KHULNA UNIVERSITY OF ENGINEERING & TECHNOLOGY

Department of Computer Science and Technology

Title: Designing a simple compiler using Flex and Bison

Course Title: Compiler Design Laboratory

Submitted by:

Md. Mashrur Alam

Roll: 1807001

Year: 3rd

Semester: 2nd

Department of Computer Science and Engineering

Khulna University of Engineering and Technology, Khulna

Submission date: 19/12/2022

Objectives:

After the completion of this project we will be able to

- Gather knowledge about how a compiler works
- Learn about Flex and Bison
- Learn how to create tokens from an input source using Flex
- Learn how to use those tokens inside a CFG and parse the input source using Bison
- Create semantic rules for the CFG
- Design a compiler for a new language using Flex and Bison

Introduction:

Compiler is a special program that processes statements written in a particular programming language and turns them into machine language or "code" that a computer's processor uses. We normally write our programming statements in a particular language to an editor then the source code is run to appropriate language compiler. Compiler makes the source code file to an executable file.

Flex and Bison:

Flex or Fast Lexical Analyzer Generator is a tool that generates tokens by pattern matching. It reads the input stream and produces the source code as output through implementing the lexical analyzer in the C program.

Bison is a parser generator. It parses the source program using the tokens created by Flex and generates a parse tree. Semantic rules used in the CFG give meaning to the relation between tokens.

Using Flex and Bison:

These commands must be used to run bison and flex from a windows command prompt or linux terminal:

- bison -d file_name.y
- flex file_name.l
- gcc lex.yy.c file_name.tab.c -o obj
- obj (windows) or ./obj (linux)

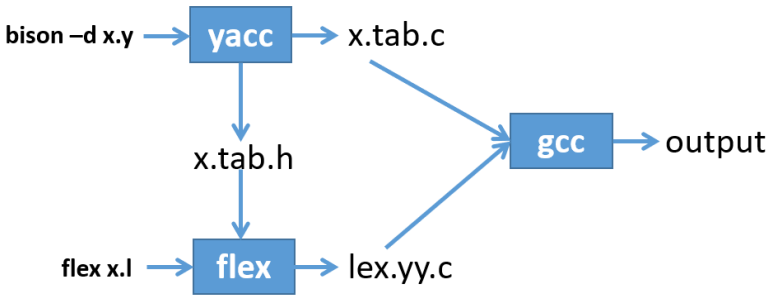

Fig-1: Workflow of Flex and Bison

Tokens used:

IDATA, VARIABLE, IF, ELIF, ELSE, MAIN, INT, FLOAT, SS, SE, SWITCH, CASE, DEFAULT, BREAK, PRIME, FOR, OUT, CMNT, WHILE, INC, DEC, ADD, SUB, MUL, DIV, POW, MOD, SQRT, GCD, LCM, FIB, EQUALS, SIN, COS, TAN, LOG, LN.

Project Manual:

Serial No.	Definition	Keyword and example
1.	Including library header files	#insert stdio.h #insert math.h
2.	Comments	!!comments
3.	Declaring variables	NUMBER a,b,c,d;
4.	Assigning values to variables	a EQUALS 10;
5.	Declaring main funciton	master() < >
6.	Input and output	IN(expression);
7.	Integer Data type	NUMBER
8.	Float Data type	FNUMBER
9.	Summation operation	a EQUALS b ADD c;
10.	Subtracion operation	a EQUALS b SUB c;

11.	Multiplication operation	a EQUALS b MUL c;
12.	Division operation	a EQUALS b DIV c;
13.	Modulo operation	a EQUALS b MOD c;
14.	Comparison operation	(a ILT b); (a IGT b);
15.	Increment operation	a INC;
16.	Decrement operation	a DEC;
17.	Break operation	END;
18.	If … else block	HOY(b IGT a) < b SUB a; > NAHOLE < a SUB b; >
19.	Switch case	SHIFT (2) < VALUE 1 : a ADD b; END; ; VALUE 2 : b ADD c; END; ; VALUE 3 : a ADD c; END; DEFAULT : c ADD 4; END; >
20.	For loop	JODI(a EQUALS 3; a ILT c) < 7 SUB 8; >
21.	While loop	JOKHON (6 ILT 8)

		< c EQUALS c ADD 1; >
22.	Return a value	PASS a;
23.	Built-in arithmetic functions	SIN(100); COS(20); TAN(50); LOG(100); LN(2000); ROOT(4);
24.	Other functions	PRIME(a); GCD(a,b); LCM(a,b); FIB(a);

CFG Used:

program:
MAIN '(' ')' SS arg SE
;

arg:
| arg statement
;

statement:
';'

|CMNT { printf("Comment Section\n");}

|BREAK { break;}

|declaration ';' { printf("Variable Declared\n"); }

|expression ';' { printf("The value of expression= %d\n", $1); $$=$1;}

|VARIABLE EQUALS expression ';' { table[$1] = $3; printf("Value of the variable= %d\t\n",$3); $$=$3;}

```
|FOR '(' expression EQUALS expression ';' expression ILT expression ')' SS statement SE
                        {if($5>$9)
                        {
                           printf("Invalid condition: Infinite loop\n");
                        }
                        else
                        {
                           int i; for(i=$5 ; i<=$9 ; i++) {printf("value of the FOR: %d
expression value: %d\n", i,$12);}
                        }
                      }

|FOR '(' expression EQUALS expression ';' expression IGT expression ')' SS statement SE {
                       if($9>$5)
                       {
                          printf("Invalid condition: Infinite loop\n");
                       }
                       else
                       {
                          int i; for(i=$5 ; i>=$9 ; i--) {printf("value of the FOR: %d expression
value: %d\n", i,$12);}
                       }
                     }

|WHILE '(' expression IGT expression ')' SS statement SE {
                      if($5>$3)
                      {
                       printf("Invalid condition: Infinite loop\n");
                      }
                      else
                      {
                       int i; for(i=$3 ; i>=$5 ; i--) {printf("value of the while : %d\n", i);}
                       }
                    }

|WHILE '(' expression ILT expression ')' SS statement SE {
                      if($3>$5)
                      { printf("Invalid condition: Infinite loop\n");
                      }
                      else
```

```
                    {
                    int i; for(i=$3 ; i<=$5 ; i++) {printf("value of the while : %d\n", i);} }
                    }

|VARIABLE INC ';' { table[$1] = table[$1] + 1;}

|VARIABLE DEC ';' { table[$1] = table[$1] - 1;}

|SWITCH '(' expression ')' BLOCK    {switchV=$3;}

|IF '(' expression ')' SS expression ';' SE %prec PRECIF {
                    if($3){
                        printf("\nThe value of expression in hoy: %d\n",$6);
                    }
                    else{
                        printf("\nCondition is false\n");
                    }
                }
|IF '(' expression ')' SS expression ';' SE ELSE SS expression ';' SE {
                    if($3){
                        printf("\nThe value of expression in hoy: %d\n",$6);
                    }
                    else{
                        printf("value of expression in nahole: %d\n",$11);
                    }
                }
|OUT '(' expression ')' ';' {printf("Value Output %d\n",$3);}

|PRIME '(' expression ')' ';' { int c=1; for(int i=2;i*i<=$3;i++){ if($3 % i==0){ printf("\n%d is
not prime\n",$3); c=0; break;}}
                        if(c){ printf("\n%d is prime\n",$3);}}

|CIN '(' VARIABLE ',' IDATA ')' ';'   { table[$3]=$5;
                    printf("input done. Value =%d \n",table[$3]);
                    }
|SQRT '(' expression ')' ';' {   printf("Value of root=%.4lf\n",sqrt($3*1.0)); }

|GCD '(' expression ',' expression ')' ';'  { int n1=$3,n2=$5,g;
                            for(int i=1;i<=n1&&i<=n2;i++)
                            {
```

```
                    if(n1%i==0&&n2%i==0)
                    {
                      g=i;

                    }
                    }
                  printf("GCD  of %d and %d = %d\n",$3,$5,g);
                }

  |LCM '(' expression ',' expression ')' ';' { int n1=$3,n2=$5,g;
                    for(int  i=1;i<=n1&&i<=n2;i++)
                    {
                     if(n1%i==0&&n2%i==0)
                     {
                       g=i;

                     }
                     }
                     int  x=n1/g*n2;
                  printf("LCM  of %d and %d = %d\n",$3,$5,x);
                }
  |FIB '(' expression ')' ';' { int n=$3; int fibo[n+6];  fibo[0]=0;  fibo[1]=1;
                 printf("The  fibonacci  series is= %d %d ",fibo[0],fibo[1]);
                  for(int  i=2;i<n;i++)
                  {
                    fibo[i]=fibo[i-1]+fibo[i-2];
                    printf("%d  ",fibo[i]);
                  }
                  printf("\n");

                  }
  |SIN '(' expression ')'     { printf("\nThe  value  of sin(%d)  is: %.2lf\n  ", $3,
sin($3*3.1416/180));  $$ = sin($3*3.1416/180);  }

  |COS '(' expression ')'     { printf("\nThe  value  of cos(%d) is: %.2lf\n",  $3,
cos($3*3.1416/180)); $$ = cos($3*3.1416/180);  }

  |TAN '(' expression ')'     { printf("\nThe  value  of tan(%d) is: %.2lf\n",  $3,
tan($3*3.1416/180));  $$ = tan($3*3.1416/180);  }
```

```
   |LOG '(' expression ')'      { if($3)
                          {printf("\nValue of log10(%d) is %lf\n",$3,(log($3*1.0)/log(10.0)));
$$=(log($3*1.0)/log(10.0));}
                       else{printf("\nillegal value as argument\n");} }

   |LN '(' expression ')'       { if($3)
                          {printf("\nValue of natural log, ln(%d) is %lf\n",$3,(log($3)));
$$=(log($3));}
                       else{printf("\nillegal value as argument\n");} }
   ;

BLOCK:
   SS SEGMENT SE

SEGMENT:
   CS
   |CS DF
   ;
CS:
   CS ';' CS

   |CASE IDATA ':' expression ';' BREAK ';' { if($2==switchV){printf("Case %d executed and
the expression value is=%d\n",$2,$4);switchF=0;}}
   ;
DF
   :DEFAULT ':' expression ';' BREAK ';' {if(switchF){printf("Default Case executed and the
expression value=%d\n",$3);}}
   ;

declaration:
   type id
   ;

type:
   INT
   |FLOAT
   ;


id: id ',' VARIABLE
```

```
   |VARIABLE
   ;

expression:
   IDATA              { $$ = $1;  }

   | VARIABLE                    { $$ = table[$1];  }
   | expression  ADD expression  { $$ = $1 + $3; }
   | expression  SUB expression  { $$ = $1 - $3; }
   | expression  MUL expression  { $$ = $1 * $3; }
   | expression  DIV expression  { if($3){
                                    $$ = $1 / $3;
                                   }
                                   else{
                                    $$ = 0;
                                    printf("\ndividing  by zero\t");
                                   }
                                 }
   | expression  MOD expression  { if($3){
                                    $$ = $1 % $3;
                                   }
                                   else{
                                    $$ = 0;
                                    printf("\nMOD  by zero\t");
                                   }
                                 }
   | expression  POW expression  { $$ = pow($1 , $3);}
   | expression  ILT expression  { $$ = $1 < $3; }
   | expression  IGT expression  { $$ = $1 > $3; }
   | '(' expression  ')'         { $$ = $2;  }
   ;
```

Sample Input and Output:

Input	Output
#insert math.h	Header file section
#insert string.h	Header file section
master()	Comment Section
<	Variable Declared

!!comment section	Value of the variable= 1
NUMBER a,b,c,d;	Value of the variable= 10
a EQUALS 1;	Value of the variable= 11
b EQUALS 10;	Value of the variable= 12
c EQUALS 11;	
d EQUALS 12;	1 is prime
PRIME(a);	GCD of 10 and 11 = 1
GCD(b,c);	LCM of 1 and 11 = 11
LCM(a,c);	Value Output 12
OUT(d);	Value of root=2.0000
ROOT(4);	The fibonacci series is= 0 1 1 2 3 5 8 13 21 34
FIB(15);	55 89 144 233 377
	Comment Section
!!if condition	
HOY(c IGT b)	The value of expression in hoy: 1
<	Comment Section
c SUB b;	
>	The value of expression in hoy: 9
	Comment Section
!!if-else condition	The value of expression= -1
HOY(b IGT a)	value of the FOR: 3 expression value: -1
<	value of the FOR: 4 expression value: -1
b SUB a;	value of the FOR: 5 expression value: -1
>	value of the FOR: 6 expression value: -1
NAHOLE	value of the FOR: 7 expression value: -1
<	value of the FOR: 8 expression value: -1
a SUB b;	value of the FOR: 9 expression value: -1
>	value of the FOR: 10 expression value: -1
	value of the FOR: 11 expression value: -1
!!for loop	Comment Section
JODI(a EQUALS 3; a ILT c)	

```
<
7 SUB 8;
>

!!switch case
SHIFT ( 2 )
<
VALUE 1 :
        a ADD b; END;
;
VALUE 2 :
        b ADD c; END;
;
VALUE 3 :
        a ADD c; END;
DEFAULT :
        c ADD 4; END;
>

!!while loop
JOKHON (6 ILT 8)
<
        c EQUALS c ADD 1;
>


d INC;
OUT(d);
d DEC;
OUT(d);
OUT(4 MUL 9);
OUT(4 ^ 2);
```

```
Default Case executed and the expression
value=15
Comment Section
Value of the variable= 12
value of the while : 6
value of the while : 7
value of the while : 8
Value Output 13
Value Output 12
Value Output 36
Value Output 16

The value of sin(100) is: 0.98

The value of cos(20) is: 0.94

The value of tan(50) is: 1.19

Value of log10(100) is 2.000000

Value of natural log, ln(2000) is 7.600902
```

SIN(100); COS(20); TAN(50); LOG(100); LN(2000); >	

Discussion:

The input code is parsed using a bottom-up parser in this compiler. Because it is only built with flex and bison, this compiler is unable to provide original functionality such as if-else, loop, and switch case features. However, when creating code in this compiler-specific style, header declaration is not required but if we need we can use header file. The float variable always returns a value in the double data type, which is a compiler requirement. Any variable's string value is not stored by this compiler. With certain modifications, the code format supported by this compiler is similar to that of the C language. This compiler is error-free while working with the stated CFG format.

Conclusion:

Every programming language has required the use of a compiler. Designing a new language without a solid understanding of how a compiler works may be a challenging endeavor. Several issues were encountered during the design phase of this compiler, such as loop, if-else, switch case functions not working as they should owing to bison limitations, character and string variable values not being stored properly, and so on. In the end, some of these issues were resolved, and given the constraints, this compiler performs admirably.

References:

- https://www.geeksforgeeks.org/flex-fast-lexical-analyzer-generator/
- https://www.geeksforgeeks.org/bison-command-in-linux-with-examples/