KHULNA UNIVERSITY OF ENGINNERING AND TECHNOLOGY
Department of Computer Science & Engineering

# COMPILER PROJECT

Course Title : Compiler Design Laboratory
Course No : CSE 3112

Topic : Simple Compiler using FLEX & Bison

SUBMITTED TO-

Name : Anisa Walida
Roll : 1907087
Section : B
Year : 3rd
Semester : 2nd

SUBMITTED TO-

**Nazia Jahan Khan Chowdhury**
Assistant Professor
CSE,KUET

**Dipannita Biswas**
Lecturer
CSE,KUET

# Objectives :

- To know about parsing
- To know about tokenizing
- To create a new compiler

# Introduction :

A compiler is a computer program that translates computer code written in one programming language into another language. The name compiler is primarily used for programs that translate source code from a high-level programming language to a lower level language to create an executable program.

**Flex and Bison**

Lex is a program that generates lexical analyzer. It is used with YACC parser generator. The lexical analyzer is a program that transforms an input stream into a sequence of tokens .It reads the input stream and produces the source code as output through implementing the lexical analyzer in the C program

Bison is a general-purpose parser generator that converts a grammar description (Bison Grammar Files) for an LALR(1) context-free grammar into a C program to parse that grammar. The Bison parser is a bottom-up parser. ... Compile the code output by Bison, as well as any other source files .

**Run the program in terminal**

1. bison -d bison.y

2. flex flex.l

3. gcc -std=c99 -o app bison.tab.c lex.yy.c

4. .\app

# Theory :

## flex.l file-

This file basically contains-

1. Regular Expressions –

   Lexical rules are often defined using regular expressions. These rules describe the patterns of characters that correspond to different types of tokens in the programming language.

   This are the regular expressions for my language-

   char [a-zA-Z]

   digit [0-9]

   special [$_@]

   space " "

   newline "\n"

   Datatype "Integer"|"Double"|"Type_Char"|"Type_Void"

   Operator
   "Op_Equ"|"Op_Plus"|"Op_Minus"|"Op_Divide"|"Op_Multiply"|"Op_And"|"Op_Or"|"Op_Mod"

   Relational_Operator
   "Ro_GT"|"Ro_LT"|"Ro_GE"|"Ro_LE"|"Ro_And"|"Ro_Or"|"Ro_Equ"|"Ro_Equ"|"Ro_NEqu"

2. Token Definitions:

   Associating regular expressions with token names.

   The tokens in my language –

```
";" {return DOT;}

"," {return CM;}

{digit}+        {
                                yylval.val = atoi(yytext);
                                return NUM;
                        }


"main" { return MAIN;}

"out_Var" { return PRINTVAR;}

"out_str" {return PRINTSTR;}

"out_Line" {return PRINTLN;}


"function_"({char}|{digit}|{special})+ {return FUNCTION;}


"Var_"({char}|{digit}|{special})+ { strcpy(yylval.text,yytext);return ID;}


"int" {  return INT;}

"point" { return DOUBLE;}

"char" { return CHAR;}


"in__" {return SCAN;}
```

"(" { return PB;}

")" { return PE;}

"{" { return BB;}

"}" { return BE;}

"=" {return ASGN;}

"+" {return PLUS;}

"-" {return MINUS;}

"*" {return MULT;}

"/" {return DIV;}

"<" {return LT;}

">" {return GT;}

"<=" {return LE;}

">=" {return GE;}

"If" { return IF;}

"ElseIf" {return ELSEIF;}

"Else" {return ELSE;}

"factorial" {return FACT;}

"for" {return FOR;}

"++" {return INC;}

"--" {return DEC;}

"To" {return TO;}

"While" {return WHILE;}

"Less"  {return LESS;}

"Great" {return GREAT;}

"Switch" {return SWITCH;}

"default" {return DEFAULT;}

":" {return COL;}

### bison.y file –

## Token

A token is the smallest element (character) of a computer language program that is meaningful to the compiler. The parser has to recognize these as tokens: identifiers, keywords, literals, operators, punctuators, and other separators.

The tokens of my compiler are –

SCAN LESS GREAT WHILE INT DOUBLE FACT CHAR MAIN PB PE BB BE DOT CM ASGN PRINTVAR PRINTSTR PRINTLN PLUS MINUS MULT DIV LT GT LE GE IF ELSE ELSEIF FOR INC DEC TO SWITCH DEFAULT COL FUNCTION

# CFG

Context-free grammars (CFGs) are used to describe context-free languages. A context-free grammar is a set of recursive rules used to generate patterns of strings. A context-free grammar can describe all regular languages and more, but they cannot describe all possible languages.

```
starthere       : function program function
                ;
program         : INT MAIN PB PE BB statement BE
                ;
statement       : /* empty */
                | statement declaration
                | statement print
                | statement expression
                | statement ifelse
                | statement assign
                | statement forloop
                | statement switch
                | statement whileloop
                | statement fact
                ;
declaration     : type variables DOT
                ;
type            : INT | DOUBLE | CHAR
                ;
variables       : variable CM variables
                | variable
                ;
variable        : ID
```

```
                    | ID ASGN expression
                    ;
assign :            ID ASGN expression DOT

                    | SCAN PB ID PE DOT
                    ;
print               : PRINTVAR PB ID PE DOT
                    | PRINTSTR PB STR PE DOT
                    | PRINTLN PB PE          DOT
                    ;
expression          : NUM {$$ = $1;}
                    | ID
                    | expression PLUS expression
                    | expression MINUS expression
                    | expression MULT expression
                    | expression DIV expression
                    | expression LT expression
                    | expression GT expression
                    | expression LE expression
                    | expression GE expression
                    | PB expression PE
                    ;

ifelse              : IF PB ifexp PE BB LoopStatement BE elseif          ;
ifexp               : expression
                    ;
elseif              : /* empty */
                    | elseif ELSEIF PB expression PE BB LoopStatement BE
                    | elseif ELSE BB LoopStatement BE
```

```
                    ;

whileloop :         WHILE PB ID LT NUM PE BB LoopStatement BE
                    | WHILE PB ID GT NUM PE BB LoopStatement BE
                    | WHILE PB ID LE NUM PE BB LoopStatement BE
                    | WHILE PB ID GE NUM PE BB LoopStatement BE ;

forloop             : FOR PB expression TO expression INC expression PE BB LoopStatement BE
                    | FOR PB expression TO expression DEC expression PE BB LoopStatement BE
                    ;

LoopStatement       :
                    | LoopStatement Lprint
                    ;

Lprint              : PRINTVAR PB ID PE DOT
                    | PRINTSTR PB STR PE DOT
                    | PRINTLN PB PE      DOT
                    ;
switch              : SWITCH PB expswitch PE BB switchinside BE
                    ;

expswitch           :  expression
                    ;


switchinside        : /* empty */
                    | switchinside expression COL BB statement BE
                    | switchinside DEFAULT COL BB statement BE
```

```
                    ;
function            : /* empty */
                    | function func
                    ;

func                : type FUNCTION PB fparameter PE BB statement BE  ;
fparameter          : /* empty */
                    | type ID fsparameter
                    ;
fsparameter         : /* empty */
                    | fsparameter CM type ID
                    ;
fact                : FACT PB expression PE DOT;
```

# Input Examples-

```
^.^ FUNCTION ^.^
int function_Max ( int Var_a, int Var_b )
{

}
int main ( ){
        out_str ( "Hello" ) ;

        ^.^ VARIABLE DECLARATION AND INITIALIZATION ^.^
        out_Line ( ) ;
    int Var_a = 2 + 87 ;
        out_Var ( Var_a ) ;
        out_Line ( ) ;
```

```
int Var_b = Var_a ;
out_Var ( Var_b ) ;
out_Line ( ) ;
int Var_c = 25,Var_d = Var_c;
out_Line ( ) ;

factorial(3);
^.^ INPUT OUTPUT ^.^

out_str ( "Enter a value (Integer): " );
int Var_x;
in__ ( Var_x );
out_Var ( Var_x );
out_Line ( ) ;

out_str ( "Enter another value (Integer): " );
int Var_y;
in__ ( Var_y );
out_Var ( Var_y );
out_Line ( ) ;

int Var_sum = Var_x + Var_y;
out_str ( "Sum : " );
out_Var ( Var_sum );
out_Line ( ) ;

int Var_sub = Var_x - Var_y;
out_str ( "Subtraction : " );
out_Var ( Var_sub );
```

```
out_Line ( ) ;

int Var_mul = Var_x * Var_y;
out_str ( "Multiplication : " );
out_Var ( Var_mul );
out_Line ( ) ;

int Var_div = Var_x / Var_y;
out_str ( "Division : " );
out_Var ( Var_div );
out_Line ( ) ;

^.^ IF ELSE ^.^

If ( 0 < 4 )
{
        out_str ( "IF " );

}
ElseIf ( 1 < 4 )
{
        out_str ( "Else IF" );

}
ElseIf ( 2 < 4 )
{
        out_str ( "Another Else IF" );
}
Else
```

```
{
        out_str ( "Else" );

}

^.^ FOR LOOP ^.^
out_Line ( ) ;
out_str ( "Loop start" );
for ( 4 To 9 ++ 1 )
{

        out_Line ( );
        out_Var ( Var_a );
        out_Line ( );

}
out_str ( "Loop end" );

out_str ( "Loop start" );
for ( 9 To 4 -- 1 )
{

        out_Line ( );
        out_Var ( Var_a );
        out_Line ( );

}
out_str ( "Loop end" );
```

```
    ^.^ SWITCH ^.^

out_Line ( ) ;
    Switch ( 7 )
    {
            1:
                    {
                            out_Line ( ) ;
                            out_str ( "CASE 1" );
                    }
            7:
                    {
                            out_Line ( ) ;
                            out_str ( "CASE 7" );
                    }


            default:  {


                    }
    }
    out_Line ( ) ;


    ^.^ WHILE LOOP ^.^
    int Var_i = 14;
    out_str ( "GT" );
    out_Line ( ) ;
```

```
While ( Var_i > 10 )
{
        out_str ( "While .. " );
        out_Line ( );
}
int Var_j = 14;
out_str ( "GE" );
out_Line ( ) ;
While ( Var_j >= 10 )
{
        out_str ( "While .. " );
        out_Line ( );
}
int Var_k = 6;
out_str ( "LT" );
out_Line ( ) ;
While ( Var_k < 10 )
{
        out_str ( "While .. " );
        out_Line ( );
}
int Var_l = 6;
out_str ( "LE" );
out_Line ( ) ;
While ( Var_l <= 10 )
{
        out_str ( "While .. " );
        out_Line ( );
}}
```

## Output Example –

Function Declared

Hello

89

89

Factorial of 3 is 6

Enter a value (Integer): 8

8

Enter another value (Integer): 2

2

Sum : 10

Subtraction : 6

Multiplication : 16

Division : 4

If executed

IF

Loop start4

4

5

6

7

8

9

Loop executes 6 times

Loop endLoop start9

9

8

7

6

5

4

Loop executes 6 times

Loop end

CASE 1

CASE 7Executed 7

GT

14 While ..

13 While ..

12 While ..

11 While ..

GE

14 While ..

13 While ..

12 While ..

11 While ..

10 While ..

LT

6 While ..

7 While ..

8 While ..

9 While ..

LE

6 While ..

7 While ..

8 While ..

9 While ..

10 While ..

Compilation Successful

# Discussion –

This project provided us a clear idea of how to design a compiler . By following the proper steps such as language design, lexer and parser implementation, symbol table and variable management and error handling a complete and efficient compiler can be designed. This compiler provided conditional logic , loops , variable declaration ,function, switch etc. There are also lots of scops and potential to enhance this compiler and make it more functionable.