KHULNA UNIVERSITY OF ENGINEERING & TECHNOLOGY

Department of Computer Science and Engineering

CSE 3212

Compiler Design Laboratory

Project title: A simple compiler using flex and Bison.

Submitted by:

Md Tajmilur Rahman

Roll: **1807100**

3rd Year 2nd Semester

Department of Computer Science and Engineering

Khulna University of Engineering and Technology, Khulna

Submission date: 20 December, 2022

Objective

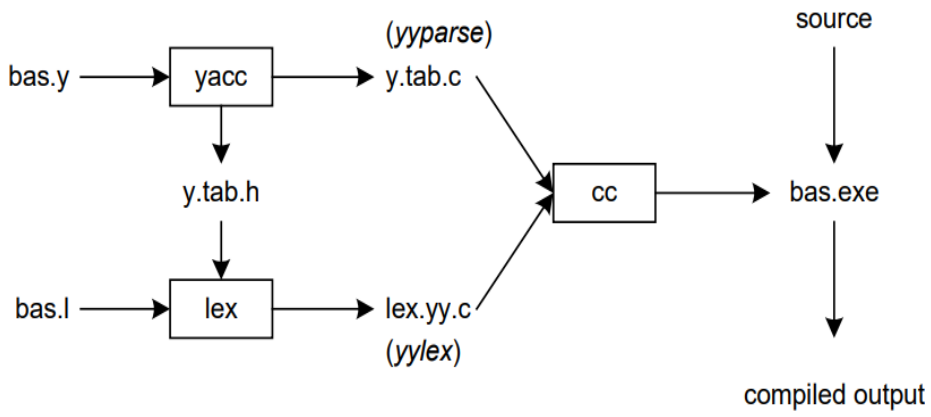
The main objective of this project is to implement the step by step operations of a complier and to learn how a basic compiler is designed also to know how the programming language designers design the compiler for their language.

To achieve these objectives, we have to fulfill some additional objectives

- How to declare CFG (context free grammar) for different grammar like if else pattern, loop and so on.
- About token and how to declare rules against token.
- How to create different and new semantic and synthetic rules for the compiler.
- About shift and reduce policy of a compiler.
- About top down and bottom up parser and how they work.
- About Flex and Bison.

Introduction

A compiler is a special program that processes statements written in a particular programming language and turns them into machine language or "code" that a computer's processor uses. Typically, a programmer writes language statements in a language such as Pascal or C one line at a time using an *editor*. The file that is created contains what are called the *source statements*. The programmer then runs the appropriate language compiler, specifying the name of the file that contains the source statements.



Figure(1): How flex and bison works to comlipe a language

Flex

Flex (fast lexical analyzer generator) is a free and open-source software alternative to lex. It is a computer program that generates lexical analyzers (also known as "scanners" or "lexers"). An input file describing the lexical analyzer to be generated named *lex.l* is written in lex language. The lex compiler transforms *lex.l* to *C* program, in a file that is always named *lex.yy.c*. The C compiler compiles the *lex.yy.c* file into an executable file called a.out. The output file a.out takes a stream of input characters and produces a stream of tokens.

```
/* definitions */
 ....
%%
/* rules */
....
%%
/* auxiliary routines */
....
```

Bison

GNU Bison, commonly known as Bison, is a parser generator that is part of the GNU Project. Bison reads a specification of a context-free language, warns about any parsing ambiguities, and generates a parser (either in C, C++, or Java) that reads sequences of tokens and decides whether the sequence conforms to the syntax specified by the grammar. **Bison** command is a replacement for the **yacc**. It is a parser generator similar to *yacc*. Input files should follow the yacc convention of ending in.*y* format.

```
/* definitions */
 ....

%%
/* rules */
```

```
....
%%

/* auxiliary routines */
....
```

Instruction in cmd when we use flex and bison together:

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

Features of this mini-compiler project:

1. **Sections:**
   - i) Header file section
   - ii) Global variable declaration section
   - iii) User defines function declaration section
   - iv) Main program
   - v) Local variable declaration

2. **Statements:**
   - i) Assignment statements
   - ii) Console output statements
   - iii) Input statement
   - iv) Conditional statements

3. **Conditionals:**
   - i) Single If condition
   - ii) If-else condition
   - iii) If-else if, else condition
   - iv) Switch case condition

4. **Loops:**
   - i) For loop for increasing and decreasing iterations
   - ii) While loop for increasing and decreasing iterations

5. Arithmetic and logical operation with expression evaluation:

i) Arithmetic: Addition, subtraction, multiplication, division, power, modular

ii) Relational: >, <, ==, <=, >=, !=

iii) Single Increment and decrement: ++ and –

iv) Multiple Increment and decrement operator: IncBy, DecBy

6. Built in functions:

i) Built-in sine function.

ii) Built-in cosine function.

iii) Built-in tan function.

iv) Built-in ln function.

v) Built-in log10 function.

vi) Built-in factorial function.

vii) Built-in oddeven function.

7. Datatypes:

i) Integer

ii) Float

iii) Character

iv) String

8. Header files:

i) Mycode’s_io.h

ii) Mycode’s_string.h

iii) Mycode’s_lib.h

iv) Mycode’s_math.h

Token

A **token** is a pair consisting of a **token** name and an optional attribute value. The **token** name is an abstract symbol representing a kind of lexical unit, e.g., a particular keyword, or sequence of input characters denoting an identifier. The **token** names are the input symbols that the parser processes.

Tokens used in this mini-project:

Bellow in the table it is shown those tokens that is used in this mini-project and their realtine meaning-

Serial no.	Token	Input string	Realtime meaning of Token
1.	NUMBER	"-"?[0-9]+	Any integer number either positive or negative.
2.	VARIABLE	[a-zA_Z0-9]+	Any string using upper case alphabets, lower case alphabets and number.
3.	UDFUNC	function_of_mycode	User defined function declaration.
4.	INT	int	Data type for integer variable.
5.	FLOAT	flt	Data type for floating-point variable.
6.	CHAR	crt	Data type for character variable.
7.	STRING	srt	Data type for string variable.
8.	VOID	nothig_to_ret	Data type for void function and parameters

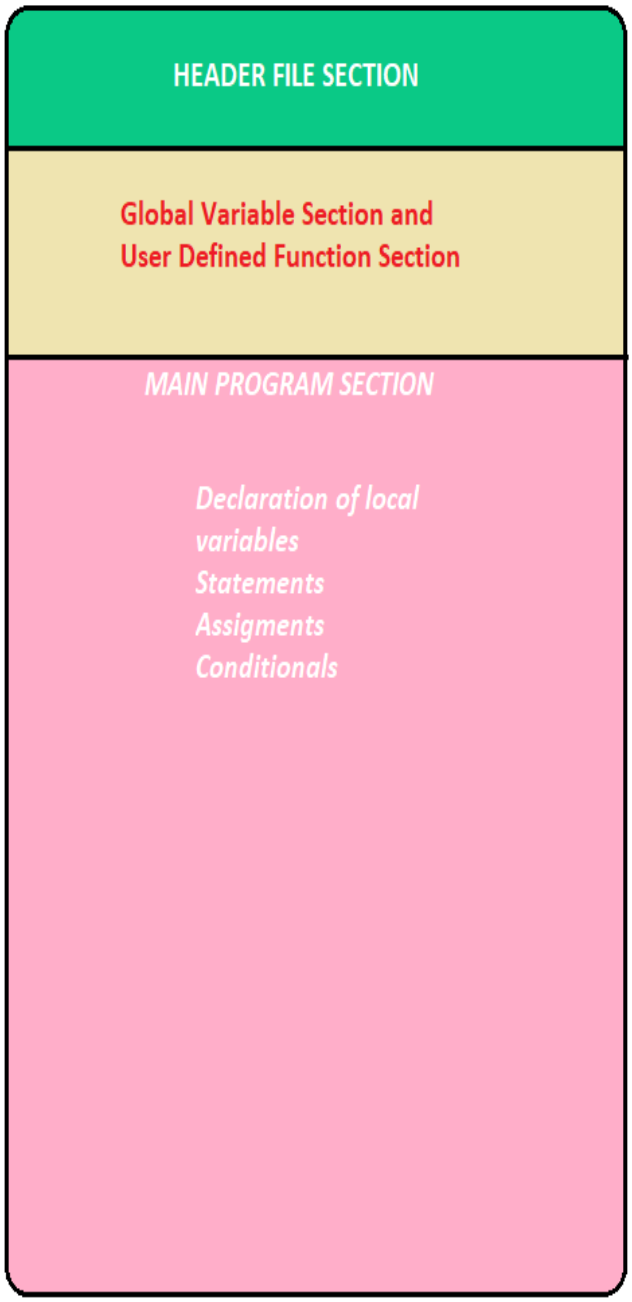
9.	INCLUDE	headerfile	Including header file
10.	HFSTD	Mycode's_io.h	Header file for standard input output
11.	HFSTR	Mycode's_string.h	Header file for string
12.	HFLIB	Mycode's_lib.h	Header file for library functions
13.	HFMATH	Mycode's_math.h	Header file for mathematical functions
14.	MAIN	Begin_with	Main function starts
15.	PRINT	show	Print and show output
16.	SCAN	scan	Scan an input
17.	RETURN	Close_with	Return a value
18.	AT	@	Type identifier
19.	IF	if	If condition start
20.	ELSE	else	Else condition and statemtnt
21.	ELIF	elif	Else-if condition start
22.	FOR	Simple_iteration	For loop line c programming
23.	WHILE	As_long_as	While loop
24.	COLON	:	Colon sign like default
25.	SWITCH	Button_of_mycode	Switch case like c programming
26.	DEFAULT	Short_ckt	Default case

27.	CASE	Key_of_mycode	Case option statements
28.	BREAK	Cut_it	Break the flow of operation
29.	INCBY	IncBy	Increment a variable value by some number
30.	DECBY	DecBy	Decrement a variable value by some number
31.	ASSIGN	<-	Assignment operator
32.	ADD1	++	Used for increment any value by one
33.	SUB1	--	Used for decrement any value by one
34.	LT	<	Less than sign
35.	GT	>	Greater than sign
36.	EQ	=	Check equal or not
37.	GTE	>=	Greater than or equal sign
38.	LTE	<=	Less than or equal sign
39.	NE	!=	Not equal symbol
40.	FACT	fact	Calculating factorial of a number
41.	SIN	Sin_of_mycode	Sine function
42.	COS	Cos_of_mycode	Cosine function
43.	TAN	Tan_of_mycode	Tangent function
44.	LN	Log_of_e	ln function
45.	LOG10	Log_of_10	Log function

46.	ODDEVEN	OddEven	For calculation Oddeven function
47.	‘+’	+	Addition operation
48.	‘-’	-	Subtraction operation
49.	‘*’	*	Multication operation
50.	‘/’	/	Division operation
51.	‘%’	%	Module operation
52.	‘(’	(	First bracket opening
53.	‘)’	)	First bracket closing
54.	‘{’	{	second bracket opening
55.	‘}’	}	second bracket closing
56.	‘^’	^	Power operation
57.	‘[’	[	Third bracket opening
58.	‘]’	]	Third bracket closing
59.	‘,’	,	Comma
60.	EOI	‘#’	End Of Instruction

Table 1. Realtime meaning of tokens that are used in project

Structure of the Program




```
headerfile<mycode's_IO.h>
headerfile<mycode's_Lib.h>

Function_of_mycode int usf( p, q)
{
    int r #
    p <- 100 #
    q <- 200 #
    r <- p+q #
    show(r)  #
}


int begin_with()
{
    int a,b,c #
    crt flag #
    String_of_mycode str #

    scan(a) #
    show ("The number is: ") #
    show(a) #

    a <- 5 #
    show(a) #

    b <- a #
    c <- a+b #
    show(c) #


    close_with 0 #

}
```

Figure (2): Structure of the input program of the compiler

CFGs used in this project

1. Grammar For Header Section:

```
/*start*/
program:
        include before start    {printf("program compiled successfully.\n");}
      |
    ;

include: include INCLUDE LT hf GT     {printf("header files included.\n");}
         | INCLUDE LT hf GT
         ;
hf: HFSTD | HFSTR | HFMATH | HFLIB
   ;
/* header include ended*/
```

2. Grammar for Global declaration and user defined function:

```
before: declare
        |func
        |
        ;
func: UDFUNC type VARIABLE '(' parameters ')' '{' lines '}'
   {
         printf("FUNC Declared!\n\n");
   }
   ;

parameters      :
   parameters ',' VARIABLE
         {
          if(func_here($3)==1)
                       printf("\nparameters already exists!");
          else
                         assign_func($3);
                    assign($3);

         }
```

```
| VARIABLE
        {
           if(func_here($1)==1)
                 printf("\nparameters already exists!");
          else
                      assign_func($1);
                       assign($1);
        } ;
```

3. Grammar for Main Function

```
/* Main function stared here */
start:
      type MAIN '(' ')' '{' lines '}' {printf("main function compiled
successfully.\n");}
      |
      ;
type: INT | FLOAT | VOID | CHAR | STRING  ;

lines:
      declare lines | stmt lines
      | {$$=1;}
    ;
```

4. Grammar for Declaration of Variable

```
//declaration
declare:
    type id EOI         { printf("\nValid declaration!\n"); } ;

id  :
    id ',' VARIABLE
        {
         if(isdeclared($3)==1)
                  printf("\nDouble Declaration!");
         else     assign($3);
        }
    | VARIABLE
        {
           if(isdeclared($1)==1)
                  printf("\nDouble Declaration! \n");
          else
                      assign($1);
        }
    ;
```

Grammar for Statements:

```
stmt:
      PRINT '(' pcontent ')' EOI {printf("Print successful.\n");}
      |
      SCAN '(' VARIABLE ')' EOI {
      if(isdeclared($3)==1)
            {
            printf("Scan is successful.\n");
        }
      }
      | VARIABLE ASSIGN expr EOI
        {
            if(setval($1,$3)==0)
            {
                $$=0;
                printf("\nNot declared\n");
            }
            else
            {
                store[getval($1)]= $3;
                $$=$3;
            }
        }
      |condstmt {printf("Conditional statement successfully complied.\n");}

      |loops

      |switch_case

      | stmt  RETURN NUMBER EOI
      ;
```

Discussion:

Our basic compiler is designer only to examine that the input file of a source code is maintaining the structure defined by the CFG in the bison file. No execution of any statement is not done here. The expressions are evaluated only to demonstrate the type of the non-terminals. The input code is parsed using a bottom-up parser in this compiler. Because it is only built with flex and bison, this compiler is unable to provide original functionality such as if-else, loop, and switch case features. However, when creating code in this compiler-specific style, header declaration is not required but if we need, we can use header file. The float variable always returns a value in the double data type, which is a compiler requirement. Any variable's string value is not stored by this compiler. This compiler is error-free while working with the stated CFG format.

Conclusion:

Compiler is nothing but a program which takes another program as input and translate it in executable files. Every programming language has required the use of a compiler. Designing a new language without a solid understanding of how a compiler works may be a challenging endeavor. Several issues were encountered during the design phase of this compiler, such as loop, if-else, switch case functions not working as they should owe to bison limitations, character and string variable values not being stored properly, and so on. In the end, some of these issues were resolved, and given the constraints, this compiler performs admirably.

References:

- Principles of Compiler Design By Alfred V.Aho & J.D Ullman
- https://whatis.techtarget.com/definition/compiler
- http://web.mit.edu/gnu/doc/html/bison_5.html
- https://www.oreilly.com/library/view/flex-bison/9780596805418/ch01.html