KHULNA UNIVERSITY OF ENGINEERING & TECHNOLOGY

Department of Computer Science and Engineering

Course Title: **Compiler Design Laboratory**

Course No : **CSE 3212**

Project Report on Compiler Design

Date of Submission: 22 November, 2023

Submitted By	Submitted To
Farhatun Shama Roll: 1907033 Year: 3rd Semester: 2nd Department of Computer Science and Engineering (CSE) Khulna University of Engineering and Technology (KUET), Khulna-9203	Nazia Jahan Khan Chowdhury Assistant Professor Dipannita Biswas Lecturer Department of Computer Science and Engineering (CSE) Khulna University of Engineering and Technology (KUET), Khulna-9203

Objectives:

- To know about the steps of compilation and compilers.
- To learn about lexical analyzers and parsers
- To learn about flex and bison
- To gather knowledge about CFG and Regular Expressions
- To implement lexical analysis, syntax analysis and semantic analysis through flex and bison

Introduction:

Compilation is the process of translating a high level programming language into a executable machine code. It involves six steps. The first three are – lexical analysis, syntax analysis and semantic analysis. The first step i.e lexical analysis breaks the source code into meaningful structures (tokens). Parsing or Syntax analysis is where CFG is used to analyze the structure of the source code. Semantics analysis checks for the meaning of the statements.

Flex: Flex is the acronym for Fast Lexical Analyzer Generator. It is a tool for generating tokens from regular expressions from the input source file. It uses the regular expressions from the flex (.l) file to generate tokens and generates a c file (lex.yy.c) containing a yylex() function.

Bison(GNU Parser Generator) : Bison generates a parser to process the tokens generated by flex. It takes CFG from the .y file and generates tab.h and tab.c file. It uses LALR(1) parsing and parses from bottom to up to traverse to the start symbol. The yyparse() generated by bison calls the yylex() to obtain tokens. Two stacks – parse stack and value stack is used by Bison.

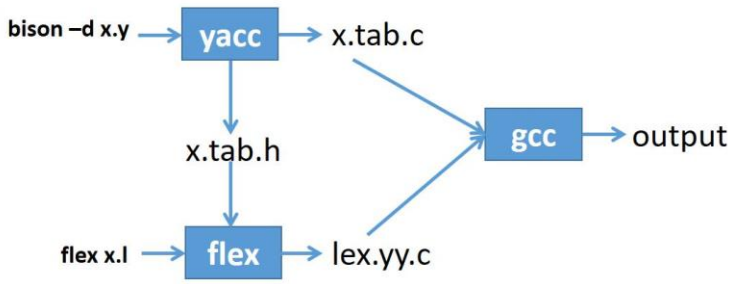

Figure 1.1: Workflow of flex and bison

Tools:

- Flex
- Bison
- C or C++ (g++) compiler
- Text editor

Methodology and Key Elements

Running the program: from cmd we write-

bison -d app.y

flex app.l

g++ app.tab.c lex.yy.c -o p

p

Regular Expressions and Tokens:

Serial	Regular Expression	Generated Token	Purpose
1.	Core	MAIN	Core or Main Function
2.	begin	BEGINN	Beginning of a function
3.	end	ENDD	End of a function
4.	Function	function	Keyword
5.	Void	VOID	Return type Void
6.	return	RETURN	Return statement
7.	Invite	CALL	Call a function
8.	arg	ARG	Access parameters of a function
9.	{white}*"("{white}*	LeftPAR	Left Bracket
10.	{white}*")"{white}*	RightPAR	Right Parenthesis
11.	{white}*"{"{white}*	RETBEGIN	Return type
12.	{white}*"}"{white}*	RETEND	Return type
13.	Constant	CONSTANT	Keyword

Serial	Regular Expression	Generated Token	Purpose
14.	Integer	INTEGER	Datatype
15.	Fraction	FLOAT	Datatype
16.	String	STRING	Datatype
17.	Binary	BOOLEAN	Datatype
18.	Symbol	CHARACTER	Datatype
19.	true	BOOLVALUE1	Boolean true
20.	false	BOOLVALUE0	Boolean false
21.	{id} id [A-Za-z][a-zA-Z0-9_@]*	IDENTIFIER	Identifier
22.	[+-]?[0-9]+	INTEGERVALUE	Integer
23.	[+-]?[0-9]+\.[0-9]+	FLOATVALUE	Float
24.	\'.\'	CHARVALUE	Character
25.	\"([^\\\"]|\\.)*\"	STRINGVALUE	String
26.	[\n]		Newline
27.	[\t]		Tab
28.	print	PRINT	Print something
29.	input	CIN	Take input from the user
30.	for	FOR	For loop
31.	next	NEXT	Continue statement
32.	out	BREAK	Break statement
33.	LoopStart	LOOPSTART	Start loop
34.	LoopEnd	LOOPEND	End loop
35.	Range	RANGE	Range loop
36.	RepeatWhile	WHILE	While loop
37.	if	IF	Condition
38.	elif	ELSEIF	Condition
39.	else	ELSE	Condition
40.	begif	BEGIF	Begin if or switch
41.	endif	ENDIF	End if or switch
42.	observe	SWITCH	Switch statement
43.	scenario	DEFAULT	Default
44.	Default	CASE	Case
45.	\;	NOSPACE_SPACE_SEMICOLON	End statement
46.	\,	NOSPACE_SPACE_COMMA	Separator
47.	\=	ASSIGN	assign
48.	\+	PLUS	Add
49.	\-	MINUS	subtract
50.	*	MUL	Product
51.	\/	DIV	Division
52.	\%	MOD	Modulo

Serial	Regular Expression	Generated Token	Purpose
53.	OR	OR	Or
54.	AND	AND	And
55.	NOT	NOT	Not
56.	ORbit	BITWISEOR	Bitwise or
57.	ANDbit	BITWISEAND	Bitwise and
58.	XORbit	BITWISEXOR	Bitwise xor
59.	\=\=	EQ	Relational op
60.	“>=”	GREATEREQ	Relational op
61.	“<=”	LESSEQ	Relational op
62.	“!=”	NOTEQUAL	Relational op
63.	“\>”	GREATER	Relational op
64.	“\<”	LESS	Relational op
65.	\!	FACT	Calculate factorial
66.	\^	POW	Calculate power
67.	\:	COLON	
68.	{white}*\<import\> {white}+\<[0-9A-Za-z_]+.lib\>{white}*	DIR	Directive
69.	{white}*\<include\> {white}+\<[0-9A-Za-z_]+.h\>{white}*	USER_DIR	Directive - User
70.	"#"[^#]*"#"	MULTI_LINE_COMMENT	Multi line comment
71.	{white}*\~\!.*	SINGLE_LINE_COMMENT	Single line comment

CFG (.y) file :

start : line globe function main function

line:

| line dir

| line SINGLE_LINE_COMMENT

| line MULTI_LINE_COMMENT

| dir

| SINGLE_LINE_COMMENT

| MULTI_LINE_COMMENT

```
        ;
globe:
                | globe declaration
                | declaration
                | globe SINGLE_LINE_COMMENT
                | globe MULTI_LINE_COMMENT
                | SINGLE_LINE_COMMENT
                | MULTI_LINE_COMMENT
                | globe const
        ;
main    : MAIN LeftPAR RightPAR RETBEGIN RETEND BEGINN statement ret ENDD
    ;
statement       :
                        | statement declaration
                        |declaration
                        | statement SINGLE_LINE_COMMENT
            | statement MULTI_LINE_COMMENT
                    | SINGLE_LINE_COMMENT
            | MULTI_LINE_COMMENT
                        |statement Arithmetic
                        |Arithmetic
                        |statement print
                        |print
                        |statement input
                        |statement ifs
                        |statement exp NOSPACE_SPACE_SEMICOLON
                        |statement loop
```

```
                        |statement range
                        |statement while
                        |statement call
                        |statement switch
                        |statement const
                        ;
/*header files */
dir: DIR
   |USER_DIR
        ;
/*function*/
function:
      |function FUNCTION IDENTIFIER LeftPAR param RightPAR RETBEGIN datatype1 RETEND
BEGINN statement ret
                ;
datatype1:
|datatype
|VOID
;
param:
    | datatype1 IDENTIFIER
         |param NOSPACE_SPACE_COMMA param
         ;
ret:
|RETURN NOSPACE_SPACE_SEMICOLON
|RETURN exp NOSPACE_SPACE_SEMICOLON
;
```

```
call : CALL IDENTIFIER LeftPAR callparam RightPAR NOSPACE_SPACE_
     ;
callparam:
      | callparam NOSPACE_SPACE_COMMA callparam
            |INTEGERVALUE
                  | FLOATVALUE
                  |STRINGVALUE
                  |BOOLVALUE0
                  |BOOLVALUE1
                  |CHARVALUE
                  |IDENTIFIER

/*variable declaration */

declaration: datatype idName NOSPACE_SPACE_ SEMICOLON
        ;
datatype: INTEGER  | FLOAT |CHARACTER |STRING  |BOOLEAN
      ;
idName: id NOSPACE_SPACE_COMMA idName
      |id

    ;
id:IDENTIFIER
          | IDENTIFIER ASSIGN exp
          |  IDENTIFIER ASSIGN float_exp
           |  IDENTIFIER ASSIGN char_exp                |  IDENTIFIER ASSIGN str_exp
          |  IDENTIFIER ASSIGN bool_exp
```

```
;

/*Constants */
const : CONSTANT datatypeconst constId NOSPACE_SPACE_SEMICOLON;
;
datatypeconst : INTEGER
          | FLOAT
                              ;
constId : conId NOSPACE_SPACE_COMMA constId
       | conId
      ;
conId :
      |IDENTIFIER ASSIGN exp
                  |IDENTIFIER ASSIGN float_exp
         ;


 /* Arithmetic Statement */
Arithmetic: IDENTIFIER ASSIGN exp NOSPACE_SPACE_SEMICOLON
|IDENTIFIER ASSIGN str_exp NOSPACE_SPACE_SEMICOLON
| IDENTIFIER ASSIGN float_exp NOSPACE_SPACE_SEMICOLON
| IDENTIFIER ASSIGN char_exp NOSPACE_SPACE_SEMICOLON
| IDENTIFIER ASSIGN bool_exp NOSPACE_SPACE_SEMICOLON
    | ARG LeftPAR IDENTIFIER RightPAR ASSIGN exp NOSPACE_SPACE_SEMICOLON
         | ARG LeftPAR IDENTIFIER RightPAR ASSIGN float_exp NOSPACE_SPACE_SEMICOLON
         | ARG LeftPAR IDENTIFIER RightPAR ASSIGN char_exp NOSPACE_SPACE_SEMICOLON
          | ARG LeftPAR IDENTIFIER RightPAR ASSIGN str_exp NOSPACE_SPACE_SEMICOLON
```

```
        | ARG LeftPAR IDENTIFIER RightPAR ASSIGN bool_exp NOSPACE_SPACE_SEMICOLON {



/*printf and scanf*/



print: PRINT LeftPAR exp RightPAR NOSPACE_SPACE_SEMICOLON

        | PRINT LeftPAR float_exp RightPAR NOSPACE_SPACE_SEMICOLON

        | PRINT LeftPAR char_exp RightPAR NOSPACE_SPACE_SEMICOLON

        | PRINT LeftPAR str_exp RightPAR NOSPACE_SPACE_SEMICOLON
        | PRINT LeftPAR bool_exp RightPAR NOSPACE_SPACE_SEMICOLON

        | PRINT LeftPAR STRINGVALUE RightPAR NOSPACE_SPACE_SEMICOLON

        | PRINT LeftPAR RightPAR NOSPACE_SPACE_SEMICOLON

        |  PRINT LeftPAR STRINGVALUE NOSPACE_SPACE_COMMA exp RightPAR
NOSPACE_SPACE_SEMICOLON

                        ;


input: CIN LeftPAR IDENTIFIER RightPAR NOSPACE_SPACE_SEMICOLON


/* Conditional */




ifs: IF LeftPAR exp RightPAR BEGIF statement11  elseif

    ;

        statement11:
```

```
            |statement
            |ifs
            ;
elseif:
    | ELSEIF LeftPAR exp RightPAR statement11 elseif
            | ELSE statement11
            ;
/* Switch Case */
switch :SWITCH LeftPAR value1 RightPAR BEGIF rec DEFAULT COLON stat ENDIF
;

rec: CASE INTEGERVALUE COLON stat rec
  |
 ;

stat:
   | stat Arithmetic
   | stat print
   | stat declaration
          |declaration
          | stat SINGLE_LINE_COMMENT
          | stat MULTI_LINE_COMMENT
          |stat input
          |stat exp NOSPACE_SPACE_SEMICOLON
   |stat const              ;
```

```
value1:IDENTIFIER                ;
/*loop*/
loop: FOR LeftPAR value COLON condition COLON value RightPAR LOOPSTART lines LOOPEND
;
lines:
   | loop
         |statement
         |lines BREAK
         |lines NEXT
         ;
value:  IDENTIFIER
                  |INTEGERVALUE
                  ;
condition: condition1 GREATER condition1
        | condition1 GREATEREQ condition1
                   | condition1 LESSEQ condition1
                   | condition1 LESS condition1
                   | condition1 EQ condition1
                   | condition1 NOTEQUAL condition1
                   |
                   ;
condition1:     IDENTIFIER {
                  |INTEGERVALUE
                  ;
range: RANGE LeftPAR value COLON value COLON value RightPAR LOOPSTART liness LOOPEND ;
liness:
      | range
```

```
            |statement
            |liness BREAK
            |liness NEXT
        ;

while : WHILE LeftPAR condition RightPAR LOOPSTART linee
linee:
    | while
            |statement
            |linee BREAK
            |linee NEXT
        ;

 /*expression*/
 exp: INTEGERVALUE
        |IDENTIFIER
       |ARG LeftPAR IDENTIFIER RightPAR
        |MINUS exp %prec UMINUS { $$ = -$2;}
        |exp PLUS exp    { $$ = $1 + $3;}
        |exp MINUS exp   { $$ = $1 - $3;}
        |exp MUL exp     { $$ = $1 * $3;}
        |exp DIV exp
        |exp MOD exp
        |exp AND exp      { $$ = $1 && $3;}
        |exp OR exp      { $$ = $1 || $3;}
        |NOT exp         { $$ = ! $2 ;}
        |exp BITWISEAND exp { $$ = $1 & $3;}
```

```
        |exp BITWISEOR exp    { $$ = $1 | $3;}
        |exp BITWISEXOR exp    { $$ = $1 ^ $3;}
        |exp FACT
        |exp POW exp
        |exp EQ exp      { $$ = $1 == $3;}
        |exp GREATER exp     { $$ = $1 > $3;}
        |exp GREATEREQ exp      { $$ = $1 >= $3;}
        |exp NOTEQUAL exp    { $$ = $1 != $3;}
        |exp LESS exp    { $$ = $1 < $3;}
        |exp LESSEQ exp    { $$ = $1 <= $3;}
        |LeftPAR exp RightPAR   {($2);}
    ;
str_exp:  STRINGVALUE
            |IDENTIFIER
;
float_exp:  FLOATVALUE
        |IDENTIFIER
          |MINUS float_exp %prec UMINUS { $$ = -$2;}
          |float_exp PLUS float_exp   { $$ = $1 + $3;}
          |float_exp MINUS float_exp   { $$ = $1 - $3;}
          |float_exp MUL float_exp     { $$ = $1 * $3;}
          |float_exp DIV float_exp
          |float_exp AND float_exp      { $$ = $1 && $3;}
          |float_exp OR float_exp      { $$ = $1 || $3;}
          |float_exp float_exp        { $$ = ! $2 ;}
          |float_exp EQ float_exp      { $$ = $1 == $3;}
          |float_exp GREATER float_exp     { $$ = $1 > $3;}
```

```
        |float_exp GREATEREQ float_exp      { $$ = $1 >= $3;}
        |float_exp NOTEQUAL float_exp    { $$ = $1 != $3;}
        |float_exp LESS float_exp    { $$ = $1 < $3;}
        |float_exp LESSEQ float_exp    { $$ = $1 <= $3;}
        |LeftPAR float_exp RightPAR   {($2);}
    ;
char_exp:  CHARVALUE     {$$=$1;}
         |IDENTIFIER
    ;
bool_exp: BOOLVALUE0
       |BOOLVALUE1
                |IDENTIFIER
```

Sample Input and output:

Input:

# Compiler Project 1907033 Flex and bison # ~! Directives : System libraries <import> <inout.lib> <import> <stl.lib> ~! Directives : User defined header <include> <file.h> <import> <py.lib> ~!Global Variable and Constant Integer global_i = 8;	87 is a while loop statement Integer x0=0; RepeatWhile (x1 == 0) LoopStart Range (1 : 100 : 20) LoopStart LoopEnd Integer x0=0; LoopEnd Integer aii = 90; observe (aii) begif

```
 Constant Fraction GLOBAL_F = 7.5;
 Symbol sym = '1';
 String global_str;

 ~!Function
Function function (Integer r, Integer y) {
Integer }
begin
arg ( r ) = 5 ;
Integer x = arg ( r ) ;
print ( x );
return 5;
end

Core ( ) { }
begin
Integer integer,integer1=1;
Fraction fraction = 9.5 , fraction1;
Symbol sym1 = 's', sym2;
String string1 ;
Binary bin1 = true;
Binary bin2 = "true";
integer = 8 + 7 + 1;
print ( "integer value : ", integer);
string1= "abc ";
Invite fun ( 3 , 4);
Invite function ( 1 , 2, 3);
Invite function ( 1 , 2 );
Constant Integer const = 9;
const = 10;
integer1 = const;
integer1= integer1 + 1*10;
print (12.09);
print ("Enter an integer : ");
~!input (integer1);
print (integer1);
print("String");
Integer bi=90 + 6,ci = bi + 34,aiii;
ci = bi + 6;


for ( bi : bi>89 : 2)
   LoopStart
   3 < 4 ;
scenario 90 :
   print (" sce0", 90 );
scenario 5 :
   print (" sce1 ", 5 );
default :
endif


if ( 0 )
   begif
   print ( 5 );
elif ( 0 )
   print ( "else if" );
elif ( 1 )
   print ( "else if" );
else
   print ( "else " );
endif

if ( 1 )
   begif
   if ( 1 )
   begif
   endif
endif

Range ( 1 : 100 : 20 )
LoopStart
if ( 0 )
   begif
   print ( 5 );
elif ( 0 )
   print ( "else if" );
elif ( 1 )
   print ( "else if" );
else
   print ( "else " );
endif
LoopEnd

Integer x = 2;
Integer y =3;
Integer z = x ORbit y ;
return;
```

```
    aiii = 889;
    LoopEnd
aiii = 80;
for ( bi : bi<89 : 2)
    LoopStart
    for ( aiii : bi<89 : 2)
    LoopStart
    print ("Aiii : ", aiii);
    LoopEnd
    out
    next
    LoopEnd
Integer x1=0;
Range ( 1 : 100 : 20 )
    LoopStart
    Fraction frax=0.0;
    Range ( 1 : 10 : 2 )
    LoopStart
    Integer x0=0;
    LoopEnd
    LoopEnd
```

```
end



Function fun ( Integer X , Fraction rtfrac ) {
Void }
begin
Binary bin = true ;
return ;
end

Function fun_to_call ( Integer Xio , Fraction
rtfracy ) { Void }
begin
Integer xop = 1;
arg ( Xio ) = xop ;
Integer gk;
gk = arg ( Xio );
Constant Fraction gh = 7.9;
print ( 3 < 4 );
return ;
end
```

Output :

```
line 2  : MultiLine Comment Ends.
line 4  : SINGLE_LINE_COMMENT
line 5  : System Library
line 6  : System Library
line 8  : SINGLE_LINE_COMMENT
line 9  : User Header
line 11  : System Library
line 13  : SINGLE_LINE_COMMENT

line 14 : Datatype : Integer
line : 14 Variable : global_i initialized with 8
line 14  : Variable Declared
```

```
line 76 : Datatype : Integer
line : 76 Variable : x0 initialized with 0
line 76  : Variable Declared

77 is a Range loop statement
initial value: 1 ,final value: 10increment : 2
Number of execution : 5
78 is a Range loop statement
initial value: 1 ,final value: 100increment : 20
Number of execution : 5

line 79 : Datatype : Integer
line : 79 Variable : x0 initialized with 0
```

```
Identifier : GLOBAL_F value: 7.5 Line : 15
Constant declared

line 16  : Datatype : character/symbol
line : 16 Variable : sym initialized with 1
line 16  : Variable Declared



line 17 : Datatype : string ,Variable global_str
line 17  : Variable Declared

line 19  : SINGLE_LINE_COMMENT

line 20 : Datatype : Integer
line 20 : Datatype : Integer
line 20 : Datatype : Integer Function begins

line : 22 value 5 assigned to arg r

line 23 : Datatype : Integer
line : 23 Variable : x initialized with 5
line 23  : Variable Declared

line 24 print output
5
Function ends
 Function declaration complete at line 26
no_arg : 2
Core Function
Function begins

line 30 : Datatype : Integer ,Variable integer

line : 30 Variable : integer1 initialized with 1
line 30  : Variable Declared



line 31  : Datatype : Float/Fraction
line : 31 Variable : fraction initialized with 9.5
,Variable fraction1
line 31  : Variable Declared



line 32  : Datatype : character/symbol
```

```
Compilation Error : Variable x0is already
declared79
line 79  : Variable Declared

85 is a Range loop statement
initial value: 1 ,final value: 100increment : 20
Number of execution : 5

line 86 : Datatype : Integer
line : 86 Variable : x0 initialized with 0

Compilation Error : Variable x0is already
declared86
line 86  : Variable Declared

line 89 : Datatype : Integer
line : 89 Variable : aii initialized with 90
line 89  : Variable Declared

line 93 print output
 sce090
line 95 print output
 sce1 5
switch
90 block is executed
Switch Statement Completed, line 97
line 102 print output
5
line 104 print output
else if
line 106 print output
else if
line 108 print output
else
if statement completed of 109
about if block of 109
 else if block 2will be executed
This is an else if ladder block
```

```
line : 32 Variable : sym1 initialized with s
,Variable sym2
line 32  : Variable Declared


line 33 : Datatype : string ,Variable string1
line 33  : Variable Declared


line 34  : Datatype : Boolean/Binary
line : 34 Variable : bin1 initialized with true
line 34  : Variable Declared


line 35  : Datatype : Boolean/Binary
Compilation Error : attempt to assign a string
value to a non string variable line 35
line 35  : Variable Declared


line : 36 value 16 assigned to integer
line 37 print output
integer value : 16

line : 38 value "abc " assigned to string1
Error : Function fun not found. line 39

ERROR : Function has 2 parameters. It does
not match with 3 number . line : 40

 Line - 41 : Function called with 2 parameters.

Identifier : const value: 9 Line : 42
Line : 42Constant declared:
Compilation Error:
Line no: 43   Variable const is not of integer
type or is contant

line : 44 value 9 assigned to integer1

line : 45 value 19 assigned to integer1
line 46 print output
12.09
line 47 print output
```

```
if statement completed of 115
about if block of 115
 if block will be executed.
NESTED IF115
if statement completed of 116
about if block of 116
 if block will be executed.
line 122 print output
5
line 124 print output
else if
line 126 print output
else if
line 128 print output
else
if statement completed of 129
about if block of 129
 else if block 2will be executed
This is an else if ladder block
130 is a Range loop statement
initial value: 1 ,final value: 100increment : 20
Number of execution : 5

line 132 : Datatype : Integer
line : 132 Variable : x initialized with 2

Compilation Error : Variable xis already
declared132
line 132  : Variable Declared


line 133 : Datatype : Integer
line : 133 Variable : y initialized with 3
line 133  : Variable Declared


line 134 : Datatype : Integer
line : 134 Variable : z initialized with 7
line 134  : Variable Declared

Function ends
 Core Function complete at line 136

line 141 : Datatype : Integer
```

```
Enter an integer :
line 48  : SINGLE_LINE_COMMENT
line 49 print output
19
line 50 print output
String

line 51 : Datatype : Integer
line : 51 Variable : bi initialized with 96

line : 51 Variable : ci initialized with 130
,Variable aiii
line 51  : Variable Declared


line : 52 value 102 assigned to ci

line : 58 value 889 assigned to aiii
This is a for loop statement line: 59
initial value: 96 ,Condition : 1increment : 2

line : 60 value 80 assigned to aiii
line 65 print output
Aiii : 80
This is a for loop statement line: 66
initial value: 80 ,Condition : 0increment : 2
NESTED FOR LOOP
break
continue
This is a for loop statement line: 69
initial value: 96 ,Condition : 0increment : 2

line 70 : Datatype : Integer
line : 70 Variable : x1 initialized with 0
line 70  : Variable Declared


line 73  : Datatype : Float/Fraction
line : 73 Variable : frax initialized with 0
line 73  : Variable Declared
```

```
line 141  : Datatype : Float/Fraction Function
begins

line 143  : Datatype : Boolean/Binary
line : 143 Variable : bin initialized with true
line 143  : Variable Declared

Function ends
 Function declaration complete at line 145
no_arg : 2

line 147 : Datatype : Integer
line 147  : Datatype : Float/Fraction Function
begins

line 149 : Datatype : Integer
line : 149 Variable : xop initialized with 1
line 149  : Variable Declared


line : 150 value 1 assigned to arg Xio

line 151 : Datatype : Integer ,Variable gk
line 151  : Variable Declared


line : 152 value 1 assigned to gk

Identifier : gh value: 7.9 Line : 153
Constant declared
line 154 print output
1
Function ends
 Function declaration complete at line 156
no_arg : 2


----------Compiled Successsfully----------
-------
```

Discussion :

In this lab we learned about compiler design phases. The first three stages are implemented in this small project. Some difficulties were encountered while completing the project like shift reduce and reduce-reduce conflicts. The grammar writing system in bison is quite similar to the theoretical approach with a few differences. The C++ language is used in this project. This project contains regular expressions as well as CFGs and corresponding actions or semantics.

Overall, this project has functions, error handling, declaration of variables and constants, assignment, arithmetic expressions, loop , conditional statements etc.

Conclusion:

Compilers are necessary tools for programming . The compiler developed in this lab does not contain all the features of a complete compiler. It only has the basic three stages implemented.

Also many features like stl , structures, loops can be added. Overall this incorporates the basic features with slots for further improvement.

References:

- https://www.skenz.it/compilers/flex_bison