

**KHULNA UNIVERSITY OF ENGINEERING &TECHNOLOGY**

**Department of Computer Science and Engineering**

**Report on CSE3212 PROJECT**

Course Title: Compiler Design Laboratory

Topic: Simple Compiler using Flex and Bison

Date of Submission: November 21, 2023

**Submitted by:**

Tanvir Hossain Tamim

Roll: 1907060

3rd Year 2nd Semester

Department of Computer Science and Engineering

Khulna University of Engineering & Technology.

**Objectives:**

1.To create a Lexer using Flex that can effectively tokenize a file written in the specified custom programming language.

2. To gain knowledge about how lexical analysis works of our own code implementation.

3.To learn how can we design our own programming language and recognize different language construct accurately.

4.To know about the basics of bison and learn how can implement to design a compiler.

**Introduction:**

A lexical analyzer, also known as a lexer or scanner, is the first phase of a compiler or interpreter. Its primary function is to read the source code and break it down into smaller pieces called tokens. Tokens are the smallest units of a programming language, such as keywords, identifiers, operators, and literals. Lexical analysis is a crucial step in language processing, enabling the transformation of human-readable code into a format suitable for machine interpretation and execution.

Bison is a general-purpose parser generator that converts an annotated context-free grammar into a deterministic LR or generalized LR (GLR) parser employing LALR(1) parser tables. As an experimental feature, Bison can also generate IELR(1) or canonical LR(1) parser tables. Once you are proficient with Bison, you can use it to develop a wide range of language parsers, from those used in simple desk calculators to complex programming languages.

**Features of the compiler:**

• Header declaration

• Body declarations

• Variable declarations (integer, character, float)

• Variable value assignment (=)

• Arithmetic operations (+, -, \*, /, % )

• Bitwise operation (Bitwise AND, Bitwise OR)

• Relational Operations (==, !=, <, <=, >, >= )

• Logical Operations(&&, ||)

• For loop • Switch-case • If-else

• Built-in Power , GCD Function and LCM Function

• Built-in Min Function and Max Function

• Single Multiline comment • Print and Scan Function

• User Defined Function

**Syntax:**

|  |  |  |  |
| --- | --- | --- | --- |
| SL No. | Syntax | Definition of Token | Example |
| 1. | #import | Import the header files. | #import header.h  #import library.h |
| 2 | $$  <! …..  !> | Single line comment and multi line comment. | $$I am Tamim  <!this is the compiler lab!> |
| 3 | <InT>,<ChaR>,<DoublE> | Specify the variable of different data type. | <InT> x:=10  <ChaR>y:=$a$ etc. |
| 4 | <PrintF> ,<ScanF> | Print the value and resd the user input. | <PrintF>(($string$))  <ScanF>((int1)) |
| 5 | IncOnE and DecOnE | Increment and decrement the value of a variable. | IncOnE((x))  DecOnE((v)) |
| 6 | <IF> | If statement like c | <IF>(( i << 7)){{ }} |
| 7 | <ElsE> | Same to if statement | <ElsE>(( i << 7)){{ }} |
| 8 | <SwitcH>,<CasE> | Switch using like in c . | <SwitcH>(( i )) |
| 9 | <FoR> | Denotes the for loop, which can execute certain block for a certain time | <FoR>(( i:=0 ;; i << 4 ;; IncOnE((i)) )) {{ }} |
| 10 | <WhilE> | While loop like c | <WhilE>((i << 4)) {{ }} |
| 11 | Func\_Def | User can define a function in his own. | Func\_Def  (( i1:=4 ,, d2:= 6. 58)) {{ }} Func\_Def  (( i, d )) |
| 12 | +op+ -op- \*op\* /op/ %op% |op| &op& //op// | Arithmatic Operator | i1 = i2 +op+ i3 d1 = i2 -op- d3 d1 = i2 /op/ d3 i1 = i2 &op& i3 |
| 13 | == !!== << >> <<== >>== | Relational operators | if (( i << 10 )) {{ }} |
| 14 | (( )) {{ }} [[ ]] | Brackets | Func(())  {{  ..something..  }} |
| 15 | &&  ||  ! | Logical operator | X=a&&b  Y=q||r |
| 16 | GCD | Build-in function | GCD((c,d)) |
| 17 | LCM | Same | LCD((a,b)) |
| 18 | MIN | same | MIN((X,Y)) |
| 19 | MAX | same | MAX((a,b)) |

**Terminal commands to run the program:**

1. bison –d project.y

2. flex project.l

3. gcc project.tab.c lex.yy.c –o app

4. app

**.l file(project.l):**

%{

#include "project.tab.h"

void yyerror(char \*);

%}

DIGIT [0-9]

CHAR [a-zA-Z]

WHITE\_SPACE [ \t]

NEWLINE [\n]

REAL\_NUMBER [-+]?[0-9]+[.][0-9]+

INTEGER [-+]?[0-9]+

ID (\_|{CHAR})({CHAR}|{DIGIT})\*

DATA\_TYPE "<InT>"|"<ChaR>"|"<FloaT>"|"<DoublE>"|"<EmptY>"

ARITHMTIC\_OPERATOR "+op+"|"-op-"|"\*op\*"|"/op/"|"%op%"|("|op|")|"&op&"|"//op//"

RELATIONAL\_OPERATOR "=="|"<<"|">>"|"<<=="|">>=="|"!!=="

LOGICAL\_OPERATOR "&&"|("||")|"!"

%%

"#import ".+".h" {

yylval.union\_variable.name = strdup(yytext);

return LIBRARY;

}

"<!" {

printf("\nMultiple line comment begins"); char c;

while(c=input()!='!'); if(c=input()=='>')

printf("\n multiple line comment ends");

}

\$\$(.\*) { printf("\nSingle-line comment: %s\n", yytext);}

"<InT>" {return INT ;}

"<ChaR>" {return CHAR ;}

"<DoublE>" {return DOUBLE ;}

"<EmptY>" {return VOID ;}

":=" {return ASSIGN ;}

"==" {return EQUAL ;}

"!!==" {return NOT\_EQUAL ;}

">>" {return GREATER\_THAN ;}

"<<" {return LESS\_THAN ;}

">>==" {return GREATER\_THAN\_AND\_EQUAL ;}

"<<==" {return LESS\_THAN\_AND\_EQUAL ;}

"&&" {return AND ;}

"||" {return OR ;}

"TruE" {return TRUE ;}

"FalsE" {return FALSE;}

"+op+" {return PLUS ;}

"-op-" {return MINUS ;}

"\*op\*" {return MULTIPLE ;}

"/op/" {return DIVISION ;}

"%op%" {return MOD ;}

"^op^" {return POW ;}

"&op&" {return BINARY\_AND ;}

"|op|" {return BINARY\_OR ;}

"IncOnE" {return INC\_ONE ;}

"DecOnE" {return DEC\_ONE ;}

"((" {return FIRST\_BRACKET\_OPEN ;}

"))" {return FIRST\_BRACKET\_CLOSE ;

"{{" {return SECOND\_BRACKET\_OPEN ;}

"}}" {return SECOND\_BRACKET\_CLOSE ;}

"[[" {return THIRD\_BRACKET\_OPEN ;}

"]]" {return THIRD\_BRACKET\_CLOSE ;}

";;" {return SEMICOLON ;}

"::" {return COLON ;}

",," {return COMMA ;}

"<ReturN>" {return RETURN ;}

"<PrintF>" {return PRINTF ;}

"<ScanF>" {return SCANF ;}

"<FoR>" {return FOR ;}

"<WhilE>" {return WHILE ;}

"<ContinuE>" {return CONTINUE ;}

"<BreaK>" {return BREAK ;}

"<AmountOF>" {return SIZE\_OF ;}

"<IF>" {return IF ;}

"<ElsE>" {return ELSE ;}

"GCD" {return GCD ;}

"LCM" {return LCM ;}

"MIN" {return MIN ;}

"MAX" {return MAX ;}

"<SwitcH>" {return SWITCH ;}

"<CasE>" {return CASE ;}

"<DefaulT>" {return DEFAULT ;}

{REAL\_NUMBER} {

yylval.union\_variable.double\_value = atof(yytext);

yylval.union\_variable.data\_type = "double";

return DOUBLE\_VALUE ;

}

{INTEGER} {

yylval.union\_variable.int\_value = atoi(yytext);

yylval.union\_variable.data\_type = "int";

return INTEGER\_VALUE ;

}

{ID} {

yylval.union\_variable.name = strdup(yytext);

return VARIABLE\_NAME ;

}

[$]{CHAR}[$] {

yylval.union\_variable.char\_value = yytext[1];

yylval.union\_variable.data\_type = "char";

return CHAR\_VALUE ;

}

"$"({WHITE\_SPACE}|.)\*"$" {

yytext++;

int l=strlen(yytext);

yylval.union\_variable.name=yytext;

yylval.union\_variable.name[l-1]='\0';

return OUTPUTTEXT;

}

{WHITE\_SPACE} {}

. {

yyerror("Unknown character");

exit(-1);

}

%%

int yywrap()

{

return 1;

}

**.y file(project.y):**

%{

#include<stdio.h>

#include<string.h>

#include<math.h>

// void yyerror(const char \*s);

extern int yylex();

extern int yyparse();

struct symbol\_table\_structure{

char \*name;

char \*data\_type;

int int\_value;

double double\_value;

char char\_value;

};

struct symbol\_table\_structure symbol\_table[1000];

int symbol\_table\_index = 0;

int switch\_value = 0;

int switch\_check = 0;

int find\_symbol\_table\_index(char \*var)

{

for (int i = 0; i < symbol\_table\_index; i++) {

if (strcmp(symbol\_table[i].name, var) == 0) return i;

}

return symbol\_table\_index;

}

void assignment(char \*name, char \*type, int int\_value, double double\_value, char char\_value)

{

int i = find\_symbol\_table\_index(name);

symbol\_table[i].name = name;

symbol\_table[i].data\_type = type;

symbol\_table[i].int\_value = int\_value;

symbol\_table[i].double\_value = double\_value;

symbol\_table[i].char\_value = char\_value;

if (i == symbol\_table\_index)

{

symbol\_table\_index++;

}

// printf("%s %d value is: %d\t",type, i, symbol\_table[i].int\_value);

}

int gcd(int x, int y)

{

if (y == 0)

{

return x;

}

else

{

return gcd(y, x % y);

}

}

int min(int a, int b)

{

return (a>b)?b:a;

}

int max(int a, int b)

{

return (a>b)?a:b;

}

%}

%union {

char text[1000];

struct datatype {

char\* name;

char\* data\_type;

int int\_value;

double double\_value;

char char\_value;

} union\_variable;

//YYSTYPE yylval;

}

%token INT CHAR VOID DOUBLE

%token DOUBLE\_VALUE INTEGER\_VALUE CHAR\_VALUE VARIABLE\_NAME

%token ASSIGN

%left PLUS MINUS BINARY\_OR BINARY\_AND

%left MULTIPLE DIVISION MOD

%left POW

%token EQUAL NOT\_EQUAL GREATER\_THAN LESS\_THAN GREATER\_THAN\_AND\_EQUAL LESS\_THAN\_AND\_EQUAL

%token AND OR

%token INC\_ONE DEC\_ONE

%token TRUE FALSE

%nonassoc FIRST\_BRACKET\_OPEN FIRST\_BRACKET\_CLOSE SECOND\_BRACKET\_OPEN SECOND\_BRACKET\_CLOSE THIRD\_BRACKET\_OPEN THIRD\_BRACKET\_CLOSE SEMICOLON COLON COMMA

%token IF ELSE\_IF ELSE SWITCH CASE DEFAULT FOR WHILE CONTINUE BREAK

%token PRINTF SCANF SIZE\_OF RETURN

%token OUTPUTTEXT LIBRARY

%nonassoc GCD LCM

%nonassoc MIN MAX

%start program

%type<union\_variable> VARIABLE\_NAME INTEGER\_VALUE DOUBLE\_VALUE CHAR\_VALUE assign\_value OUTPUTTEXT LIBRARY block statement

%%

program:

|program statement

;

statement:

declaration { /\* action for declaration \*/ }

| expressions { /\* action for expressions \*/ }

| print { /\* action for print \*/ }

| scan { /\* action for scan \*/ }

| while { /\* action for while \*/ }

| for { /\* action for for \*/ }

| if { /\* action for if \*/ }

| library { /\* action for library \*/ }

| function { /\* action for function \*/ }

| function\_call { /\* action for function\_call \*/ }

| switch { /\* action for switch \*/ }

| case { /\* action for case \*/ }

| gcd { /\* action for gcd \*/ }

| lcm { /\* action for lcm \*/ }

| min { /\* action for min \*/ }

| max { /\* action for max \*/ }

;

library: LIBRARY { printf("%s is header file that included.\n", $1.name);}

function:

TYPE VARIABLE\_NAME FIRST\_BRACKET\_OPEN params FIRST\_BRACKET\_CLOSE SECOND\_BRACKET\_OPEN statement block {

printf("User defined function.\n");

}

;

params:

params COMMA declaration\_for\_function

|declaration\_for\_function {printf("Function declared\n");}

;

declaration\_for\_function:

|TYPE expression

;

function\_call:

VARIABLE\_NAME FIRST\_BRACKET\_OPEN expressions FIRST\_BRACKET\_CLOSE

|VARIABLE\_NAME FIRST\_BRACKET\_OPEN FIRST\_BRACKET\_CLOSE

;

declaration:

TYPE expressions

;

TYPE :

INT

| DOUBLE

| CHAR

| VOID

;

expressions :

expressions COMMA expression

| expression

;

expression:

VARIABLE\_NAME ASSIGN assign\_value {assignment($1.name, $3.data\_type, $3.int\_value, $3.double\_value, $3.char\_value);}

| INC\_ONE FIRST\_BRACKET\_OPEN VARIABLE\_NAME FIRST\_BRACKET\_CLOSE {

int i = find\_symbol\_table\_index($3.name);

if (i != symbol\_table\_index) {

symbol\_table[i].int\_value = symbol\_table[i].int\_value + 1;

}

}

| DEC\_ONE FIRST\_BRACKET\_OPEN VARIABLE\_NAME FIRST\_BRACKET\_CLOSE {

int i = find\_symbol\_table\_index($3.name);

if (i != symbol\_table\_index) {

symbol\_table[i].int\_value = symbol\_table[i].int\_value - 1;

}

}

;

assign\_value: INTEGER\_VALUE {$$.data\_type=$1.data\_type; $$.int\_value=$1.int\_value; $$.double\_value=$1.double\_value; $$.char\_value=$1.char\_value;}

| DOUBLE\_VALUE {$$.data\_type=$1.data\_type; $$.int\_value=$1.int\_value; $$.double\_value=$1.double\_value; $$.char\_value=$1.char\_value;}

| CHAR\_VALUE {$$.data\_type=$1.data\_type; $$.int\_value=$1.int\_value; $$.double\_value=$1.double\_value; $$.char\_value=$1.char\_value;}

| VARIABLE\_NAME {

for (int i = 0; i < symbol\_table\_index; i++)

{

if(strcmp($1.name, symbol\_table[i].name)==0)

{

$$.int\_value = symbol\_table[i].int\_value, $$.double\_value = symbol\_table[i].double\_value, $$.data\_type = symbol\_table[i].data\_type;

break;

}

}

}

| assign\_value AND assign\_value {

if (strcmp($1.data\_type, "int")==0)

{

$$.int\_value = ($1.int\_value && $3.int\_value);

$$.data\_type = $1.data\_type;

}

}

| assign\_value OR assign\_value {

if (strcmp($1.data\_type, "int")==0)

{

$$.int\_value = ($1.int\_value || $3.int\_value);

$$.data\_type = $1.data\_type;

}

}

| assign\_value LESS\_THAN assign\_value {

if (strcmp($1.data\_type, "int")==0)

{

$$.int\_value = ($1.int\_value < $3.int\_value);

$$.data\_type = $1.data\_type;

}

}

| assign\_value GREATER\_THAN assign\_value {

if (strcmp($1.data\_type, "int")==0)

{

$$.int\_value = ($1.int\_value > $3.int\_value);

$$.data\_type = $1.data\_type;

}

}

| assign\_value LESS\_THAN\_AND\_EQUAL assign\_value {

if (strcmp($1.data\_type, "int")==0)

{

$$.int\_value = ($1.int\_value <= $3.int\_value);

$$.data\_type = $1.data\_type;

}

}

| assign\_value GREATER\_THAN\_AND\_EQUAL assign\_value {

if (strcmp($1.data\_type, "int")==0)

{

$$.int\_value = ($1.int\_value >= $3.int\_value);

$$.data\_type = $1.data\_type;

}

}

| assign\_value EQUAL assign\_value {

if (strcmp($1.data\_type, "int")==0)

{

$$.int\_value = ($1.int\_value == $3.int\_value);

$$.data\_type = $1.data\_type;

}

}

| assign\_value NOT\_EQUAL assign\_value {

if (strcmp($1.data\_type, "int")==0)

{

$$.int\_value = ($1.int\_value != $3.int\_value);

$$.data\_type = $1.data\_type;

}

}

| assign\_value PLUS assign\_value { $$.data\_type = $1.data\_type, $$.int\_value = $1.int\_value + $3.int\_value, $$.double\_value = $1.double\_value + $3.double\_value, $$.char\_value = $1.char\_value + $1.char\_value;}

| assign\_value MINUS assign\_value { $$.data\_type = $1.data\_type, $$.int\_value = $1.int\_value - $3.int\_value, $$.double\_value = $1.double\_value - $3.double\_value, $$.char\_value = $1.char\_value - $1.char\_value;}

| assign\_value BINARY\_AND assign\_value { $$.data\_type = $1.data\_type, $$.int\_value = $1.int\_value & $3.int\_value;}

| assign\_value BINARY\_OR assign\_value { $$.data\_type = $1.data\_type, $$.int\_value = $1.int\_value | $3.int\_value;}

| assign\_value MULTIPLE assign\_value { $$.data\_type = $1.data\_type, $$.int\_value = $1.int\_value \* $3.int\_value, $$.double\_value = $1.double\_value \* $3.double\_value, $$.char\_value = $1.char\_value \* $1.char\_value;}

| assign\_value DIVISION assign\_value { $$.data\_type = $1.data\_type, $$.int\_value = $1.int\_value / $3.int\_value, $$.double\_value = $1.double\_value / $3.double\_value, $$.char\_value = $1.char\_value / $1.char\_value;}

| assign\_value MOD assign\_value { $$.data\_type = $1.data\_type, $$.int\_value = $1.int\_value % $3.int\_value;}

| assign\_value POW assign\_value { $$.data\_type = $1.data\_type, $$.int\_value = pow($1.int\_value, $3.int\_value), $$.double\_value = pow($1.double\_value, $3.double\_value), $$.char\_value = pow($1.char\_value, $1.char\_value);}

| FIRST\_BRACKET\_OPEN assign\_value FIRST\_BRACKET\_CLOSE {$$=$2}

;

print:

PRINTF FIRST\_BRACKET\_OPEN assign\_value FIRST\_BRACKET\_CLOSE {

if (strcmp($3.data\_type, "int") == 0)

printf("%d\n", $3.int\_value);

else if (strcmp($3.data\_type, "double") == 0)

printf("%lf\n", $3.double\_value);

else if(strcmp($3.data\_type, "char") == 0)

printf("%c\n", $3.char\_value);

}

| PRINTF FIRST\_BRACKET\_OPEN OUTPUTTEXT FIRST\_BRACKET\_CLOSE { printf("%s\n", $3.name);}

;

scan:

SCANF FIRST\_BRACKET\_OPEN VARIABLE\_NAME FIRST\_BRACKET\_CLOSE {

int i = find\_symbol\_table\_index($3.name);

if (i != symbol\_table\_index) {

printf("Enter value for %s := ", symbol\_table[i].name);

if (strcmp(symbol\_table[i].data\_type, "int") == 0) {

scanf("%d", &symbol\_table[i].int\_value);

}

else if (strcmp(symbol\_table[i].data\_type, "double") == 0) {

scanf("%lf", &symbol\_table[i].double\_value);

}

else if (strcmp(symbol\_table[i].data\_type, "char") == 0){

scanf("%c", &symbol\_table[i].char\_value);

}

}

else {

printf("Variable not declared\n");

}

while: WHILE FIRST\_BRACKET\_OPEN assign\_value FIRST\_BRACKET\_CLOSE SECOND\_BRACKET\_OPEN block {

if($3.int\_value)

{

printf("while loop is executed");

}

}

for: FOR FIRST\_BRACKET\_OPEN expression SEMICOLON assign\_value SEMICOLON expression FIRST\_BRACKET\_CLOSE SECOND\_BRACKET\_OPEN block {

if($5.int\_value)

{

printf("for loop executed");

}

}

if: IF FIRST\_BRACKET\_OPEN assign\_value FIRST\_BRACKET\_CLOSE SECOND\_BRACKET\_OPEN block {

if($3.int\_value)

{

printf("If statement will be executed.\n");

}

}

| IF FIRST\_BRACKET\_OPEN assign\_value FIRST\_BRACKET\_CLOSE SECOND\_BRACKET\_OPEN block ELSE SECOND\_BRACKET\_OPEN block {

if($3.int\_value)

{

printf("If statement will be executed.\n");

}

else

{

printf("Else statement will be executed.\n");

}

}

block:

statement block

| statement SECOND\_BRACKET\_CLOSE {$$.int\_value=$1.int\_value;}

;

switch:

SWITCH FIRST\_BRACKET\_OPEN assign\_value FIRST\_BRACKET\_CLOSE {

switch\_value = $3.int\_value;

switch\_check = 0;

printf("Switch statement check\n");

}

;

case:

cases

{

if(switch\_check == 0)

{

printf("default\n");

}

}

;

cases:

CASE assign\_value COLON SECOND\_BRACKET\_OPEN statement SECOND\_BRACKET\_CLOSE cases {

if(switch\_value==$2.int\_value)

{

printf("%d\n",$2.int\_value);

switch\_check = 1;

}

}

| CASE assign\_value COLON SECOND\_BRACKET\_OPEN statement SECOND\_BRACKET\_CLOSE default\_function {

if(switch\_value==$2.int\_value)

{

printf("%d\n",$2.int\_value);

switch\_check = 1;

}

}

;

default\_function:

DEFAULT COLON SECOND\_BRACKET\_OPEN statement SECOND\_BRACKET\_CLOSE {}

;

gcd: GCD FIRST\_BRACKET\_OPEN assign\_value COMMA assign\_value FIRST\_BRACKET\_CLOSE {

printf("%d %d GCD is %d\n",$3.int\_value, $5.int\_value, gcd($3.int\_value, $5.int\_value));

}

lcm: LCM FIRST\_BRACKET\_OPEN assign\_value COMMA assign\_value FIRST\_BRACKET\_CLOSE {

printf("%d %d LCM is %d\n",$3.int\_value, $5.int\_value, $3.int\_value \* $5.int\_value / gcd($3.int\_value, $5.int\_value));

}

min: MIN FIRST\_BRACKET\_OPEN assign\_value COMMA assign\_value FIRST\_BRACKET\_CLOSE {

printf("%d %d Min is %d\n",$3.int\_value, $5.int\_value, min($3.int\_value, $5.int\_value));

}

max: MAX FIRST\_BRACKET\_OPEN assign\_value COMMA assign\_value FIRST\_BRACKET\_CLOSE {

printf("%d %d Max is %d\n",$3.int\_value, $5.int\_value, max($3.int\_value, $5.int\_value));

}

%%

void yyerror(const char \*s)

{

fprintf(stderr, "Error: %s\n", s);

}

int main(void)

{

yyparse();

return 0;

}

**Discussion:**

This compiler uses a bottom up parser to parse the input code. This compiler is unable to provide original functionality of if else, loop, switch case features as it is only build using flex and bison. However header declaration is not mandatory while writing a code in this compiler specific format. This compiler doesn’t stores string value of any variable. The code format that is supported by this compiler is close to that of C language with some modification. This compiler works perfectly with the declared CFG format without any error.

**Conclusion:**

Compiler has been an essential part of every programming language. Without a sound knowledge about how a compiler works, designing a new language can be very difficult task. Several difficulties are faced during design period of this compiler such as loop, if else, switch case functions not working as it should be due to limitation of bison, character and string variable value isn’t storing properly, etc. At the end, some of this problems have been fixed and considering the limitations this compiler works just fine.

**References:**

**1.** LEX & YACC TUTORIAL by Tom Niemann

2. https://www.gnu.org/software/bison/#:~:text=Bison%20is%20a%20general%2Dpurpose,LR(1)%20parser%20tables.