

## Experiment No. 3

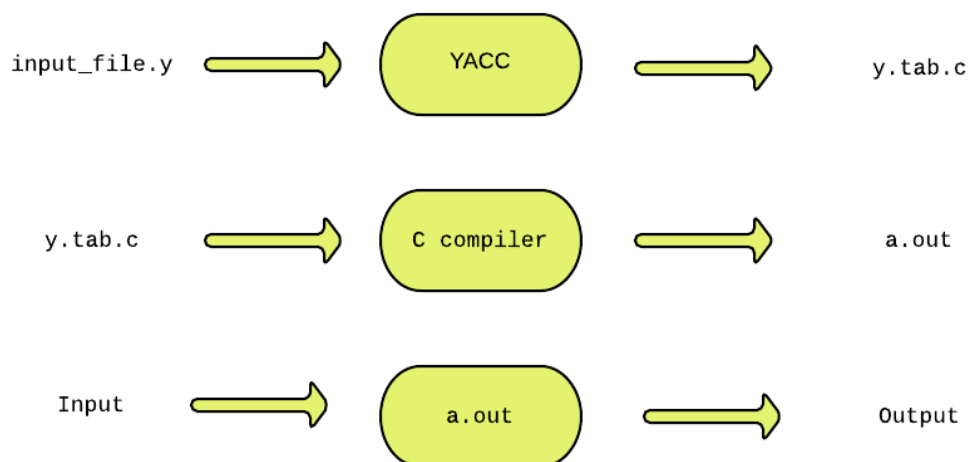
**Aim:** To perform arithmetic operations using YACC

**Requirements:** Lex, Yacc and C

**Theory:**

### Introduction to YACC

YACC (Yet Another Compiler Compiler) is a tool used to generate a parser. This document is a tutorial for the use of YACC to generate a parser for ExpL. YACC translates a given Context Free Grammar (CFG) specifications (input in input\_file.y) into a C implementation (y.tab.c) of a corresponding push down automaton (i.e., a finite state machine with a stack). This C program when compiled, yields an executable parser.



The source SIL program is fed as the input to the generated parser ( a.out ). The parser checks whether the program satisfies the syntax specification given in the input\_file.y file.

YACC was developed by Stephen C. Johnson at Bell labs.

## Parser

A parser is a program that checks whether its input (viewed as a stream of tokens) meets a given grammar specification. The syntax of SIL can be specified using a Context Free Grammar. As mentioned earlier, YACC takes this specification and generates a parser for SIL.

### Context Free Grammar (CFG)

A context free grammar is defined by a four tuple  $(N, T, P, S)$  - a set  $N$  of non-terminals, a set  $T$  of terminals (in our project, these are the tokens returned by the lexical analyzer and hence we refer to them as tokens frequently), set  $P$  of productions and a start variable  $S$ . Each production consists of a non-terminal on the left side (head part) and a sequence of tokens and non-terminals (of zero or more length) on the right side (body part).

### The structure of YACC programs

A YACC program consists of three sections: Declarations, Rules and Auxiliary functions. (Note the similarity with the structure of LEX programs).

#### DECLARATIONS

%%

#### RULES

%%

#### AUXILIARY FUNCTIONS

### 1 Declarations

The declarations section consists of two parts: (i) C declarations and (ii) YACC declarations . The C Declarations are delimited by `%{` and `%}`. This part consists of all the declarations required for the C code written in the *Actions* section and the *Auxiliary functions* section.

YACC copies the contents of this section into the generated y.tab.c file without any modification.

## **2 Rules**

A rule in a YACC program comprises of two parts (i) the production part and (ii) the action part. In this project, the syntax of SIL programming language will be specified in the form of a context free grammar.

### **2.1 Productions**

Each production consists of a production head and a production body.

### **2.2 Actions**

The action part of a rule consists of C statements enclosed within a '{' and '}'. These statements are executed when the input is matched with the body of a production and a reduction takes place.

## **3 Auxiliary Functions**

The Auxiliary functions section contains the definitions of three mandatory functions main(), yylex() and yyerror(). You may wish to add your own functions (depending on the requirement for the application) in the y.tab.c file. Such functions are written in the auxiliary functions section. The main() function must invoke yyparse() to parse the input.

### **Code:**

Lex:

```
% {  
#include<stdio.h>  
#include "calc.tab.h"  
extern int yylval;  
% }
```

```

%%
[0-9]+ {
    yylval=atoi(yytext);
    return NUMBER;
}
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
    return 1;
}

```

Yacc:

```

% {
    #include<stdio.h>
    int flag=0;

% }
%token NUMBER

%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
%%
ArithmeticExpression: E{
    printf("\nResult=%d\n",$$);
    return 0;
}
E:E+'E' {$$=$1+$3;}
|E-'E' {$$=$1-$3;}
|E'*E' {$$=$1*$3;}
|E'/E' {$$=$1/$3;}
|E'%E' {$$=$1%$3;}
|('E') {$$=$2;}
|NUMBER {$$=$1;}
;
%%

void main()
{
    printf("\nEnter Any Arithmetic Expression which can have operations Addition,
Subtraction, Multiplication, Divison, Modulus and Round brackets:\n");
    yyparse();
    if(flag==0)
        printf("\nEntered arithmetic expression is Valid\n\n");
}

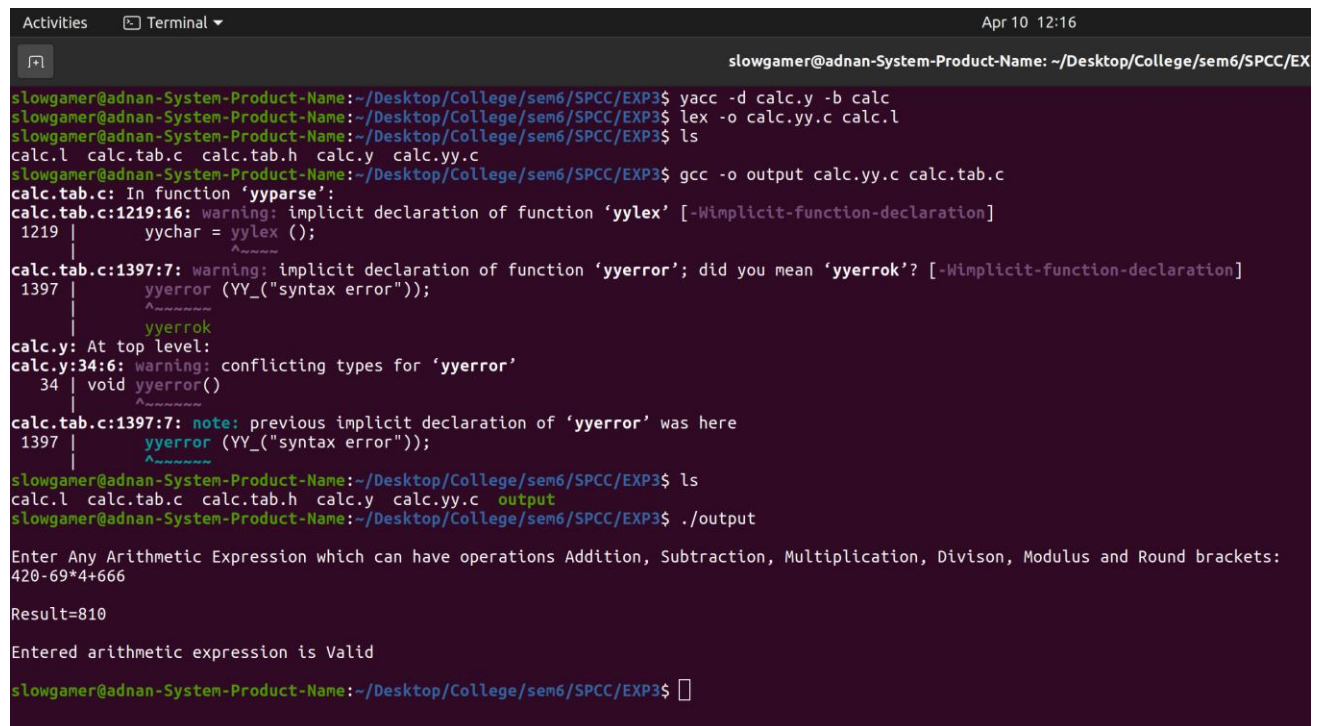
```

```

}
void yyerror()
{
    printf("\nEntered arithmetic expression is Invalid\n\n");
    flag=1;
}

```

### Output:



```

slowgamer@adnan-System-Product-Name: ~/Desktop/College/sem6/SPCC/EXP3$ yacc -d calc.y -b calc
slowgamer@adnan-System-Product-Name: ~/Desktop/College/sem6/SPCC/EXP3$ lex -o calc.yy.c calc.l
slowgamer@adnan-System-Product-Name: ~/Desktop/College/sem6/SPCC/EXP3$ ls
calc.l  calc.tab.c  calc.tab.h  calc.y  calc.yy.c
slowgamer@adnan-System-Product-Name: ~/Desktop/College/sem6/SPCC/EXP3$ gcc -o output calc.yy.c calc.tab.c
calc.tab.c: In function 'yyparse':
calc.tab.c:1219:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
1219 |     yychar = yylex ();
     |                ^~~~~
calc.tab.c:1397:7: warning: implicit declaration of function 'yyerror'; did you mean 'yyerrok'? [-Wimplicit-function-declaration]
1397 |     yyerror (YY_("syntax error"));
     |     ^~~~~~
     |     yyerrok
calc.y: At top level:
calc.y:34:6: warning: conflicting types for 'yyerror'
  34 | void yyerror()
     |      ^~~~~~
calc.tab.c:1397:7: note: previous implicit declaration of 'yyerror' was here
1397 |     yyerror (YY_("syntax error"));
     |     ^~~~~~
slowgamer@adnan-System-Product-Name: ~/Desktop/College/sem6/SPCC/EXP3$ ls
calc.l  calc.tab.c  calc.tab.h  calc.y  calc.yy.c  output
slowgamer@adnan-System-Product-Name: ~/Desktop/College/sem6/SPCC/EXP3$ ./output

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:
420-69*4+666

Result=810

Entered arithmetic expression is Valid

slowgamer@adnan-System-Product-Name: ~/Desktop/College/sem6/SPCC/EXP3$ 

```

**Conclusion:** We have successfully implemented program to perform arithmetic operations using YACC.