



**CSE 3212**

**Compiler Design Laboratory**

Submitted To

**DOLA DAS DIPANNITA BISWAS**

Assistant Professor Lecturer

Department of Computer Science and Department of Computer Science and

Engineering (CSE) Engineering (CSE)

Khulna University of Engineering and Khulna University of Engineering and

Technology (KUET) Technology (KUET)

Submitted By

**SHAIKH PROTHOMAA BINTE MINHAZ**

**Roll: 1807010**

Year: 3rd Semester: 2nd

Department of Computer Science and Engineering (CSE)

Khulna University of Engineering and Technology (KUET)

Date of Submission: 19 December,2022

**Objectives:**

1. To gain knowledge about semantic and synthetic rules for the compiler
2. To know about Flex and Bison
3. To know how to declare CFG (context-free grammar) for different grammar like Loop, if else pattern and so on and how they work
4. To know about shift and reduce policy of a compiler
5. To know about bottom-up parsers and how they work

**Introduction:**

A compiler is a special program that translates a source code into machine code or another programming language. A programmer writes the source code in a high-level language or human readable language such as Pascal or C using any code editor. Then the programmer runs the source code using appropriate language compiler specifying the source file’s name. A compiler that supports the source programming language reads the files, analyzes the code, and translates it into a format suitable for the target platform. Compilers that translate source code to [machine code](https://www.techtarget.com/whatis/definition/machine-code-machine-language) target specific operating systems and computer architectures.

**Flex:**

Flex (fast lexical analyzer generator) is a tool or computer program for generating lexical analyzers. It also uses for pattern matching. It reads the input stream, finds identifiers and generates token for each of it.The extension of flex file is ‘.l’. This creates a C file named lex.yy.c after extension. The function yylex() is automatically generated by the flex and this yylex() function is expected by parser to call to retrieve tokens from current token stream. The yylex() also called yywrap() function which returns 1 when the scanning is finished. The general structure of ‘.l’ file-

/\* definitions \*/

%%

/\* rules \*/

%%

/\*subroutines\*/

**Bison:**

GNU Bison, commonly known as Bison, is a parser generator. Bison uses grammar rules that allow it to analyze tokens from the ‘.l’ file and create a syntax tree. Bison command is a replacement for yacc. Yacc reads the grammar descriptions in ‘.y’ file and generates a syntax analyzer (parser), that includes function yyparse, in the file ‘.tab.c’. Included in file ‘.y’ file are token declarations. Lex reads the pattern descriptions in ‘.i’ file, includes file ‘.tab.h’ file, and generates a lexical analyzer, that includes function yylex, in file lex.yy.c. From main we call yyparse to run the compiler. Function yyparse automatically calls yylex to obtain each token.

**Instruction in cmd when we use flex and bison together:**

1. bison -d work3.y
2. flex work3.l
3. gcc lex.yy.c work3.tab.c -o w3
4. w3

**Features in compiler project:**

1. Import section or header declaration section
2. User-define function section
3. Main part of the program
4. Single-line or multiple-line comment sections
5. Variable declaration of different datatype (Integer, Float) and assignment operation
6. If else condition
7. Arithmetic and logical operation
8. Loop (for loop, while loop and do-while loop)
9. Switch-case condition
10. Read and write different values
11. Increment and decrement of variables by one
12. Built-in sine function
13. Built-in cosine function
14. Built-in tangent function
15. Built-in ln function
16. Built-in factorial function
17. Built-in Fibonacci function
18. Built-in EvenOdd function
19. Built-in Prime function
20. Built-in Leap\_year function
21. Built-in nth number summation function
22. Built-in divisors function
23. Built-in gcd function
24. Built-in lcm function
25. Built-in max function
26. Built-in min function
27. Built-in square function
28. Built-in square-root function
29. Built-in cube function

**Token:**

Token is basically a sequence of characters that are treated as a unit. . The token names are the input symbols that the parser processes.

**Token used in compiler project:**

|  |  |  |  |
| --- | --- | --- | --- |
| Serial no. | Token | Input string | Meaning of Token |
|  | MAIN\_FUNC | Start() | Main func start |
|  | DEF | Def | User define function definition |
|  | INT | Integer | Integer variable declaration |
|  | FLOAT | Float | Float variable declaration |
|  | VARIABLE | [a-z] | String for variable |
|  | READ | Read | Take input from user |
|  | WRITE | Write | Display value of variable |
|  | PLUS | + | Addition operation |
|  | MINUS | - | Subtraction operation |
|  | INTO | \* | Multiplication operation |
|  | DIV | / | Division operation |
|  | MOD | % | Module operation |
|  | POWER | ^ | Power operation |
|  | EQUAL | = | Assignment operator |
|  | PE | += | Addition and assignment |
|  | ME | -= | Subtraction and assignment |
|  | INC | ++ | Increment by one |
|  | DEC | -- | Decrement by one |
|  | LT | LT | Less than sign |
|  | GT | GT | Greater than sign |
|  | LE | LE | Less than Equal sign |
|  | GE | GE | Greater than Equal sign |
|  | NE | NE | Not Equal sign |
|  | COMA | , | Comma |
|  | SEMI | ; | End of line |
|  | COLON | : | Colon |
|  | FLPAR | ( | First left parenthesis |
|  | FRPAR | ) | First right parenthesis |
|  | SLPAR | { | Second left parenthesis |
|  | SRPAR | } | Second right parenthesis |
|  | SQR | Sqr | Square operator |
|  | CUBE | Cube | Cube operator |
|  | SQRT | Sqrt | Square-root operator |
|  | GCD | ! | Gcd operator |
|  | LCM | !! | Lcm operator |
|  | MAX | max | Max operator |
|  | MIN | min | Min operator |
|  | SINE | sin | Sine operator |
|  | COSINE | cos | Cos operator |
|  | TANGENT | tan | Tan operator |
|  | LN | ln | Ln operator |
|  | FACTORIAL | Fact | Factorial operator |
|  | FIB | Fib | Fibonacci operator |
|  | PRIME | Prime | Prime number operator |
|  | DIVISORS | Divisors | Divisors operator |
|  | Leap\_Year | Leap\_year | Leap year operator |
|  | SUMMATION | Summation | Summation operator |
|  | EVEN\_ODD | EvenOdd | Even odd operator |
|  | IF | If | If condition |
|  | ELSE | Else | Else condition |
|  | FOR | For | For loop |
|  | IN | In | In |
|  | RANGE | Range | Range |
|  | WHILE | While | While loop |
|  | DO | Do | Do |
|  | DWHILE | Dwhile | Do-while |
|  | SWITCH | Switch | Switch |
|  | CASE | Case | Case |
|  | DEFAULT | Default | Default |
|  | AND | AND | Logical AND |
|  | OR | OR | Logical OR |
|  | VARI | [a-z]+ | Token for user define function name |
|  | NUMBER | "-"?{digit}+  And "-"?{digit}\*[.][0-9]+ | Number |
|  | digit | [0-9] | Digit |
|  |  | [ ]\*[#][!][i][m][p][o][r][t][ ]+[a-zA-Z]+[.h]\* | Header file |
|  |  | @@ | Single line comment |
|  |  | @...@ | Multiple line comment |

**CFG used in compiler project with action:**

program:

    | MAIN\_FUNC SLPAR STATEMENTS SRPAR { …. }

    | DEF VARI FLPAR VARIABLES FRPAR SLPAR STATEMENTS SRPAR   {…… } program

    ;

STATEMENTS:

    | STATEMENTS STATEMENT

    | STATEMENTS BUILT\_IN\_FUNCTIONS

    | STATEMENTS LOOP

    | STATEMENTS SWITCH\_CASE

    | STATEMENTS IF\_ELSE    {…..}

    | STATEMENTS DECLARATION

    ;

STATEMENT:

     EXPRESSION SEMI        {…..}

    | VARIABLE EQUAL EXPRESSION SEMI

                            {…..}

    | VARIABLE PE EXPRESSION SEMI    {…… }

    | VARIABLE ME EXPRESSION SEMI    {….. }

    | READ VARIABLE SEMI         {….. }

    | WRITE FLPAR VARIABLE FRPAR SEMI

                            { ……}

    ;

BUILT\_IN\_FUNCTIONS:

      Leap\_Year stmt SEMI   {*……*}

    | PRIME stmt SEMI       {…..}

    | DIVISORS stmt SEMI    {…… }

    | SUMMATION stmt SEMI   {*……*}

    | EVEN\_ODD stmt SEMI    {*…..*}

    ;

LOOP:

      FOR VARIABLE EQUAL NUMBER IN RANGE FLPAR NUMBER COMA NUMBER FRPAR COLON stmt SEMI

                            {…..}

    | WHILE VARIABLE LE NUMBER COLON stmt SEMI

                            {…..}

    | WHILE VARIABLE GE NUMBER COLON stmt SEMI

                            {……}

    | DO COLON stmt SEMI DWHILE NUMBER GE NUMBER SEMI

                            {…..}

    ;

SWITCH\_CASE:

     SWITCH FLPAR NUMBER {…. } FRPAR SLPAR multiple\_case SRPAR {….}

    ;

multiple\_case:

     CASE NUMBER COLON stmt SEMI { ….. } multiple\_case

    | DEFAULT COLON stmt SEMI            {…..}

    ;

IF\_ELSE:

    IF EXPRESSION SLPAR stmt SEMI SRPAR

                            {…..}

    | IF EXPRESSION SLPAR stmt SEMI SRPAR ELSE SLPAR stmt SEMI SRPAR

                            {…..}

    | IF EXPRESSION SLPAR IF\_ELSE SRPAR ELSE SLPAR IF\_ELSE SRPAR

                            {…..}

    ;

stmt:

    EXPRESSION  { ….}

    ;

DECLARATION:

    TYPE VARIABLES SEMI  {…}

    ;

TYPE:

    FLOAT

    | INT

    ;

VARIABLES:

    VARIABLES COMA VARIABLE   {…}

    | VARIABLE              {….. }

    ;

EXPRESSION:

      EXPRESSION PLUS TERM { ….. }

    | EXPRESSION MINUS TERM { …. }

    | EXPRESSION AND TERM   { ….}

    | EXPRESSION OR TERM   { ….}

    | TERM                { …. }

    ;

TERM:

     TERM INTO FACTOR      { ….. }

    | TERM DIV FACTOR     {…..}

    | TERM MOD FACTOR   { ……}

    | TERM GCD FACTOR              { ……}

    | TERM LCM FACTOR           {….. }

    | MAX TERM FACTOR           {…. }

    | MIN TERM FACTOR           {……. }

    | FACTOR              { ….. }

    ;

FACTOR:

      G POWER FACTOR    {…..}

    | G LT FACTOR         { ….. }

    | G GT FACTOR         { …… }

    | G LE FACTOR         { ….. }

    | G GE FACTOR         { ………..}

    | G NE FACTOR         { …………. }

    | SQR G               { …………… }

    | CUBE G              { ………… }

    | SQRT G              { …………..}

    | SINE G              { ………….}

    | COSINE G            {………..}

    | TANGENT G           { ……………..}

    | LN G                { ……….}

    | FACTORIAL G         {………..}

    | FIB G              { ………..}

    | INC G              { *……..* }

    | DEC G              { *………* }

    | G                   { …… }

    ;

G:

    NUMBER                   { …… }

    | VARIABLE                 { ….. }

    |FLPAR EXPRESSION FRPAR { …..}

    ;

**Input and output of compiler project:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| #!import study.h  Def function(p,q)  {  Integer k;  k = 20;  k = k +5;  }  Start()  {  @@Variable Declaration  Float a,b,c,d,e,i,j,m;  Integer s;  Float f;  Read f;  Write(f);  a = 10;  b = 4;  c = 3;  d = a;  m = 4;  p = 3;  ++p;  --a;  b^c;  e+=a ;  e-=b ;  e = b AND 0;  e = b OR 0;  e = b\*f ;  e = a+e;  e = e-c;  f = (a^c )/e;  s = a%b;  Sqr(e);  Cube(a);  Sqrt(e);  s = a!b;  s = a!!b;  s = max a b;  s = min a b;  Fact 5;  Fib a;  sin(e);  cos(f);  tan(f);  ln(e);  Integer y;  Read y;  Leap\_year(y);  Prime(b);  Prime(c);  Divisors(d);  Summation(d);  EvenOdd(y);    If 3 GT 4  {  5\*5;  }  Else  {  7^5;  }  If 3 LT 2  {  3+2;  }  If 2 GT 2  {  If 4 GE 5  {  6\*7;  }  Else  {  2;  }  }  Else  {  If 4 NE 4  {  Sqr(8);  }  Else  {  8;  }  }    If 8 NE 9  {  If 4 LT 6  {  Fact 8;  }  Else  {  7;  }  }  Else  {  If 4 LT 4  {  6\*7;  }  }    For i=10 In Range(3,3):  (1\*5)-(7/2);    For j=10 In Range(5,1):  ((1\*5)+(7/2));    While j LE 7:  7;  While m GE 2:  tan(120);  Switch(2)  {  Case 1 : 12+2;  Case 2 : 13+3;  Case 3 : 14+4;  Default : 10;  }  Switch(3)  {  Case 1: 10+1;  Default : 100;  }    Do : 7+7 ;  Dwhile 2 GE 4;    @  Multiline Comment Section  Multiline Comment Section  @  } | This is a header file  Variable declared  k assigned 20.00  k assigned 25.00  Function Declaration!  Single Line Comment  Variable declared  Variable declared  Variable declared  Value taken from user for f  Value of f is 100.00  a assigned 10.00  b assigned 4.00  c assigned 3.00  d assigned 10.00  m assigned 4.00  p assigned 3.00  Increment(++) Value of expression : 4.00  Decrement(--) Value of expression : 9.00  Value of expression : 64.00  Result of += e assigned 10.00  Result of -= e assigned 4.00  Logical AND is False  e assigned 4.00  Logical OR is True  e assigned 4.00  e assigned 400.00  e assigned 410.00  e assigned 407.00  f assigned 2.46  Modulus assigned 2.00  Square Value of expression : 165649.00  Cubic Value of expression : 1000.00  Square-root Value of expression : 20.17  GCD values assigned 2.00  LCM values assigned 20.00  Max values assigned 10.00  Min values assigned 4.00  Factorial Value of expression : 120.00  Fibonacci series is= 0 1 1 2 3 5 8 13 21 34  Value of expression : 5.00  SINE Value of expression : 0.73  COSINE Value of expression : 1.00  TANGENT Value of expression : 0.04  Ln Value of expression : 6.01  Variable declared  Value taken from user for y  2023 not leap year  No! 4 not prime  Yes! 3 prime  All the divisors of 10 are -> 1 2 5 10  Summation of 1st 10 numbers is 55  2023 Odd Number  Value of Expression in Valid Condition: 16807.00  Condition false  Square Value of Expression in Valid Condition: 8.00  Factorial Value of Expression in Valid Condition: 40320.00  For loop Found  Value of for loop: 1.50  Value of for loop: 1.50  Value of for loop: 1.50  For loop Found  Value of for loop: 8.50  Value of for loop: 8.50  Value of for loop: 8.50  Value of for loop: 8.50  Value of for loop: 8.50  While loop found & it works properly!  Value in while loop: 7.00  Value in while loop: 7.00  Value in while loop: 7.00  TANGENT While loop found & it works properly!  Value in while loop: -1.73  Value in while loop: -1.73  Value in while loop: -1.73  CASE found 2 !  Switch done !  Executed Default  Switch done !  Value in Do while loop: 14.00  Do While loop found !  Multiline Comment  Execution Done! |

**Discussion:**

Input code of this compiler project is parsed using bottom-up parser because it is built by flex and bison. This compiler is unable to provide functionality of elseif, sort, struct features of C language. Also we cannot declare variable in char or string type using this compiler. In this compiler float variables return value in double data type. Implementation of the CFG for addition-assignment, subtraction-assignment have some problem. This compiler cannot give the output of logical not operation. This compiler is error-free while working with stated CFGs.

**Conclusion:**

Every programming language has required the use of a compiler. By this lab we able to create our own language and compiler. Our designed program may not contain all the features of a language. Though this compiler project have some limitations but this project help us to learn how to design a language and its compiler. This lesson may help us to make language for public use in a vast area.

**Reference:**

1. LEX & YACC TUTORIAL by Tom Niemann
2. <https://www.gnu.org/software/bison/>
3. <https://www.geeksforgeeks.org/flex-fast-lexical-analyzer-generator/>