“Lab Report”

ON

‘Project of Simple Compiler Design Using Flex And Bison’

Course_Name: Compiler Design Laboratory

Course _No: 3212

Submitted To:

Dola Das,
Lecturer.
Md. Ahsan Habib Nayan,
Lecturer.
Department of Computer Science and Engineering,
Khulna University of Engineering and Technology,
Khulna.

Submitted By:

Name : Redwona Islam.
Roll : 1707113.
Student of -
Department of Computer Science and Engineering,
Khulna University of Engineering and Technology,
Khulna.

Submission Date: 08 / 06 / 2021.

Introduction:

Compiler Design is the structure and set of principles that guide the translation, analysis, and optimization process of a compiler. And a Compiler is computer software that transforms program source code which is written in a high-level language into low-level machine code.

Flex

Flex is a tool for generating scanners.Flex source is a table of regular expressions and corresponding program fragments. It is a program that generates lexical analyzers. It is used with the 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.

Flex specifications:

{ definitions }
%%
{ rules }
%%
{ user subroutines }

Bison

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. It is a powerful and free version of Yacc.

Bison input file format is given below :

```
%{
  C declarations { types , variables , functions , preprocessor
                   commands  }
 %}
 /* Bison Declarations (grammar symbols , operator precedence
                        declaration , attribute data type) */
%%
  /* grammar rules go here */
%%
/* additional c code goes here */
```

Equipment :

Laptop, Flex , Bison , Command Prompt , NotePad .

Procedure:

1.The code is divided into two parts flex file (.l) and bison file (.y) .

2.Input expression checks the lex (.y) file and if the expression satisfies the rule then it checks the CFG into the bison file .

3.it's a bottom up parser and the parser constructs the parse tree .firstly ,matches the leaves node with the rules and if the CFG matches then it gradually goes to the root .

Run The program in terminal:

1. bison -d mainf.y
2. flex mainf.l
3. gcc lex.yy.c mainf.tab.c -o app
4. out.

Description :

A lexical analyzer divides the input into meaningful units or 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 .

My compiler Tokens -

1. VOIDMAIN -> This token is returned after getting the "Main_start" pattern.

2. NUM -> This token is returned after getting "[-+]?[0-9]+" & [0-9]*"."[0-9]+ pattern. Example : 4 , -2 ,1.7 etc.

3. VAR-> This token is returned after getting the "{variable}" pattern. {Variable} is included with [a-zA-Z]. Example : a,Z,b,S etc.

4. PST -> This token is returned when the "SP" pattern is found. Ilt's like left parenthesis "{".

5. PEN ->This token is returned when the "EP" pattern is found. It's like right parenthesis "}".

6. BST -> This token is returned when the "{{" pattern is found. It's like 2nd Left Bracket "{".

7.BEN -> This token is returned when the "}}" pattern is found. It's like 2nd Right Bracket "}".

8. SM -> This token is returned when the "[;][;]" pattern is found. It's like Semicolon(;).

9. INT -> This token is returned when the "t_int" pattern is found.

10. FLOAT -> This token is returned when the "t_float" pattern is found.

11. CHAR -> This token is returned when the "t_char" pattern is found.

12. ARRAY -> This token is returned when the "array" pattern is found.

13. IF -> This token is returned when the “iff” pattern is found.

14. EISE -> This token is returned when the“els” pattern is found.

15. EISEIF -> This token is returned when the “elf” pattern is found.

16. FOR -> This token is returned when the “for_” pattern is found.

17. WHILE -> This token is returned when the “while_” pattern is found.

18.SWITCH -> This token is returned when the “switch_” pattern is found.

19.CASE ->This token is returned when the “CASE” pattern is found.

20. DEFAULT ->This token is returned when the “DEFAULT” pattern is found.

21. PRINTFUNCTION -> This token is returned after getting the “PRINT_fun” pattern. Example - PRINT_fun SP 45 -- 55 EP gives like- Print Expression -10.

22.PLUS -> This token is returned when the "++"pattern is found. It means ‘Addition’ of numbers. Example: 1 ++ 4 = 5.

23.MINUS -> This token is returned when the “--” pattern is found. It means ‘Subtraction’ of numbers. Example: 10 -- 4 = 6.

24. MUL -> This token is returned when “” pattern is found. It means ‘Multiplication’ of numbers. Example: 3 ** 4=12.**

25. DIV -> This token is returned when “//”

pattern is found. It means ‘Division’ of numbers. Example: 20 // 4 = 5.

26. MOD -> This token is returned when the “% %” pattern is found. It means ‘MOD’ of numbers. Example: 21 %% 4 = 1.

27. LT -> This token is returned when the “<<” pattern is found. It refers “Less Than” between two numbers. Example- 5 << 10.

28. GT -> This token is returned when “>>” pattern is found.It refers “Greater Than” between two numbers. Example- 55>> 10.

29. LE -> This token is returned when the “<<=” pattern is found. It refers “Less Than Equal to” between two numbers.
Example- 5 <<= 10, 2 <<= 2.

30. GE -> This token is returned when the “>>=” pattern is found. It refers “GreaterThan Equal to” between two numbers.
Example- 50 >>= 10, 22 >>= 22.

31. PRIME ->This token is returned when the “prime” pattern is found. This function will find if the given number is prime or not.

32. FACTORIAL -> This token is returned after getting the “factorial_” pattern. This function will find the given factorial of a number like - factorial_ SP 5 EP is -- 20.

33. ODDEVEN -> This token is returned after getting “oddeven_” pattern. This function will find odd or even numbers like - oddeven_ SP 12 EP is EVEN.

34. SORT -> This token is returned after getting the “sort” pattern. This function will sort numbers in ascending order like - sort 5,2,8,7,1 is -- 1,2,5,7,8.

35. MAX -> This token is returned after getting the “max” pattern. This function will find the maximum value between two numbers like- max SP 5,10 EP is --`10.

36. MIN -> This token is returned after getting “min” pattern.This function will find the minimum value between two numbers like- min SP 5,1 EP is --`1.

37. GCD -> This token is returned after getting “gcd” pattern.This function will find Greatest Common Divisor (GCD) like - gcd SP 81,153 EP is -- 9.

38. LCM-> This token is returned after getting “lcm” pattern.This function will find Least Common Multiple(LCM)like - lcm SP 8,10 EP is -- 40.

39. INC -> This token is returned after getting the “increment” pattern. It increments the value of a number. Like- increment 5 is -- 6.

40. DEC -> This token is returned after getting “decrement” pattern. It decreases the value of a number. Like- decrement 7 is -- 6.

41. AND -> This token is returned after getting the “AND_” pattern. It creates AND logic between expressions. If both are true,then the result is true. Else result is false.

42. OR -> This token is returned after getting the “OR_” pattern. It creates OR logic between expressions. If one of them is True,then the result is true. Else the result is false.

43. SQR -> This token is returned after getting “sqr” pattern. It finds the square of the number like - sqr 2 is -- 4.

44. SQRT -> This token is returned after getting the “sqrt” pattern. It finds the square root of the number like - sqrt 3 is -- 1.732051 .

45. EXP -> This token is returned after getting the “exponential” pattern.It finds the exponential of the number like - exponential 1 is -- 2.718282 .

46. POW -> This token is returned when the “pow” pattern is found.This function will find the power of the first number like- pow 4 ^ 2 is-- 16.

47. LOG -> This token is returned when the “log_” pattern is found. It will find the log base 2 value of the given number like-
log_ SP 5.2 EP is -- 1.609438.

48. SIN -> This token is returned when the “sin_” pattern is found. This function gives the SIN value of a number like -
sin_ SP 60 EP is -- 0.866027.

49. COS -> This token is returned when the “cos_” pattern is found. This function gives the SIN value of a number like -
cos_ SP 60 EP is -- 0.499998.

50. TAN -> This token is returned when the “tan_” pattern is found. This function gives the SIN value of a number like -
tan_ SP 60 EP is -- 1.732061.

51. FUNCTION -> This token is returned after getting “function_” pattern. I’ve used it for calling a simple function by it’s name(F). Like-

```
function_ F :
{{
11 // 11;;
}}

 F : ;;
```

52. TRY -> This token is returned when the “TRY” pattern is found.

53. CATCH -> This token is returned after getting the “CATCH” pattern. The function will show “Try" and "catch" that are keywords that represent the handling of exceptions due to data or coding errors during program execution.

54. CLASS -> This token is returned after getting the “CLASS” pattern. The function is used to define some classes. Inheritance is also shown here. Like - CLASS A, CLASS B.

55. {Comment} is represented by [#].* and it prints Single line comments . Example - #Mathematical Expression .

56. {CommentMulti} is represented by "##"[^##]*"##" and it prints Multiple line comments . Examples-
Increment
Decrement ##

57. {header} is represented by "import ".* and it prints Header Files. Examples- import math .

58. [+-/*<>=,=,:^] -> These return the exact things.

59. All tabs , spaces and newlines are ignored.

60. If the analyzer finds anything except the above tokens , it will print Unknown Syntax.

In my project the name of the main function is Main_start that returns VOIDMAIN as a token and I have three data types and two type arrays. For Variable Assignment I’ve used:

```
a = 80;;
b = 20;;
c = 0;;
```

--In my project the patterns for loops,condition and switch case are:

IF PART: I used 7 types of Conditions.

```
#IF | IF Wrong | IF_ELSE
iff SP 8 >> -3 EP
{{
   11 ++ 6;;
}}
iff SP 8 << 5 EP
{{
   12 // 6;;
}}
iff SP 8 >> 32 EP
{{
  14 ++ 6;;
}} els
{{
33 -- 10;;
}}
```

```
#ELSE_ELSEIf_ELSE_Ladder
iff SP +4 << 2 EP
      {{
            15 -- 10;;
      }}
elf SP 8 ++ 4 -- 2 EP
      {{
            45 // 5;;
      }}

els
      {{
          19 ++ 2;;
       }}
```

```
#Nested IF
iff SP 3 >>= 8 EP
          {{
                    iff SP 5 << 1 EP
                     {{
                              5 ++ 12;;
                     }}
```

```
                els
                {{
                        5 -- 2;;
                }}
        }}
els {{
          5 ++ 3 ;;
    }}
```

#Nested ELSE ::

```
 iff SP 3 >>= 8 EP
        {{
           5 %% 4;;
        }}
  els   {{
           iff SP 9 >> 5 EP {{
              2 ** 2;;
                }}
             els
                {{
                        18 // 2;;
                }}
      }}
```

#NESTED IF(IF_ELSEIF_IF_Ladder)

```
     iff SP 13 >>= 8 EP
        {{
           iff SP 4 <<= 2 EP
              {{
                   15 -- 10;;
              }}
            elf SP  2 -- 2 EP
              {{
                   45 // 5;;
              }}
           els
              {{
                   19 -- 11;;
              }}
        }}
   els
       {{
          90 // 2;;
       }}
```

Loop Part: I used 2 types For loop. One is like printing after a specific range like python programming & other is like normal looping.

```
#Range Looping
for_ SP 9,20,3 EP
{{
    99 ++ 1;;
}}
```

```
#Normal Looping
 for_ SP 1,5 EP
 {{
    9 %% 2;;
  }}
```

While Part:

2 types of While Loop are used for increment and decrement.

```
#increment
while_ SP 1 << 5 EP
{{
10 ++ 10;;
}}
#decrement
while_ SP 10 >> 6 EP
{{
11 -- 1;;
}}
```

Switch Part:

```
switch_ SP 3 EP
{{
CASE 1: 14 ++ 16;;

CASE 2: 90 // 9;;
```

```
CASE 3: 6 -- 30;;

DEFAULT : 22 ** 3;;
}}
```

Main Features of this compiler

1.Header file.

2. Main function.

3. Single & Multiple Line Comments.

4.Variable declaration.

5. IF ELSE Block .

6. For loop -2types

7. While loop -2 types.

8. Switch Case.

9. Variable assignment.

10. Array Declaration.

11. Print function.

12. User Defined function by calling function.

13. Try & Catch Operation.

14. Class Operation.

15. Some Built_in Functions:

a. **Factorial .**
b. **Odd & Even.**
c. **Prime.**
d. **Sorting.**

e. GCD
f. LCM.
g. Maximum & Minimum .
h. Increment & Decrement.
i. Logical Operator (AND & OR) .

16. Some Arithmetical Operation:

a. Addition.
b. Subtraction.
c. Multiplication.
d. Division.
e. MOD.

17. Some Mathematical Expression:

a. Square.
b. Square Root.
c. Exponential.
d. Power.
e. Log().
f. Sin().
g. Cos().
h. Tan().

Discussion & Conclusion:

In the program,above features are used. Unfortunately, shift reduction problems occur in the compilation time. If any grammar matches with the input text then the compiler shows the token is declared.

This compiler is designed as something like the Python & the C Programming language and in y file program is written into C programming language.