Semester II **CS202: PROGRAMMING PARADIGMS & PRAGMATICS** 2022 – 2023

Final Project: Compiler for B--

AIM Learning to write a compiler for a simple language from scratch

1. Use Flex & Bison to generate lexical analyser (or scanner) and syntax analyser (or parser) that can recognize source of code of programming language B--
2. Additionally, it should generate meaningful error messages to identify various errors in the syntax of provided sample source codes of B--

INTRODUCTION

B-- is a toy programming language based on BASIC programming language.

This language is line-based, with one statement on each line. A statement is an instruction to tell the computer to do something. In B--, each line must start with a unique unsigned integer value between 1 and 9999 called the line number. These numbers specify the order of the statements and give each statement a unique identifier. Here is a sample of a small program:

Program Source Code

```
10 REM DISPLAY ODD NUMBERS FROM 1 TO 9
20 LET I=1
30 PRINT I
40 LET I=I+2
50 IF I<=9 THEN 30
60 END
```

As shown above, each line begins with a line number, and the lines are displayed in ascending order. When the program runs, it will start on the lowest numbered line. Note:

- Each line has a statement that begins with a keyword after the initial line number (e.g. LET, REM, etc)
- Tabs are not permitted in the source code
- Each item on the line is separated by at least one space
- The last line of the program must be an END and no other line can have an END statement
- B-- requires that the ASCII character set be used
- Lower-case alphabetic characters are not permitted

VARIABLES

1. Variable Names: Single Upper-Case Letter (A – Z) followed by an optional single digit (0 – 9).
1. Examples: A, F, H, A0, Z9
2. Data Types: Numeric – Integer (%), Single Precision (!), Double Precision (#) & Strings ($)
3. Type declaration uses special characters along with variable names as given above (with default being integer)
    a. Examples: P1# (double precision), N$ (string), A9 (integer), M! (single precision)

EXPRESSIONS

An expression may be a simple string or numeric constant, or a variable, or it may combine constants and variables with operators to produce a single value. There are four categories of operations:

Arithmetic Operators:

In the order of precedence (with left-to-right evaluation for equal precedence operators)

Operator	Operation	Example
()	Parenthesis	(X + Y)
^	Exponentiation	X ^ Y
-	Negation	-X
*, /	Multiplication / Division	X * Y, X / Y
+, -	Addition / Subtraction	X + Y, X - Y

Relational Operators:

Operator	Operation	Example
=	Equality	X = Y
< >	Inequality	X <> Y
<	Less Than	X < Y
>	Greater Than	X > Y
< =	Less than or equal to	X <= Y
> =	Greater than or equal to	X >= Y

Logical Operators: NOT, AND, OR, XOR

Example: X + Y < (T -1)/Z AND D > 40 (Order : Arithmetic, Relational, Logical)

Examples:

```
3*X – Y^2
A(1) + A(2) + A(3)
2^(-X)
-X/Y
```

STATEMENTS

1. The **DATA** statement is used to contain values that will be later used by the READ statement. The form of the DATA statement is: DATA value1, value2,... where value is either a numeric constant or a string constant

   Examples:

   ```
   DATA 3.14159, "PI"
   ```

2. The **DEF** statement is used to define a user-defined function of one numeric variable or a pseudo-constant. The two forms of the DEF statement is:

   DEF FNx = numeric_expression or DEF FNx (parameter) = numeric_expression

   where x is one of the single letters between A and Z inclusive, and a parameter is a scalar numeric variable. The parameter is a local variable that is only visible in the following numeric_expression, and it is distinct from any global variable of the same name. There is no way to use the global variable with the same name as the parameter in the numeric_expression of a DEF statement, although all other global variables can be used.

   Examples:

   ```
   DEF FNF(X) = X^4 - 1
   DEF FNA(X) = A*X + B
   DEF FNP = 3.14159
   ```

3. The **DIM** statement is used to specify non-default sizes of numeric arrays. The form of the DIM statement is:

   DIM declaration1, declaration2, …

   Declarations come in two forms: X(maxsubscript) or X(maxsubscript1,maxsubscript2) where X is one of the 26 possible single-letter array variable names.

   Examples:

   ```
   DIM A(6), B(10,20)
   ```

4. The **END** statement is used to specify the end of the source program, and it **must** occur exactly once in the program on the last line of the source which must have a line number higher than all other line numbers in the source program. The general form of the END statement is: END

   Example: See sample program on last page

5. The **FOR** statement is used for coding pre-test loops that use an index numeric variable. Every **FOR** statement must have a corresponding **NEXT** statement using the same index numeric variable. The FOR statement comes in two forms:

   FOR varname=expression1 TO expression2 STEP expression3 or

   FOR varname=expression1 TO expression2

   …

   …

   NEXT varname

   Examples:

   ```
   10 FOR X=1 TO 9 STEP 4
   20 PRINT X
   30 NEXT X
   40 PRINT "AFTER LOOP X IS"; X
   50 END
   ```

6. The **GOSUB** statement is used to call a subroutine. The general form of the **GOSUB** statement is: GOSUB lineno where lineno is an integer value specifying the line number of a line that exists in the program.

   The **GOTO** statement is used to branch unconditionally to a new statement. The general form of the **GOTO** statement is: GOTO lineno where lineno is an integer value specifying the line number of a line that exists in the program.

   Example: See sample program on last page

7. The **IF** statement is used to branch conditionally to a new statement. The general form of the IF statement is:

   IF condition THEN lineno

   where lineno is an integer line number that exists in the program, and condition follows this pattern:

   expression relop expression

   where relop is one of these six relational operators: < , <= , = , => , >, <>

   The types of the expressions must match so that both are numeric expressions or both are strings. For strings, only equals and not equals are permitted.

   Examples:
   ```
   IF F<>1 THEN 260
   IF A$="Y" THEN 170
   IF A$="N" THEN 340
   ```

   230 IF F=1 THEN 260

8. The **LET** statement is used to assign a value to a variable. The **LET** statement comes in two forms:

   LET string_variable = string_literal or LET numeric_variable = numeric_expression

   Examples:
   ```
   LET A(X,3) = X*Y - 1
   LET A$ = “ABC”
   LET A$ = B$
   LET P! = 3.14159
   ```

9. The **INPUT** statement is used to read data into one or more variables from the keyboard. The general form of the **INPUT** statement is: INPUT v1, v2, … where vN is a scalar numeric variable or a scalar string variable.

   Examples:
   ```
   INPUT X
   INPUT X, A$, Y(2)
   INPUT A, B, C
   ```

10. The **PRINT** statement is used to send output to the terminal. There are three general forms of the **PRINT** statement:

    PRINT

    PRINT expression1 delimiter1 … expressionN

PRINT expression1 delimiter1 … expressionN delimiter

Expressions are either numeric expressions, string literals, scalar string variables.
Two possible delimiters exist, the comma and the semicolon.

```
Examples:
PRINT X
PRINT X; (Y+Z)/2
PRINT
PRINT A$, "IS DONE"
PRINT "X = ", 10
```

11. The **REM** statement is used to add a comment to the source code of the program. The form of the **REM** statement is: REM UPPER-CASE COMMENT TEXT (**Example:** See sample program on last page)
12. The **RETURN** statement is used to exit a subroutine that was entered with **GOSUB** and continue execution on the line immediately following the **GOSUB** that invoked the subroutine. The general form of the **RETURN** statement is: RETURN

    It is a fatal error to attempt to execute a **RETURN** statement when no corresponding **GOSUB** statement was executed. (**Example:** See sample program on last page)

13. The **STOP** statement will halt execution of the program immediately. The general form of the **STOP** statement is: STOP

    Unlike the **END** statement, the **STOP** statement may occur multiple times and on any line of the program except the very last line, which must be an **END** statement. (**Ex:** See sample program on last page)

Submitting your work:

- All source files and class files as one tar-gzipped archive.
- When unzipped, it should create a directory with your ID. Example: **2008CS1001** (NO OTHER FORMAT IS ACCEPTABLE!!! Case sensitive!!!)
- Should include:
  - ✓ BMM_Scanner.l
  - ✓ BMM_Parser.y
  - ✓ BMM_Main.c/cpp (Optional)
  - ✓ CorrectSample.bmm (Sample BMM source code with no errors)
  - ✓ IncorrectSample.bmm (Sample BMM source code with errors)
  - ✓ README file (with clear instructions on how to use/run your program)
- Negative marks for any problems/errors in running your programs
- If any aspects of tasks are confusing, make an assumption and state it clearly in your **README** *file!*
- Submit/Upload it to Google Classroom

```
10 REM SAMPLE PROGRAM WITH SUBROUTINES
20 REM
30 REM A IS ARRAY TO HOLD THE DATA ITEMS
40 REM I IS THE LOOP INDEX VARIABLE
50 REM X HOLDS THE VALUE WE SEEK
60 REM F IS A FLAG, 0 MEANS NOT FOUND, 1 MEANS FOUND
70 REM N IS NUMBER OF ELEMENTS IN A
80 REM
100 DIM A(19)
110 REM
120 REM **************** MAIN *******************
130 REM
140 REM READ DATA INTO ARRAY A
150 LET N=20
160 GOSUB 380
170 REM GET VALUE FOR WHICH TO SEARCH
180 PRINT "FIND WHAT";
190 INPUT X
200 REM DO SEQUENTIAL SEARCH
210 GOSUB 470
220 REM REPORT RESULTS
230 IF F=1 THEN 260
240 PRINT X;"NOT FOUND"
250 GOTO 270
260 PRINT X;"FOUND IN SLOT";I
270 REM TRY AGAIN?
280 PRINT "TRY AGAIN";
290 INPUT A$
300 IF A$="Y" THEN 170
310 IF A$="N" THEN 340
320 PRINT "ANSWER MUST BE Y OR N!"
330 GOTO 280
340 STOP
350 REM
360 REM **************** SUBROUTINES *******************
370 REM
380 REM SUBROUTINE TO LOAD DATA FROM DATA STATEMENTS INTO A
390 REM INPUT N NUMBER OF ELEMENTS
400 REM OUTPUT A(), ARRAY WITH N ELEMENTS
410 REM
420 FOR I=0 TO N-1
430 INPUT A(I)
440 NEXT I
450 RETURN
460 REM
470 REM SUBROUTINE TO DO SEQUENTIAL SEARCH FOR X IN A
480 REM INPUT N NUMBER OF ELEMENTS
490 REM INPUT A(), ARRAY WITH N ELEMENTS
500 REM INPUT X, ELEMENT VALUE TO SEARCH FOR
510 REM OUTPUT F, 0 MEANS NOT FOUND, F=1 MEANS FOUND
520 REM OUTPUT I, INDEX OF X IN A() IF F=1, N OTHERWISE
530 REM
540 LET F=0
550 FOR I=0 TO N-1
560 IF A(I)<>X THEN 590
570 LET F=1
580 GOTO 600
590 NEXT I
600 RETURN
610 REM
620 REM ******************* DATA ***********************
630 DATA 21,85,80,14,60,76,87,49,78,81,96,25,17,22,13,91,23,62,5,57
640 REM ******************* END ************************
650 END
```