



Khulna University of Engineering and Technology

REPORT on Custom Built Programming Language Using Flex and Bison  
Course Name: Compiler Design Laboratory Course Code: CSE 3212

### Submitted From

Name: Naimur Rahman

Roll: 1907031

Section: A

Dept of CSE, KUET

### Submitted to

Nazia Jahan Khan Chowdhury  
Assistant Professor

Dipannita Biswas  
Lecturer

Dept of CSE, KUET

Date of Submission:  
21-11-2023



## **Objectives:**

- ✓ Recognize keywords, identifiers, literals, and operators in the source code
- ✓ Handle whitespace and comments gracefully
- ✓ Generate error tokens/messages for duplicate variables
- ✓ Optimize regular expressions for improved lexer performance
- ✓ Provide clear and informative error messages
- ✓ Implement Loops
- ✓ Implement Conditionals
- ✓ Implement Functions

## **Introduction:**

Flex and Bison are powerful tools in the realm of compiler construction, aiding developers in the creation of lexers and parsers. Flex, the fast lexical analyzer generator, excels in recognizing patterns in input, while Bison, the parser generator, is adept at parsing the structure defined by a context-free grammar. Together, they form a dynamic duo for processing and interpreting the syntax of programming languages or structured input. This report delves into the implementation and integration of Flex and Bison in a project aimed at parsing and analyzing a new syntax-based programming language. Through this exploration, we aim to showcase the effectiveness of these tools in constructing robust and efficient language processors.

## **ID and Type Design:**

### **Datatypes:**

datatype	int   float   string
void	void (Only used as a function prefix)

### **Type Design:**

Int	"-"?{digits}+
Float	"-"?({digits}+)?".{digits}+
String	"\"\"^[^"]*"\""
Identifiers	[a-zA-Z][a-zA-Z0-9]*

## **Headers:**

Import Prefix	"#include"
Main Style	{Identifiers}".h"

## **Loops:**

- ✓ while                      similar to while loop  
    while (conditions) {statements}
- ✓ for                         similar to for loop  
    for (initialization, condition, inc/dec condition) {statements}

## **Operator Handling:**

+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Power
mod	Remainder
()	Brackets (To Specify Custom Precedence)
less	Less Than
great	Greater Than
equal	Equal To
great_eq	Greater Than Equal To
less_eq	Less Than Equal To
not_eq	Not Equal To
++	Increment
--	Decrement
!	Not

## **Conditional Operations:**

- ✓ if                         similar to if  
    if (condition) {statements}
- ✓ else\_if                 similar to else if  
    else\_if (condition) {statements}
- ✓ else                     similar to else  
    else {statements}

## **Functions:**

main	{Datatype}[ ]+"main"
functions	"function" {identifier} (parameters) {statements}
call	{identifier} (parameters) ';'

## **Built-In Functions:**

sin	Mathematical sin
cos	Mathematical cos
tan	Mathematical tan
log	Mathematical 10 based Log
ln	Mathematical e based Log
isOddEven	Determine Even or Odd Number
factorial	Calculate Factorial Value
max	Calculate Max Value
min	Calculate Min Value
isPrime	Determine Prime or Not Prime

## **Comments:**

Single Line Comment	\\\[^\n]*
Multi Line Comment	\\\[^\n]*\\\[^\n]*\\\[^\n]*

## **Showing Values:**

Print	Prints Value of Variable
-------	--------------------------

## **Switch Case:**

switch	Similar to Switch
switch (condition) {switch statements}	
case condition: {statements} or	
case default: {statements}	

## Source File:

### input.txt

```
#include <stdio.h>

function Abc (int i, int j) {
    i = 5;
}

int main() {
    int a = 6;
    int b = a;
    int c = 7, d = 5;
    float x = 7.5;
    string y = "abc";
    c = 3;
    b = 8;
    a = b + c;
    a = b - c;
    a = b mod c;
    a = b*c;
    a = b/c;
    a = b^c;
    a = b+b/c+c;
    a = b great c;
    a = b less c;
    a = b equal c;
    a = b not_eq c;
    a = b less_eq c;
    a = b great_eq c;
    c = a++;
    c = a--;
    c = !a;
    c = sin(b);
    c = cos(b);
    c = tan(b);
    c = log(b);
    c = ln(b);
    isOddEven(b);
    factorial(b);
    a = max(b,c);
    a = min(b,c);
    isPrime(b);
    Abc(b,c);
    print(a);
    if(a equal b) {
        print(a);
    }
    else_if(a equal c) {
        print(a);
    }
    else {
        print(a);
    }
    int l;
    for(l=0, l less a, l inc 1) {
        print(b);
    }
    while(l great b) {
        print(c);
    }
    switch(a) {
        case 8: {
            print(a);
        }
        default: {
            print(b);
        }
    }
    }
    //single line comment
    /*Multi
    Line
    Comment*/
}
```

## Flex File:

1907031.1

```
1  digits [0-9]
2  Datatype "int"|"float"|"void"|"string"
3  Identifiers [a-zA-Z][a-zA-Z0-9]*
4  S_comment \\/[^\n]*
5  M_comment \\/*([^\n]|\\*\\/*)*\\/*
6
7  %{
8      #include "1907031.tab.h"
9      #include <stdio.h>
10     #include <stdlib.h>
11     #include <string.h>
12     int varindex(char *var);
13     extern int yylex();
14     extern int yyparse();
15     extern FILE *yyin;
16     extern FILE *yyout;
17     int yyerror(char *s);
18     //int lineNo = 1;
19  %}
20
21  %%
22
23  {S_comment} { printf("\nSingle Line Comment\n"); }
24  {M_comment} { printf("\nMultiple Line Comment\n"); }
25
26  "int" { return INT; }
27  "float" { return FLOAT; }
28  "string" { return STRING; }
29
30  "(" { return '('; }
31  ")" { return ')'; }
32  "<" { return '<'; }
33  ">" { return '>'; }
34  "{" { return '{'; }
35  "}" { return '}'; }
36  ";" { return END; }
37  "," { return ','; }
38  "=" { return '='; }
39  ":" { return ':'; }
40
41  "+" { return '+'; }
42  "-" { return '-'; }
43  "*" { return '*'; }
44  "/" { return '/'; }
45  "^" { return '^'; }
46  "mod" { return MOD; }
47
48  "less" { return LT; }
49  "great" { return GT; }
50  "equal" { return EQ; }
51  "great_eq" { return GEQ; }
52  "less_eq" { return LEQ; }
53  "not_eq" { return NEQ; }
54
55  "++" { return INC; }
56  "--" { return DEC; }
57  "!" { return NOT; }
58
59  "sin" { return SIN; }
```

```

60 "cos" { return COS; }
61 "tan" { return TAN; }
62 "ln" { return LN; }
63 "log" { return LOG; }
64 "isOddEven" { return ODDEVEN; }
65 "factorial" { return FACTORIAL; }
66 "max" { return MAX; }
67 "min" { return MIN; }
68 "isPrime" { return PRIME; }
69
70 "print" { return DISPLAY; }
71
72 "if" { return IF; }
73 "else_if" { return ELSE_IF; }
74 "else" { return ELSE; }
75
76 "for" { return FOR; }
77 "inc" { return FLINC; }
78 "dec" { return FLDEC; }
79 "while" { return WHILE; }
80
81 "case" { return CASE; }
82 "switch" { return SWITCH; }
83 "default" { return DEFAULT; }
84
85 "-"?{digits}+ {
86     yylval.string = strdup(yytext);
87     return NUMBER;
88 }
89
90 "-"?({digits}+)?"."{digits}+ {
91     yylval.string = strdup(yytext);
92     return NUMBER;
93 }
94
95 "\"\"[^\"]*\"\" {
96     yylval.string = strdup(yytext);
97     return STR;
98 }
99
100 {Datatype}[ ]+"main" { return MAIN; }
101 "#include" { return IMPORT; }
102 {Identifiers} ".h" { return HEADER; }
103
104 "function" { return DEF; }
105
106 {Identifiers} {
107     yylval.string = strdup(yytext);
108     return VARIABLE;
109 }
110
111 [ \t\n]*
112
113 . {yyerror("Unknown Character.\n");}
114
115 %%

```

## Bison File:

1907031.y

```
1  %{
2  #include<stdio.h>
3  #include <math.h>
4  #include<stdlib.h>
5  #include<string.h>
6  extern FILE *yyin;
7  extern FILE *yyout;
8  int yylex();
9  int yyerror(char *s);
10
11 // Symbol Table Arrays
12 char var_name[1000][100];
13 int store_int[1000];
14 float store_float[1000];
15 char store_String[1000][100];
16 int type[1000];
17 int var_type_pointer = 0; // 0 = int, 1 = float, 2 = string, 3 = function
18
19 // Conditional Statement Variables
20 int if_pointer = 0;
21 int store_if[1000];
22
23 // Switch Handling Variables
24 int switch_var = 0;
25 int switch_case = 0;
26
27 // Variable Counter
28 int var_cnt = 0;
29
30 // Variable Declaration Check
31 int checkDeclared(char *s){
32     int i;
33     for(i=0; i<var_cnt; i++){
34         if(strcmp(var_name[i], s) == 0)
35             return 1;
36     }
37     return 0;
38 }
39
40 // New Variable Declaration
41 int varAssign(char *s){
42     if(checkDeclared(s) == 1){
43         return 0;
44     }
45     strcpy(var_name[var_cnt], s);
46     store_int[var_cnt] = 0;
47     store_float[var_cnt] = 0.0;
48     strcpy(store_String[var_cnt], "");
49     type[var_cnt] = var_type_pointer;
50     char name[10];
51     if(var_type_pointer == 0) {
52         strcpy(name, "Int");
53     }
54     else if(var_type_pointer == 1) {
55         strcpy(name, "Float");
56     }
57     else if(var_type_pointer == 2) {
58         strcpy(name, "String");
59     }
```



```

60     printf("\nNew Variable Declared With Name: %s and Type: %s\n", var_name[var_cnt], name);
61     var_cnt++;
62     return 1;
63 }
64
65 // New Function Declaration
66 int functionAssign(char *s){
67     if(checkDeclared(s) == 1){
68         return 0;
69     }
70     strcpy(var_name[var_cnt], s);
71     store_int[var_cnt] = 0;
72     store_float[var_cnt] = 0.0;
73     strcpy(store_String[var_cnt], "");
74     type[var_cnt] = 3;
75     printf("\nNew Function Declared With Name: %s\n", var_name[var_cnt]);
76     var_cnt++;
77     return 1;
78 }
79
80 // Assigning Value to Variable
81 int setValue(char *s, char* val){
82     if(checkDeclared(s) == 0){
83         return 0;
84     }
85     int ok=0, i;
86     for(i=0; i<var_cnt; i++){
87         if(strcmp(var_name[i], s) == 0){
88             ok = i;
89             break;
90         }
91     }
92     if(type[ok] == 0){
93         store_int[ok] = atoi(val);
94         printf("\nNew Value Assigned to Variable Name: %s and Value: %d\n", var_name[ok],
store_int[ok]);
95     }
96     else if(type[ok] == 1){
97         store_float[ok] = atof(val);
98         printf("\nNew Value Assigned to Variable Name: %s and Value: %f\n", var_name[ok],
store_float[ok]);
99     }
100     else if(type[ok] == 2){
101         strcpy(store_String[ok], val);
102         //store_String[ok] = val;
103         printf("\nNew Value Assigned to Variable Name: %s and Value: %s\n", var_name[ok],
store_String[ok]);
104     }
105     else{
106         printf("\nCan't Assign Value as Variable is a Function!\n");
107     }
108     return 1;
109 }
110
111 // Get Variable Value
112 int getValue(char *s){
113     int pos=-1;
114     int i;
115     for(i=0; i<var_cnt; i++){
116         if(strcmp(var_name[i], s) == 0){
117             pos=i;
118             break;
119         }

```

```

120     }
121     return pos;
122 }
123 %}
124
125 %union
126 {
127     int num;
128     float flt;
129     char* string;
130 };
131
132 %token END INT FLOAT STRING MOD
133 %token LT GT GEQ LEQ EQ NEQ
134 %token <string> VARIABLE
135 %token <string> NUMBER
136 %type <string> expression
137 %token <string> STR
138 %token IMPORT HEADER MAIN
139 %token INC DEC NOT
140 %token SIN COS LOG TAN LN
141 %token ODDEVEN FACTORIAL MAX MIN PRIME
142 %token DEF DISPLAY
143 %token IF ELSE_IF ELSE
144 %token FOR FLINC FLDEC WHILE
145 %token CASE SWITCH DEFAULT
146
147 %left LT GT GEQ LEQ EQ NEQ
148 %left '+' '-'
149 %left '*' '/' MOD
150 %left '^'
151
152 %%
153
154 program:
155     import func main '(' ' ' ')' '{' statements '}' { printf("\nProgram Successfully Ended!\n"); }
156     | /* NULL */
157     ;
158
159 main:
160     MAIN { printf("\nMain Function Declared!\n"); }
161
162 import: /* NULL */
163     | import IMPORT '<' HEADER '>' { printf("\nHeader File Found!\n"); }
164     ;
165
166 func:
167     func_head '(' param ')' '{' statements '}' {
168         printf("\nUser Defined Function Ended!\n");
169     }
170     | /* NULL */
171     ;
172
173 func_head:
174     DEF VARIABLE {
175         if(checkDeclared($2)==1) {
176             printf("\nDuplicate Function Name!\n");
177         }
178         else {
179             functionAssign($2);
180         }
181     }

```

```

182
183 param:
184     param ',' type pid { printf("\nValid Function Parameter Declaration!\n"); }
185     | type pid { printf("\nValid Function Parameter Declaration!\n"); }
186     ;
187
188 pid :
189     VARIABLE {
190         if(checkDeclared($1)==1) {
191             printf("\nDuplicate Declaration!\n");
192         }
193         else {
194             varAssign($1);
195         }
196     }
197     ;
198
199 statements:
200     statements cstatement
201     | /* NULL */
202     ;
203
204 cstatement:
205     END
206     | declare
207     | expression END
208     | VARIABLE '=' expression END {
209         if(checkDeclared($1) == 0) {
210             printf("\n%s Not Declared!\n", $1);
211         }
212         else {
213             setValue($1, $3);
214         }
215     }
216     | function_call END
217     | DISPLAY '(' VARIABLE ')' END {
218         if(checkDeclared($3)==0) {
219             printf("\nCan't print, Variable is not declared\n");
220         }
221         else {
222             int index = getValue($3);
223             if(type[index] == 0){
224                 printf("\nPrinting Value of the variable %s: %d\n", $3, store_int[index]);
225             }
226             else if(type[index] == 1){
227                 printf("\nPrinting Value of the variable %s: %f\n", $3, store_float[index]);
228             }
229             else if(type[index] == 2){
230                 printf("\nPrinting Value of the variable %s: %s\n", $3, store_String[index]);
231             }
232             else{
233                 printf("\nCan't Display Value as Variable is a Function!\n");
234             }
235         }
236     }
237     | if_condition '{' statements '}' {
238         printf("\nIf Block is Successfully Handled!\n");
239     }
240     | else_if_condition '{' statements '}' {
241         printf("\nElse If Block is Successfully Handled!\n");
242     }
243     | else_condition '{' statements '}' {

```

```

244     printf("\nElse Block is Successfully Handled!\n");
245 }
246 | for_start '(' for_loop ')' '{' statements '}' {
247     printf("\nFor Loop Execution Finshed!\n");
248 }
249 | while_start '(' while_loop ')' '{' statements '}' {
250     printf("\nWhile Loop Execution Finshed!\n");
251 }
252 | switch_start '(' switch_exp ')' '{' switch_statement '}' {
253     printf("\nSwitch Execution Finshed!\n");
254 }
255 ;
256
257 switch_start:
258     SWITCH {
259         printf("\nSwitch Case Started!\n");
260     }
261 ;
262
263 switch_exp :
264     expression {
265         switch_case = 0;
266         switch_var = atoi($1);
267     }
268 ;
269
270 switch_statement: /* NULL */
271     | switch_statement CASE expression ':' '{' statements '}' {
272         int x = atoi($3);
273         if(x == switch_var && switch_case == 0 ) {
274             printf("\nSwitch Case Executed is %d!\n", x);
275             switch_case = 1;
276         }
277         else {
278             printf("\nSwitch Case No: %d is Ignored!\n", x);
279         }
280     }
281     | switch_statement DEFAULT ':' '{' statements '}' {
282         if(switch_case == 0) {
283             switch_case = 1;
284             printf("\nSwitch Default Case is Executed!\n");
285         }
286     }
287 ;
288
289 while_loop:
290     VARIABLE loop_exp loop_assign {
291         if(checkDeclared($1) == 0) {
292             printf("\n%s Not Declared!\n", $1);
293         }
294         else {
295             printf("\nWhile Loop Variable Declaration is Correct!\n");
296         }
297     }
298
299 while_start:
300     WHILE {
301         printf("\nWhile Loop Started!\n");
302     }
303
304 for_start:
305     FOR {

```

```

306     printf("\nFor Loop Started!\n");
307 }
308
309 for_loop:
310 | VARIABLE '=' loop_assign ',' VARIABLE loop_exp loop_assign ',' VARIABLE f_state
loop_assign {
311     if(checkDeclared($1) == 0) {
312         printf("\n%s Not Declared!\n", $1);
313     }
314     if(strcmp($1, $5) == 0) {
315         if(strcmp($1, $9) == 0) {
316             printf("\nFor Loop Variable Declaration is Correct!\n");
317         }
318     }
319     else {
320         printf("\nDifferent Variables Used: %s %s %s\n", $1, $5, $9);
321     }
322 }
323
324 loop_assign:
325     NUMBER
326     | VARIABLE {
327         if(checkDeclared($1) == 0) {
328             printf("\n%s Not Declared!\n", $1);
329         }
330         else {
331             printf("\nVariable Correctly Assigned to Loop!\n");
332         }
333     }
334 ;
335
336 loop_exp:
337     LT
338     | GT
339     | GEQ
340     | LEQ
341     | EQ
342     | NEQ
343 ;
344
345 f_state:
346     FLINC {
347         printf("\nLoop is of Increasing Manner!\n");
348     }
349     | FLDEC {
350         printf("\nLoop is of Decreasing Manner!\n");
351     }
352 ;
353
354 if_condition:
355     IF '(' expression ')' {
356         int x = atoi($3);
357         if( x >= 1 ) {
358             store_if[if_pointer] = 1;
359             printf("\nIf Block is Executed!\n");
360         }
361         else {
362             printf("\nIf Block is Not Executed!\n");
363         }
364         if_pointer++;
365     }
366

```



```

367 else_if_condition:
368     ELSE_IF '(' expression ')' {
369         int x = atoi($3);
370         if( x >= 1 && store_if[if_pointer] == 0) {
371             store_if[if_pointer] = 1;
372             printf("\nElse If Block is Executed!\n");
373         }
374         else {
375             printf("\nElse If Block is Not Executed!\n");
376         }
377     }
378
379 else_condition:
380     ELSE {
381         if( store_if[if_pointer] == 0) {
382             store_if[if_pointer] = 1;
383             printf("\nElse Block is Executed!\n");
384         }
385         else {
386             printf("\nElse Block is Not Executed!\n");
387         }
388     }
389
390 function_call:
391     f_var '(' call_param ')' {
392         printf("\nValid Function Call!\n");
393     }
394     ;
395
396 f_var:
397     VARIABLE {
398         if(checkDeclared($1) == 0) {
399             printf("\n%s Function is Not Declared!\n", $1);
400         }
401         else {
402             printf("\n%s Function is Called!\n", $1);
403         }
404     }
405
406 call_param:
407     call_param ',' VARIABLE {
408         if(checkDeclared($3) == 0) {
409             printf("\n%s Variable is Not Declared\n", $3);
410         }
411         else {
412             printf("\n%s Passed as Parameter For Function!\n", $3);
413         }
414     }
415     | VARIABLE {
416         if(checkDeclared($1) == 0) {
417             printf("\n%s Variable is Not Declared\n", $1);
418         }
419         else {
420             printf("\n%s Passed as Parameter For Function!\n", $1);
421         }
422     }
423     ;
424     /* NULL */
425     ;
426
427 declare:

```

```

429     type id END { printf("\nValid Syntax For Variable Declaration!\n"); }
430     ;
431
432 type:
433     INT { var_type_pointer = 0; }
434     | FLOAT { var_type_pointer = 1; }
435     | STRING { var_type_pointer = 2; }
436     ;
437
438 id:
439     id ',' VARIABLE {
440         if(checkDeclared($3)==1) {
441             printf("\nDuplicate Declaration!\n");
442         }
443         else {
444             varAssign($3);
445         }
446     }
447     | id ',' VARIABLE '=' expression {
448         if(checkDeclared($3)==1) {
449             printf("\nDuplicate Declaration!\n");
450         }
451         else {
452             varAssign($3);
453             setValue($3, $5);
454         }
455     }
456     | VARIABLE {
457         if(checkDeclared($1)==1)
458             printf("\nDuplicate Declaration!\n");
459         else
460             varAssign($1);
461     }
462     | VARIABLE '=' expression {
463         if(checkDeclared($1)==1) {
464             printf("\nDuplicate Declaration!\n");
465         }
466         else {
467             varAssign($1);
468             setValue($1, $3);
469         }
470     }
471     ;
472
473 expression:
474     NUMBER {
475         $$ = malloc(20);
476         strcpy($$, $1);
477     }
478     | STR {
479         $$ = malloc(20);
480         strcpy($$, $1);
481     }
482     | VARIABLE {
483         $$ = malloc(20);
484         if(checkDeclared($1) == 0) {
485             sprintf($$, "%d", 0);
486             printf("\n%s Not Declared!\n", $1);
487         }
488         else {
489             int index = getValue($1);
490             if(type[index] == 0){

```

```

491         sprintf($$, "%d", store_int[index]);
492     }
493     else if(type[index] == 1){
494         sprintf($$, "%f", store_float[index]);
495     }
496     else if(type[index] == 2){
497         strcpy($$, store_String[index]);
498     }
499     else{
500         printf("\nCan't Process Value as Variable is a Function or Invalid!\n");
501     }
502 }
503 }
504 | expression '+' expression {
505     $$ = malloc(20);
506     float num1 = atof($1);
507     float num2 = atof($3);
508     float num3 = num1 + num2;
509     sprintf($$, "%f", num3);
510     printf("\nAdd Value: %f\n", num3);
511 }
512 | expression '-' expression {
513     $$ = malloc(20);
514     float num1 = atof($1);
515     float num2 = atof($3);
516     float num3 = num1 - num2;
517     sprintf($$, "%f", num3);
518     printf("\nSub Value: %f\n", num3);
519 }
520 | expression '*' expression {
521     $$ = malloc(20);
522     float num1 = atof($1);
523     float num2 = atof($3);
524     float num3 = num1 * num2;
525     sprintf($$, "%f", num3);
526     printf("\nMul Value: %f\n", num3);
527 }
528 | expression '/' expression {
529     $$ = malloc(20);
530     float num1 = atof($1);
531     float num2 = atof($3);
532     if(num2!=0.0) {
533         float num3 = num1 / num2;
534         sprintf($$, "%f", num3);
535         printf("\nDiv Value: %f\n", num3);
536     }
537     else {
538         sprintf($$, "%f", 0);
539         printf("\nDiv by Zero is Not Possible!\n");
540     }
541 }
542 | expression '^' expression {
543     $$ = malloc(20);
544     float num1 = atof($1);
545     float num2 = atof($3);
546     float num3 = pow(num1, num2);
547     sprintf($$, "%f", num3);
548     printf("\nPower Value: %f\n", num3);
549 }
550 | expression MOD expression {
551     $$ = malloc(20);
552     int num1 = atoi($1);

```



```

553     int num2 = atoi($3);
554     int num3 = num1 % num2;
555     sprintf($$, "%d", num3);
556     printf("\nRemainder Value: %d\n", num3);
557 }
558 | '(' expression ')' {
559     $$ = malloc(20);
560     strcpy($$, $2);
561 }
562 | expression LT expression {
563     $$ = malloc(20);
564     float num1 = atof($1);
565     float num2 = atof($3);
566     int num3 = num1 < num2;
567     sprintf($$, "%d", num3);
568     printf("\nLess Than Value: %d\n", num3);
569 }
570 | expression GT expression {
571     $$ = malloc(20);
572     float num1 = atof($1);
573     float num2 = atof($3);
574     int num3 = num1 > num2;
575     sprintf($$, "%d", num3);
576     printf("\nGreater Than Value: %d\n", num3);
577 }
578 | expression LEQ expression {
579     $$ = malloc(20);
580     float num1 = atof($1);
581     float num2 = atof($3);
582     int num3 = num1 <= num2;
583     sprintf($$, "%d", num3);
584     printf("\nLess Than or Equal To Value: %d\n", num3);
585 }
586 | expression GEQ expression {
587     $$ = malloc(20);
588     float num1 = atof($1);
589     float num2 = atof($3);
590     int num3 = num1 >= num2;
591     sprintf($$, "%d", num3);
592     printf("\nGreater Than or Equal To Value: %d\n", num3);
593 }
594 | expression EQ expression {
595     $$ = malloc(20);
596     float num1 = atof($1);
597     float num2 = atof($3);
598     int num3 = num1 == num2;
599     sprintf($$, "%d", num3);
600     printf("\nEqual To Value: %d\n", num3);
601 }
602 | expression NEQ expression {
603     $$ = malloc(20);
604     float num1 = atof($1);
605     float num2 = atof($3);
606     int num3 = num1 != num2;
607     sprintf($$, "%d", num3);
608     printf("\nNot Equal To Value: %d\n", num3);
609 }
610 | VARIABLE INC {
611     $$ = malloc(20);
612     if( checkDeclared($1) == 0) {
613         sprintf($$, "%d", 0);
614         printf("\n%s is Not Declared!\n", $1);

```

```

615     }
616     else {
617         int index = getValue($1);
618         if(type[index] == 0){
619             int tmp = store_int[index];
620             tmp = tmp+1;
621             store_int[index] = tmp;
622             sprintf($$, "%d", tmp);
623             printf("\nValue After Increment: %d\n", tmp);
624         }
625         else if(type[index] == 1){
626             float tmp = store_float[index];
627             tmp = tmp+1;
628             store_float[index] = tmp;
629             sprintf($$, "%f", tmp);
630             printf("\nValue After Increment: %f\n", tmp);
631         }
632         else{
633             printf("\nCan't Process Increment as Variable is a String or a Function!\n");
634         }
635     }
636 }
637 | VARIABLE DEC {
638     $$ = malloc(20);
639     if( checkDeclared($1) == 0) {
640         sprintf($$, "%d", 0);
641         printf("\n%s Not Declared!\n", $1);
642     }
643     else {
644         int index = getValue($1);
645         if(type[index] == 0){
646             int tmp = store_int[index];
647             tmp = tmp-1;
648             store_int[index] = tmp;
649             sprintf($$, "%d", tmp);
650             printf("\nValue After Decrement: %d\n", tmp);
651         }
652         else if(type[index] == 1){
653             float tmp = store_float[index];
654             tmp = tmp-1;
655             store_float[index] = tmp;
656             sprintf($$, "%f", tmp);
657             printf("\nValue After Decrement: %f\n", tmp);
658         }
659         else{
660             printf("\nCan't Process Decrement as Variable is a String or a Function!\n");
661         }
662     }
663 }
664 | NOT VARIABLE {
665     $$ = malloc(20);
666     if( checkDeclared($2) == 0) {
667         sprintf($$, "%d", 0);
668         printf("\n%s Not Declared!\n", $2);
669     }
670     else {
671         int index = getValue($2);
672         if(type[index] == 0){
673             int tmp = store_int[index];
674             tmp = !tmp;
675             store_int[index] = tmp;
676             sprintf($$, "%d", tmp);

```

```

677         printf("\nValue After NOT Operation: %d\n", tmp);
678     }
679     else if(type[index] == 1){
680         int tmp = store_float[index];
681         tmp = !tmp;
682         store_float[index] = tmp;
683         sprintf($$, "%d", tmp);
684         printf("\nValue After NOT Operation: %d\n", tmp);
685     }
686     else{
687         printf("\nCan't Process NOT Operation as Variable is a String or a Function!\n"
);
688     }
689 }
690 }
691 | SIN '(' expression ')' {
692     $$ = malloc(20);
693     float x = atof($3);
694     printf("\nValue of Sin(%f): %lf\n", x, sin(x*3.1416/180));
695     sprintf($$, "%lf", sin(x*3.1416/180));
696 }
697 | COS '(' expression ')' {
698     $$ = malloc(20);
699     float x = atof($3);
700     printf("\nValue of Cos(%f): %lf\n", x, cos(x*3.1416/180));
701     sprintf($$, "%lf", cos(x*3.1416/180));
702 }
703 | TAN '(' expression ')' {
704     $$ = malloc(20);
705     float x = atof($3);
706     printf("\nValue of Tan(%f): %lf\n", x, tan(x*3.1416/180));
707     sprintf($$, "%lf", tan(x*3.1416/180));
708 }
709 | LOG '(' expression ')' {
710     $$ = malloc(20);
711     float x = atof($3);
712     printf("\nValue of Log(%f): %lf\n", x, (log(x*1.0)/log(10.0)));
713     sprintf($$, "%lf", (log(x*1.0)/log(10.0)));
714 }
715 | LN '(' expression ')' {
716     $$ = malloc(20);
717     float x = atof($3);
718     printf("\nValue of Ln(%f): %lf\n", x, (log(x)));
719     sprintf($$, "%lf", (log(x)));
720 }
721 | ODD EVEN '(' expression ')' {
722     $$ = malloc(20);
723     int x = atoi($3);
724     if(x%2==0) {
725         sprintf($$, "%d", 0);
726         printf("\n%d is An Even Number\n", x);
727     }
728     else {
729         sprintf($$, "%d", 1);
730         printf("\n%d is An Odd Number\n", x);
731     }
732 }
733 | FACTORIAL '(' expression ')' {
734     $$ = malloc(20);
735     int ans = 1;
736     int i;
737     int x = atoi($3);

```

```

738     for(i=1; i<=x; i++) {
739         ans = ans*i;
740     }
741     printf("\nFactorial of %d is: %d\n", x, ans);
742     sprintf($$, "%d", ans);
743 }
744 | MAX '(' expression ',' expression ')' {
745     $$ = malloc(20);
746     float num1 = atof($3);
747     float num2 = atof($5);
748     if( num1 < num2 ) {
749         sprintf($$, "%f", num2);
750         printf("\nMax Number Between %f and %f is: %f\n", num1, num2, num2);
751     }
752     else {
753         sprintf($$, "%f", num1);
754         printf("\nMax Number Between %f and %f is: %f\n", num1, num2, num1);
755     }
756 }
757 | MIN '(' expression ',' expression ')' {
758     $$ = malloc(20);
759     float num1 = atof($3);
760     float num2 = atof($5);
761     if( num1 < num2 ) {
762         sprintf($$, "%f", num1);
763         printf("\nMin Number Between %f and %f is: %f\n", num1, num2, num1);
764     }
765     else {
766         sprintf($$, "%f", num2);
767         printf("\nMin Number Between %f and %f is: %f\n", num1, num2, num2);
768     }
769 }
770 | PRIME '(' expression ')' {
771     $$ = malloc(20);
772     int x = atoi($3);
773     int ck = 0;
774     int i;
775     for(i=2; i*i<=x; i++) {
776         if( x%i == 0 ) {
777             ck = 1;
778             break;
779         }
780     }
781     if(ck || x==1) {
782         sprintf($$, "%d", 0);
783         printf("\n%d is Not A Prime Number\n", x);
784     }
785     else {
786         sprintf($$, "%d", 1);
787         printf("\n%d is A Prime Number\n", x);
788     }
789 }
790 ;
791
792 %%
793
794 int yywrap()
795 {
796     return 1;
797 }
798
799 int main()

```

```
800 {
801     yyin = freopen("input.txt","r",stdin);
802     yyout = freopen("output.txt","w",stdout);
803     yyparse();
804     return 0;
805 }
806
807 int yyerror(char *s)
808 {
809     printf( "%s\n", s);
810 }
```

## Output File:

### **output.txt**

```
Header File Found!

New Function Declared With Name: Abc

New Variable Declared With Name: i and Type: Int

Valid Function Parameter Declaration!

New Variable Declared With Name: j and Type: Int

Valid Function Parameter Declaration!

New Value Assigned to Variable Name: i and Value: 5

User Defined Function Ended!

Main Function Declared!

New Variable Declared With Name: a and Type: Int

New Value Assigned to Variable Name: a and Value: 6

Valid Syntax For Variable Declaration!

New Variable Declared With Name: b and Type: Int

New Value Assigned to Variable Name: b and Value: 6

Valid Syntax For Variable Declaration!

New Variable Declared With Name: c and Type: Int

New Value Assigned to Variable Name: c and Value: 7

New Variable Declared With Name: d and Type: Int

New Value Assigned to Variable Name: d and Value: 5

Valid Syntax For Variable Declaration!

New Variable Declared With Name: x and Type: Float

New Value Assigned to Variable Name: x and Value: 7.500000

Valid Syntax For Variable Declaration!

New Variable Declared With Name: y and Type: String

New Value Assigned to Variable Name: y and Value: "abc"

Valid Syntax For Variable Declaration!

New Value Assigned to Variable Name: c and Value: 3

New Value Assigned to Variable Name: b and Value: 8

Add Value: 11.000000

New Value Assigned to Variable Name: a and Value: 11

Sub Value: 5.000000

New Value Assigned to Variable Name: a and Value: 5

Remainder Value: 2

New Value Assigned to Variable Name: a and Value: 2
```



Mul Value: 24.000000  
New Value Assigned to Variable Name: a and Value: 24  
Div Value: 2.666667  
New Value Assigned to Variable Name: a and Value: 2  
Power Value: 512.000000  
New Value Assigned to Variable Name: a and Value: 512  
Div Value: 2.666667  
Add Value: 10.666667  
Add Value: 13.666667  
New Value Assigned to Variable Name: a and Value: 13  
Greater Than Value: 1  
New Value Assigned to Variable Name: a and Value: 1  
Less Than Value: 0  
New Value Assigned to Variable Name: a and Value: 0  
Equal To Value: 0  
New Value Assigned to Variable Name: a and Value: 0  
Not Equal To Value: 1  
New Value Assigned to Variable Name: a and Value: 1  
Less Than or Equal To Value: 0  
New Value Assigned to Variable Name: a and Value: 0  
Greater Than or Equal To Value: 1  
New Value Assigned to Variable Name: a and Value: 1  
Value After Increment: 2  
New Value Assigned to Variable Name: c and Value: 2  
Value After Decrement: 1  
New Value Assigned to Variable Name: c and Value: 1  
Value After NOT Operation: 0  
New Value Assigned to Variable Name: c and Value: 0  
Value of Sin(8.000000): 0.139173  
New Value Assigned to Variable Name: c and Value: 0  
Value of Cos(8.000000): 0.990268  
New Value Assigned to Variable Name: c and Value: 0  
Value of Tan(8.000000): 0.140541  
New Value Assigned to Variable Name: c and Value: 0  
Value of Log(8.000000): 0.903090  
New Value Assigned to Variable Name: c and Value: 0

Value of  $\ln(8.000000)$ : 2.079442

New Value Assigned to Variable Name: c and Value: 2

8 is An Even Number

Factorial of 8 is: 40320

Max Number Between 8.000000 and 2.000000 is: 8.000000

New Value Assigned to Variable Name: a and Value: 8

Min Number Between 8.000000 and 2.000000 is: 2.000000

New Value Assigned to Variable Name: a and Value: 2

8 is Not A Prime Number

Abc Function is Called!

b Passed as Parameter For Function!

c Passed as Parameter For Function!

Valid Function Call!

Printing Value of the variable a: 2

Equal To Value: 0

If Block is Not Executed!

Printing Value of the variable a: 2

If Block is Successfully Handled!

Equal To Value: 1

Else If Block is Executed!

Printing Value of the variable a: 2

Else If Block is Successfully Handled!

Else Block is Not Executed!

Printing Value of the variable a: 2

Else Block is Successfully Handled!

New Variable Declared With Name: l and Type: Int

Valid Syntax For Variable Declaration!

For Loop Started!

Variable Correctly Assigned to Loop!

Loop is of Increasing Manner!

For Loop Variable Declaration is Correct!

Printing Value of the variable b: 8

For Loop Execution Finshed!

While Loop Started!

Variable Correctly Assigned to Loop!

While Loop Variable Declaration is Correct!



```
Printing Value of the variable c: 2
While Loop Execution Finshed!
Switch Case Started!
Printing Value of the variable a: 2
Switch Case No: 8 is Ignored!
Printing Value of the variable b: 8
Switch Default Case is Executed!
Switch Execution Finshed!
Single Line Comment
Multiple Line Comment
Program Successfully Ended!
```

## **Limitations:**

- ✓ Can't handle operations involving Strings
- ✓ If/Else if/Else block statements execute regardless of it matched or not
- ✓ Switch Case block statements execute regardless of it matched or not
- ✓ Function call execution is not possible
- ✓ Loop statements do not actually loop
- ✓ User defined functions can only be declared before the main function
- ✓ Print function only prints variables
- ✓ Function parameter variables need to be unique and can't be declared again in main function or any other statements
- ✓ Variable that has declared once can not be declared again regardless of the block it was declared in
- ✓ Most of the calculations are handled in float manner

## **Discussion:**

The implementation of Flex and Bison in this project proved instrumental in achieving efficient lexical analysis and parsing. Flex's ability to generate rapid lexical analyzers streamlined the recognition of patterns within the input, while Bison facilitated the construction of a parser based on a well-defined context-free grammar. The integration of these tools showcased a seamless flow of control between the lexer and parser, enabling the comprehensive analysis of a new syntax-based programming language. Challenges encountered during testing were effectively addressed, leading to a robust language processing system. The results underscore the significance of Flex and Bison in the development of language processors, offering insights into their combined strength in handling complex parsing tasks.

## **Conclusion:**

In conclusion, the integration of Flex and Bison in this project has demonstrated their pivotal role in efficient lexical analysis and parsing. Flex's swift pattern recognition, coupled with Bison's context-free grammar parsing capabilities, has yielded a robust language processing system. The successful handling of a new syntax-based programming language underscores the effectiveness of these tools in constructing language processors. As we reflect on the project, it is evident that Flex and Bison stand as indispensable assets in the development of parsers, offering a powerful tandem for tackling intricate syntax analysis tasks.

## **Reference:**

- ✓ Lab Lectures
- ✓ Lab Codes
- ✓ Bison Manual - <https://www.gnu.org/software/bison/manual/bison.html>
- ✓ Flex Manual - <https://westes.github.io/flex/manual/>
- ✓ ChatGPT (Used for Debugging)