University of Asia Pacific

Department of Computer Science & Engineering

Course Title : Compiler Desing Lab

Course Code : CSE-430

Lab Report Title : Simple Compiler Design Project using Flex and Bison

Submitted By,	**Submitted To,**
Apurba Dey	Nipa Anjum
Reg ID : 20101043	Lecturer
Section : A2	CSE , UAP

Date of Submission **:** 28 April ,2024

Introduction:

In my lab, I learned to build a basic compiler using Flex and Bison. Flex helps to create patterns for a lexical analyzer, which converts strings in the source code into numerical tokens. Bison then reads grammar rules to generate a syntax analyzer, constructing a hierarchical structure called a syntax tree. This process translates high-level code into machine-readable instructions, making it possible to execute programs on different data. Flex generates a ".c" file from patterns written in a ".l" file, while Bison, a free version of Yacc, produces C code for the syntax analyzer from a ".y" file, resulting in ".tab.h" and ".tab.c" files.

Description:

Structure of Flex file :

```
{Definitions}

  %%

{Rules}

%%

{User Subroutines}
```

Structure of Bison file :

```
%{
    C declarations
%}

Bison declarations

%%

Grammar Rules

%%

 Additional C codes
```

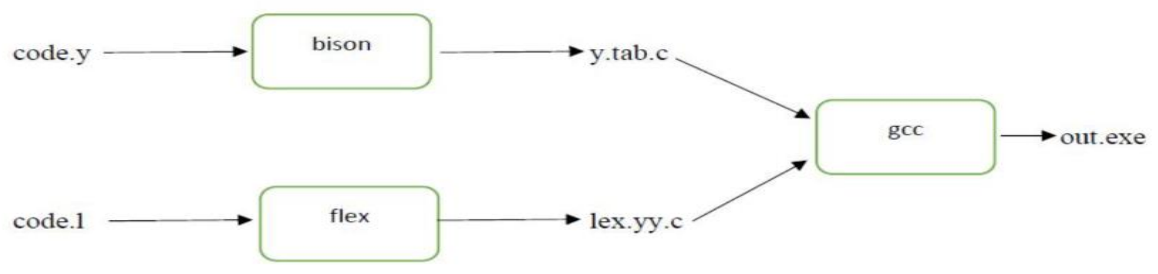

Fig: Design a Compiler with Flex and Bison.

Feature of the Compiler project:

1. Header declaration
2. Variable declaration
3. Variable values assignment
4. Arithmetic Operations(addition, subtraction,multiplication,division)
5. If-else condition
6. For loop
7. While loop
8. Single line comment
9. Multiple lines comment

Used CFG:

```
program: MAIN START begin END     {}
             ;

begin:                           {}
             | begin code
             ;

code:        ';'                 {}
             |     expression    ';'    {}

             |     declaration_part ';'      {}

               |   assignment_part {}
```

```
                |      PRINT '(' ID ')' ';'      { expression}


                | FROM INT TO INT INC INT START expression END{ expression }


                |IF expression START expression ';' END ELSE START expression ';' END
                          { expression }

             | WHILE ID INCREMENT '<' INT START expression END { expression}

                | WHILE ID DECREMENT '>' INT START expression END   { expression }


declaration_part:
                     T_INT INTID

                    | T_DOUBLE DOUBLEID

                    | T_STRING CHARID
                    ;

INTID  :
                      INTID ',' INT_ID
                    | INT_ID
INT_ID:
                     ID '=' expression{


                    | ID                { expression }

DOUBLEID:
                      DOUBLEID ',' DOUBLE_ID
                    | DOUBLE_ID

DOUBLE_ID:
                      ID '=' expression {   expression }

                    | ID
CHARID:
                      CHARID ',' CHAR_ID

                    | CHAR_ID
                    ;
CHAR_ID:
                      ID '=' STRING                  { expression }

expression:          INT   {$$ = $1;}

                    | DOUBLE {$$ = $1;}

                    | ID
```

```
                          | expression '+' expression {$$ = $1 + $3;}

                        | expression '-' expression {$$ = $1 - $3;}

                        | expression '*' expression {$$ = $1 * $3;}

                        | expression '/' expression { expression }

                        | expression '^' expression {$$ = powl($1,$3);}

                        | expression '%' expression { expression }

                        | expression '>' expression {$$ = ($1 > $3);}

                        | expression '<' expression {$$ = ($1 < $3);}

                        | expression EQUAL expression {$$ = ($1 ==$3);}

                        | expression NT_EQUAL expression {$$ = ($1!=$3);}

                        | expression LS_EQUAL expression {$$ = ($1<=$3);}

                        | expression GT_EQUAL expression {$$ = ($1<=$3);}



                        ;
%%
```

Sample Input :

```
apurba std.h
apurba math.h


main{

#@ Variable Declaration Part

INTEGER a,b,c,i=1;

#@ Assignment Part
     a=2;
     b=1;
     c=5;
#@ If ElSE Part
```

```
    IF b>c{
      c+1;
    }
    ELSE{
      c-1;
    }

#@ For loop Part
 from 2 to 6 inc 1{
    a+3
   }

#@ While loop Part
  while b ++ < 5{
      b+c
   }

#@ Arithmetic Operation Part
     c=b+1;
     b=c+a;
     c=a*b;
     a=c/b;
     b=c-b;

#@ Multiline Part

 @@@ I am Apurba Dey. My Final examination is
     knocking at the door.Try hard to do
     something good.@@@
```

Sample Output :

```
Header file found

Header file found

             Program Begins.

 A Single Line Comment is Found.

a is declared

b is declared
```

```
c is declared

i is declared with value :1

A Single Line Comment is Found.

2 is assigned to a

1 is assigned to b

5 is assigned to c


 A Single Line Comment is Found.


Value of expression in else block is 4.00


A Single Line Comment is Found.

Inside For loop.Value of expression: in 2th : 5.00

Inside For loop.Value of expression: in 3th : 5.00

Inside For loop.Value of expression: in 4th : 5.00

Inside For loop.Value of expression: in 5th : 5.00

Inside For loop.Value of expression: in 6th : 5.00


 A Single Line Comment is Found.

Inside while loop.
Value of expression: 6.00

Inside while loop.
Value of expression: 6.00

Inside while loop.
Value of expression: 6.00

Inside while loop.
Value of expression: 6.00


 A Single Line Comment is Found.

2 is assigned to c
```

```
4 is assigned to b

8 is assigned to c
2 is assigned to a

4 is assigned to b


 A Single Line Comment is Found.

A Multi-line Comment is Found.


   Program Ends
```

<u>Conclusion :</u> Creating a compiler is a big deal in computer science. If I want to dive deep into programming languages, I've got to understand how compilers work. So, I decided to try my hand at making a simple one that behaves like Python and C. It's like teaching the computer to understand and follow the rules of these languages. It's a challenging task, but it's essential for getting to the heart of how programming languages function.

References :

1. Writing a simple Compiler on my own - Combine Flex and Bison — Steemit
2. Introduction to Flex and Bison - sinkinben (cnblogs.com)
3. compiler.pdf (admb-project.org)