop\n");printf("Line number : %d\n",line);}

{StatementIf} {++line;++statement;printf("If condition\n");printf("Line number : %d\n",line);}
{IfElse} {++line;++statement;printf("If Else condition\n");printf("Line number : %d\n",line);}
{Expression} {++line;++statement;printf("Expression\n");printf("Line number : %d\n",line);}
{Variables} {++statement;printf("variable declared\n");++line;printf("Line number : %d\n",line);
        strcpy(text,yytext);
```

```
        int length=strlen(text); printf("%s\n",text); int i=0;
        for(;i<length;i++)
        {
           if(text[i]=='i' && text[i+1]=='n' && text[i+2]=='t')
           {
              variableType(text,i+3,"Integer",length);
              break;
           }
           else if(text[i]=='f' && text[i+1]=='l' && text[i+2]=='o' && text[i+3]=='a' &&
text[i+4]=='t')
           {
              variableType(text,i+5,"Float",length);
              break;
           }
           else if(text[i]=='c' && text[i+1]=='h' && text[i+2]=='a' && text[i+3]=='r')
           {
              variableType(text,i+4,"Character",length);
              break;
           }
           if(text[i]=='s' && text[i+1]=='t' && text[i+2]=='r' && text[i+3]=='i' && text[i+4]=='n' &&
text[i+5]=='g' )
           {
              variableType(text,i+6,"String",length);
              break;
           }
           else
           {
              continue;
           }
        }
        }

{MultipleComment} {++line;printf("Multiple line Comment\n");printf("Line number :
%d\n",line);}
{SingleComment} {++line;printf("Single line Comment\n");printf("Line number : %d\n",line);}

{Error} {++error;++line;printf("Error at line %d\n",line);}
{NewLine} {++line;printf("New Line %d\n",line);}
%%

int yywrap(){}

int main()
{
```

```
   printf("-----Language create-----\n");
   yyin=fopen("test.rm","r");
//    yyout=fopen("result.txt","a");
   yylex();
   int i=0;
   for(;i<variableCount;i++)
   {
      printf("Variable Type : %s\t name : %s\n",variable[i].type,variable[i].name);
   }
   return 0;
}
```

Sample Output:

```
-----Language create-----
Single line Comment
Line number : 1
Multiple line Comment
Line number : 2
variable declared
Line number : 3
int a,b=4

variable declared
Line number : 4
char c='6',B

variable declared
Line number : 5
float C=2.5,d=-2.00

variable declared
Line number : 6
string n="This is a string",m

New Line 7
Input
Line number : 8
Expression
Line number : 9
Output
Line number : 10
If condition
Line number : 11
If condition
```

Line number : 12
New Line 13
If condition
Line number : 14
New Line 15
If Else condition
Line number : 16
If Else condition
Line number : 17
New Line 18
Error at line 19
New Line 20
Error at line 21
Variable Type : Integer name : a
Variable Type : Integer name : b
Variable Type : Character name : c
Variable Type : Character name : B
Variable Type : Float name : C
Variable Type : Float name : d
Variable Type : String name : n
Variable Type : String name : m

Discussion:

The designed language is easy to use and understand. There is no need for any type of special character to indicate the end of the statement. The lexical analyzer will automatically recognize the end of the statement. Also, the error will be shown after analyzing each syntax with line number.

Conclusion:

There are still many features remain to be implemented in the future. This is language is primarily created by the motivation of c and swift language.

References:

1. **C language**
2. **Swift language**
3. **Regex101.com/**