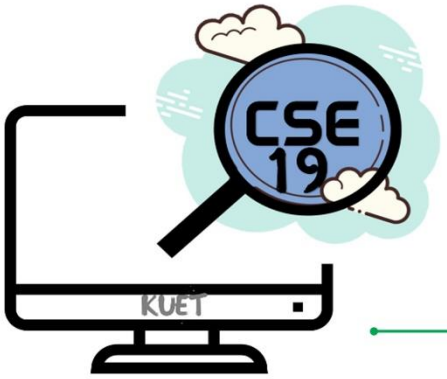
Khulna University of Engineering and Technology

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

Submitted From	Submitted to
Name: Naimur Rahman	Nazia Jahan Khan Chowdhury Assistant Professor
Roll: 1907031	Dipannita Biswas Lecturer
Section: A	Dept of CSE, KUET
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	\/*([^*]|*[^\/])**\/

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.l

```
digits [0-9]
Datatype "int"|"float"|"void"|"string"
Identifiers [a-zA-Z][a-zA-Z0-9]*
S_comment \/\/[^\n]*
M_comment \/\*([^*]|\*[^\/])*\*\/

%{
    #include "1907031.tab.h"
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    int varindex(char *var);
    extern int yylex();
    extern int yyparse();
    extern FILE *yyin;
    extern FILE *yyout;
    int yyerror(char *s);
    //int lineNo = 1;
%}

%%

{S_comment} { printf("\nSingle Line Comment\n"); }
{M_comment} { printf("\nMultiple Line Comment\n"); }

"int" { return INT; }
"float" { return FLOAT; }
"string" { return STRING; }

"(" { return '('; }
")" { return ')'; }
"<" { return '<'; }
">" { return '>'; }
"{" { return '{'; }
"}" { return '}'; }
";" { return END; }
"," { return ','; }
"=" { return '='; }
":" { return ':'; }

"+" { return '+'; }
"-" { return '-'; }
"*" { return '*'; }
"/" { return '/'; }
"^" { return '^'; }
"mod" { return MOD; }

"less" { return LT; }
"great" { return GT; }
"equal" { return EQ; }
"great_eq" { return GEQ; }
"less_eq" { return LEQ; }
"not_eq" { return NEQ; }

"++" { return INC; }
"--" { return DEC; }
"!" { return NOT; }

"sin" { return SIN; }
```

```
"cos" { return COS; }
"tan" { return TAN; }
"ln" { return LN; }
"log" { return LOG; }
"isOddEven" { return ODDEVEN; }
"factorial" { return FACTORIAL; }
"max" { return MAX; }
"min" { return MIN; }
"isPrime" { return PRIME; }

"print" { return DISPLAY; }

"if" { return IF; }
"else_if" { return ELSE_IF; }
"else" { return ELSE; }

"for" { return FOR; }
"inc" { return FLINC; }
"dec" { return FLDEC; }
"while" { return WHILE; }

"case" { return CASE; }
"switch" { return SWITCH; }
"default" { return DEFAULT; }

"-"?{digits}+ {
    yylval.string = strdup(yytext);
    return NUMBER;
}

"-"?({digits}+)?"."{digits}+ {
    yylval.string = strdup(yytext);
    return NUMBER;
}

"\""[^"]*"\"" {
    yylval.string = strdup(yytext);
    return STR;
}

{Datatype}[ ]+"main" { return MAIN; }
"#include" { return IMPORT; }
{Identifiers}".h" { return HEADER; }

"function" { return DEF; }

{Identifiers} {
    yylval.string = strdup(yytext);
    return VARIABLE;
}

[ \t\n]*

. {yyerror("Unknown Character.\n");}

%%
```

Bison File:

1907031.y

```
%{
#include<stdio.h>
#include <math.h>
#include<stdlib.h>
#include<string.h>
extern FILE *yyin;
extern FILE *yyout;
int yylex();
int yyerror(char *s);

// Symbol Table Arrays
char var_name[1000][100];
int store_int[1000];
float store_float[1000];
char store_String[1000][100];
int type[1000];
int var_type_pointer = 0; // 0 = int, 1 = float, 2 = string, 3 = function

// Conditional Statement Variables
int if_pointer = 0;
int store_if[1000];

// Switch Handling Variables
int switch_var = 0;
int switch_case = 0;

// Variable Counter
int var_cnt = 0;

// Variable Declaration Check
int checkDeclared(char *s){
    int i;
    for(i=0; i<var_cnt; i++){
        if(strcmp(var_name[i], s) == 0)
                return 1;
    }
    return 0;
}

// New Variable Declaration
int varAssign(char *s){
    if(checkDeclared(s) == 1){
        return 0;
    }
    strcpy(var_name[var_cnt], s);
    store_int[var_cnt] = 0;
    store_float[var_cnt] = 0.0;
    strcpy(store_String[var_cnt], "");
    type[var_cnt] = var_type_pointer;
    char name[10];
    if(var_type_pointer == 0) {
        strcpy(name, "Int");
    }
    else if(var_type_pointer == 1) {
        strcpy(name, "Float");
    }
    else if(var_type_pointer == 2) {
        strcpy(name, "String");
    }
```

```
    printf("\nNew Variable Declared With Name: %s and Type: %s\n", var_name[var_cnt], name);
    var_cnt++;
    return 1;
}

// New Function Declaration
int functionAssign(char *s){
    if(checkDeclared(s) == 1){
        return 0;
    }
    strcpy(var_name[var_cnt], s);
    store_int[var_cnt] = 0;
    store_float[var_cnt] = 0.0;
    strcpy(store_String[var_cnt], "");
    type[var_cnt] = 3;
    printf("\nNew Function Declared With Name: %s\n", var_name[var_cnt]);
    var_cnt++;
    return 1;
}

// Assigning Value to Variable
int setValue(char *s, char* val){
    if(checkDeclared(s) == 0){
        return 0;
    }
    int ok=0, i;
    for(i=0; i<var_cnt; i++){
        if(strcmp(var_name[i], s) == 0){
            ok = i;
            break;
        }
    }
    if(type[ok] == 0){
        store_int[ok] = atoi(val);
        printf("\nNew Value Assigned to Variable Name: %s and Value: %d\n", var_name[ok],
store_int[ok]);
    }
    else if(type[ok] == 1){
        store_float[ok] = atof(val);
        printf("\nNew Value Assigned to Variable Name: %s and Value: %f\n", var_name[ok],
store_float[ok]);
    }
    else if(type[ok] == 2){
        strcpy(store_String[ok], val);
        //store_String[ok] = val;
        printf("\nNew Value Assigned to Variable Name: %s and Value: %s\n", var_name[ok],
store_String[ok]);
    }
    else{
        printf("\nCan't Assign Value as Variable is a Function!\n");
    }
    return 1;
}

// Get Variable Value
int getValue(char *s){
    int pos=-1;
    int i;
    for(i=0; i<var_cnt; i++){
        if(strcmp(var_name[i], s) == 0){
            pos=i;
            break;
        }
```

```
    }
    return pos;
}
%}

%union
{
    int num;
    float flt;
    char* string;
};

%token END INT FLOAT STRING MOD
%token LT GT GEQ LEQ EQ NEQ
%token <string> VARIABLE
%token <string> NUMBER
%type <string> expression
%token <string> STR
%token IMPORT HEADER MAIN
%token INC DEC NOT
%token SIN COS LOG TAN LN
%token ODDEVEN FACTORIAL MAX MIN PRIME
%token DEF DISPLAY
%token IF ELSE_IF ELSE
%token FOR FLINC FLDEC WHILE
%token CASE SWITCH DEFAULT

%left LT GT GEQ LEQ EQ NEQ
%left '+' '-'
%left '*' '/' MOD
%left '^'

%%

program:
    import func main '(' ')' '{' statements '}' { printf("\nProgram Successfully Ended!\n"); }
    | /* NULL */
    ;

main:
    MAIN { printf("\nMain Function Declared!\n"); }

import: /* NULL */
    | import IMPORT '<' HEADER '>' { printf("\nHeader File Found!\n"); }
    ;

func:
    func_head '(' param ')' '{' statements '}' {
        printf("\nUser Defined Function Ended!\n");
    }
    | /* NULL */
    ;

func_head:
    DEF VARIABLE {
        if(checkDeclared($2)==1) {
            printf("\nDuplicate Function Name!\n");
        }
        else {
            functionAssign($2);
        }
    }
```

```
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 
```

```
else_if_condition:
    ELSE_IF '(' expression ')' {
        int x = atoi($3);
        if( x >= 1 && store_if[if_pointer] == 0) {
            store_if[if_pointer] = 1;
            printf("\nElse If Block is Executed!\n");
        }
        else {
            printf("\nElse If Block is Not Executed!\n");
        }
    }

else_condition:
    ELSE {
        if( store_if[if_pointer] == 0) {
            store_if[if_pointer] = 1;
            printf("\nElse Block is Executed!\n");
        }
        else {
            printf("\nElse Block is Not Executed!\n");
        }
    }

function_call:
    f_var '(' call_param ')' {
        printf("\nValid Function Call!\n");
    }
    ;

f_var:
    VARIABLE {
        if(checkDeclared($1) == 0) {
            printf("\n%s Function is Not Declared!\n", $1);
        }
        else {
            printf("\n%s Function is Called!\n", $1);
        }
    }

call_param:
    call_param ',' VARIABLE {
        if(checkDeclared($3) == 0) {
            printf("\n%s Variable is Not Declared\n", $3);
        }
        else {
            printf("\n%s Passed as Parameter For Function!\n", $3);

        }
    }
    | VARIABLE {
        if(checkDeclared($1) == 0) {
            printf("\n%s Variable is Not Declared\n", $1);
        }
        else {
            printf("\n%s Passed as Parameter For Function!\n", $1);

        }
    }
    | /* NULL */
    ;

declare:
```

```
    type id END { printf("\nValid Syntax For Variable Declaration!\n"); }
    ;

type:
    INT { var_type_pointer = 0; }
    | FLOAT  { var_type_pointer = 1; }
    | STRING  { var_type_pointer = 2; }
    ;

id:
    id ',' VARIABLE {
        if(checkDeclared($3)==1) {
            printf("\nDuplicate Declaration!\n");
        }
        else {
            varAssign($3);
        }
    }
    | id ',' VARIABLE '=' expression {
        if(checkDeclared($3)==1) {
            printf("\nDuplicate Declaration!\n");
        }
        else {
            varAssign($3);
            setValue($3, $5);
        }
    }
    | VARIABLE {
        if(checkDeclared($1)==1)
            printf("\nDuplicate Declaration!\n");
        else
            varAssign($1);
    }
    | VARIABLE '=' expression {
        if(checkDeclared($1)==1) {
            printf("\nDuplicate Declaration!\n");
        }
        else {
            varAssign($1);
            setValue($1, $3);
        }
    }
    ;

expression:
    NUMBER {
        $$ = malloc(20);
        strcpy($$, $1);
    }
    | STR {
        $$ = malloc(20);
        strcpy($$, $1);
    }
    | VARIABLE {
        $$ = malloc(20);
        if(checkDeclared($1) == 0) {
            sprintf($$, "%d", 0);
            printf("\n%s Not Declared!\n", $1);
        }
        else {
            int index = getValue($1);
            if(type[index] == 0){
```

```
                sprintf($$, "%d", store_int[index]);
            }
            else if(type[index] == 1){
                sprintf($$, "%f", store_float[index]);
            }
            else if(type[index] == 2){
                strcpy($$, store_String[index]);
            }
            else{
                printf("\nCan't Process Value as Variable is a Function or Invalid!\n");
            }
        }
    }
    | expression '+' expression {
        $$ = malloc(20);
        float num1 = atof($1);
        float num2 = atof($3);
        float num3 = num1 + num2;
        sprintf($$, "%f", num3);
        printf("\nAdd Value: %f\n", num3);
    }
    | expression '-' expression {
        $$ = malloc(20);
        float num1 = atof($1);
        float num2 = atof($3);
        float num3 = num1 - num2;
        sprintf($$, "%f", num3);
        printf("\nSub Value: %f\n", num3);
    }
    | expression '*' expression {
        $$ = malloc(20);
        float num1 = atof($1);
        float num2 = atof($3);
        float num3 = num1 * num2;
        sprintf($$, "%f", num3);
        printf("\nMul Value: %f\n", num3);
    }
    | expression '/' expression {
        $$ = malloc(20);
        float num1 = atof($1);
        float num2 = atof($3);
        if(num2!=0.0) {
            float num3 = num1 / num2;
            sprintf($$, "%f", num3);
            printf("\nDiv Value: %f\n", num3);
        }
        else {
            sprintf($$, "%f", 0);
            printf("\nDiv by Zero is Not Possible!\n");
        }
    }
    | expression '^' expression {
        $$ = malloc(20);
        float num1 = atof($1);
        float num2 = atof($3);
        float num3 = pow(num1, num2);
        sprintf($$, "%f", num3);
        printf("\nPower Value: %f\n", num3);
    }
    | expression MOD expression {
        $$ = malloc(20);
        int num1 = atoi($1);
```

```
        int num2 = atoi($3);
        int num3 = num1 % num2;
        sprintf($$, "%d", num3);
        printf("\nRemainder Value: %d\n", num3);
    }
    | '(' expression ')' {
        $$ = malloc(20);
        strcpy($$, $2);
    }
    | expression LT expression {
        $$ = malloc(20);
        float num1 = atof($1);
        float num2 = atof($3);
        int num3 = num1 < num2;
        sprintf($$, "%d", num3);
        printf("\nLess Than Value: %d\n", num3);
    }
    | expression GT expression {
        $$ = malloc(20);
        float num1 = atof($1);
        float num2 = atof($3);
        int num3 = num1 > num2;
        sprintf($$, "%d", num3);
        printf("\nGreater Than Value: %d\n", num3);
    }
    | expression LEQ expression {
        $$ = malloc(20);
        float num1 = atof($1);
        float num2 = atof($3);
        int num3 = num1 <= num2;
        sprintf($$, "%d", num3);
        printf("\nLess Than or Equal To Value: %d\n", num3);
    }
    | expression GEQ expression {
        $$ = malloc(20);
        float num1 = atof($1);
        float num2 = atof($3);
        int num3 = num1 >= num2;
        sprintf($$, "%d", num3);
        printf("\nGreater Than or Equal To Value: %d\n", num3);
    }
    | expression EQ expression {
        $$ = malloc(20);
        float num1 = atof($1);
        float num2 = atof($3);
        int num3 = num1 == num2;
        sprintf($$, "%d", num3);
        printf("\nEqual To Value: %d\n", num3);
    }
    | expression NEQ expression {
        $$ = malloc(20);
        float num1 = atof($1);
        float num2 = atof($3);
        int num3 = num1 != num2;
        sprintf($$, "%d", num3);
        printf("\nNot Eqaul To Value: %d\n", num3);
    }
    | VARIABLE INC {
        $$ = malloc(20);
        if( checkDeclared($1) == 0) {
            sprintf($$, "%d", 0);
            printf("\n%s is Not Declared!\n", $1);
```

```
        }
        else {
            int index = getValue($1);
            if(type[index] == 0){
                int tmp = store_int[index];
                tmp = tmp+1;
                store_int[index] = tmp;
                sprintf($$, "%d", tmp);
                printf("\nValue After Increment: %d\n", tmp);
            }
            else if(type[index] == 1){
                float tmp = store_float[index];
                tmp = tmp+1;
                store_float[index] = tmp;
                sprintf($$, "%f", tmp);
                printf("\nValue After Increment: %f\n", tmp);
            }
            else{
                printf("\nCan't Process Increament as Variable is a String or a Function!\n");
            }
        }
    }
    | VARIABLE DEC {
        $$ = malloc(20);
        if( checkDeclared($1) == 0) {
            sprintf($$, "%d", 0);
            printf("\n%s Not Declared!\n", $1);
        }
        else {
            int index = getValue($1);
            if(type[index] == 0){
                int tmp = store_int[index];
                tmp = tmp-1;
                store_int[index] = tmp;
                sprintf($$, "%d", tmp);
                printf("\nValue After Decrement: %d\n", tmp);
            }
            else if(type[index] == 1){
                float tmp = store_float[index];
                tmp = tmp-1;
                store_float[index] = tmp;
                sprintf($$, "%f", tmp);
                printf("\nValue After Decrement: %f\n", tmp);
            }
            else{
                printf("\nCan't Process Decrement as Variable is a String or a Function!\n");
            }
        }
    }
    | NOT VARIABLE {
        $$ = malloc(20);
        if( checkDeclared($2) == 0) {
            sprintf($$, "%d", 0);
            printf("\n%s Not Declared!\n", $2);
        }
        else {
            int index = getValue($2);
            if(type[index] == 0){
                int tmp = store_int[index];
                tmp = !tmp;
                store_int[index] = tmp;
                sprintf($$, "%d", tmp);
```

```
                printf("\nValue After NOT Operation: %d\n", tmp);
            }
            else if(type[index] == 1){
                int tmp = store_float[index];
                tmp = !tmp;
                store_float[index] = tmp;
                sprintf($$, "%d", tmp);
                printf("\nValue After NOT Operation: %d\n", tmp);
            }
            else{
                printf("\nCan't Process NOT Operation as Variable is a String or a Function!\n"
);
            }
        }
    }
    | SIN '(' expression ')' {
        $$ = malloc(20);
        float x = atof($3);
        printf("\nValue of Sin(%f): %lf\n", x, sin(x*3.1416/180));
        sprintf($$, "%lf", sin(x*3.1416/180));
    }
    | COS '(' expression ')' {
        $$ = malloc(20);
        float x = atof($3);
        printf("\nValue of Cos(%f): %lf\n", x, cos(x*3.1416/180));
        sprintf($$, "%lf", cos(x*3.1416/180));
    }
    | TAN '(' expression ')' {
        $$ = malloc(20);
        float x = atof($3);
        printf("\nValue of Tan(%f): %lf\n", x, tan(x*3.1416/180));
        sprintf($$, "%lf", tan(x*3.1416/180));
    }
    | LOG '(' expression ')' {
        $$ = malloc(20);
        float x = atof($3);
        printf("\nValue of Log(%f): %lf\n", x, (log(x*1.0)/log(10.0)));
        sprintf($$, "%lf", (log(x*1.0)/log(10.0)));
    }
    | LN '(' expression ')' {
        $$ = malloc(20);
        float x = atof($3);
        printf("\nValue of Ln(%f): %lf\n", x, (log(x)));
        sprintf($$, "%lf", (log(x)));
    }
    | ODDEVEN '(' expression ')' {
        $$ = malloc(20);
        int x = atoi($3);
        if(x%2==0) {
            sprintf($$, "%d", 0);
            printf("\n%d is An Even Number\n", x);
        }
        else {
            sprintf($$, "%d", 1);
            printf("\n%d is An Odd Number\n", x);
        }
    }
    | FACTORIAL '(' expression ')' {
        $$ = malloc(20);
        int ans = 1;
        int i;
        int x = atoi($3);
```

```
        for(i=1; i<=x; i++) {
            ans = ans*i;
        }
        printf("\nFactorial of %d is: %d\n", x, ans);
        sprintf($$, "%d", ans);
    }
    | MAX '(' expression ',' expression ')' {
        $$ = malloc(20);
        float num1 = atof($3);
        float num2 = atof($5);
        if( num1 < num2 ) {
            sprintf($$, "%f", num2);
            printf("\nMax Number Between %f and %f is: %f\n", num1, num2, num2);
        }
        else {
            sprintf($$, "%f", num1);
            printf("\nMax Number Between %f and %f is: %f\n", num1, num2, num1);
        }
    }
    | MIN '(' expression ',' expression ')' {
        $$ = malloc(20);
        float num1 = atof($3);
        float num2 = atof($5);
        if( num1 < num2 ) {
            sprintf($$, "%f", num1);
            printf("\nMin Number Between %f and %f is: %f\n", num1, num2, num1);
        }
        else {
            sprintf($$, "%f", num2);
            printf("\nMin Number Between %f and %f is: %f\n", num1, num2, num2);
        }
    }
    | PRIME '(' expression ')' {
        $$ = malloc(20);
        int x = atoi($3);
        int ck = 0;
        int i;
        for(i=2; i*i<=x; i++) {
            if( x%i == 0 ) {
                ck = 1;
                break;
            }
        }
        if(ck || x==1) {
            sprintf($$, "%d", 0);
            printf("\n%d is Not A Prime Number\n", x);
        }
        else {
            sprintf($$, "%d", 1);
            printf("\n%d is A Prime Number\n", x);
        }
    }
    ;

%%

int yywrap()
{
    return 1;
}

int main()
```

```
{
    yyin = freopen("input.txt","r",stdin);
    yyout = freopen("output.txt","w",stdout);
    yyparse();
    return 0;
}

int yyerror(char *s)
{
    printf( "%s\n", s);
}
```

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 Eqaul 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)