Assignment No: 2

Problem Statement: Implement a parser for an expression grammar using YACC and LEX for the subset of C. Cross check your output with Stanford LEX and YACC.

Objective: To implement parser for an expression grammar
To understand the use of YACC tool in syntax analysis phase.

Outcome: Implemetation of a parser for a subset of C language using Lex and Yacc tool.

Software Requirements : Lex, Yacc,GCC

Hardware Requirements: 4GB RAM, 500 GB HDD

Theory:

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.

YACC (yet another compiler-compiler) is an LALR(LookAhead, Left-to-right, Rightmost derivation producer with 1 lookahead token) parser generator. YACC was originally designed for being complemented by Lex.

YACC input file is divided in three parts.

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

%%
/* rules */
....
%%

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

Definition Part:

- The definition part includes information about the tokens used in the syntax definition:

  ```
  %token NUMBER
  %token ID
  ```

- Yacc automatically assigns numbers for tokens, but it can be overridden by

%token NUMBER 621

- Yacc also recognizes single characters as tokens. Therefore, assigned token numbers should no overlap ASCII codes.
- The definition part can include C code external to the definition of the parser and variable declarations, within **%{** and **%}** in the first column.
- It can also include the specification of the starting symbol in the grammar:

  %start nonterminal

Rule Part:

- The rules part contains grammar definition in a modified BNF form.
- Actions is C code in { } and can be embedded inside (Translation schemes).

Input File: Auxiliary Routines Part:

- The auxiliary routines part is only C code.
- It includes function definitions for every function needed in rules part.
- It can also contain the main() function definition if the parser is going to be run as a program.
- The main() function must call the function yyparse().

Input File:

- If yylex() is not defined in the auxiliary routines sections, then it should be included:

  #include "lex.yy.c"

- YACC input file generally finishes with:

  .y

Output Files:

- The output of YACC is a file named **y.tab.c**
- If it contains the **main()** definition, it must be compiled to be executable.
- Otherwise, the code can be an external function definition for the function **int yyparse()**
- If called with the **–d** option in the command line, Yacc produces as output a header file **y.tab.h** with all its specific definition (particularly important are token definitions to be included, for example, in a Lex input file).
- If called with the **–v** option, Yacc produces as output a file **y.output** containing a textual description of the LALR(1) parsing table used by the parser. This is useful for tracking down how the parser solves conflicts.

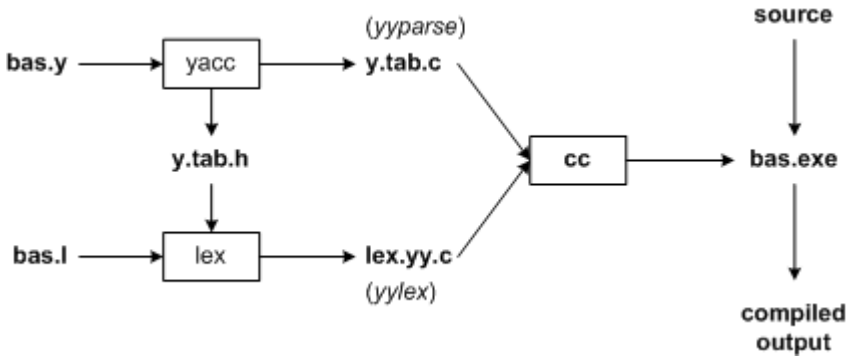
Methods:

yyparse()

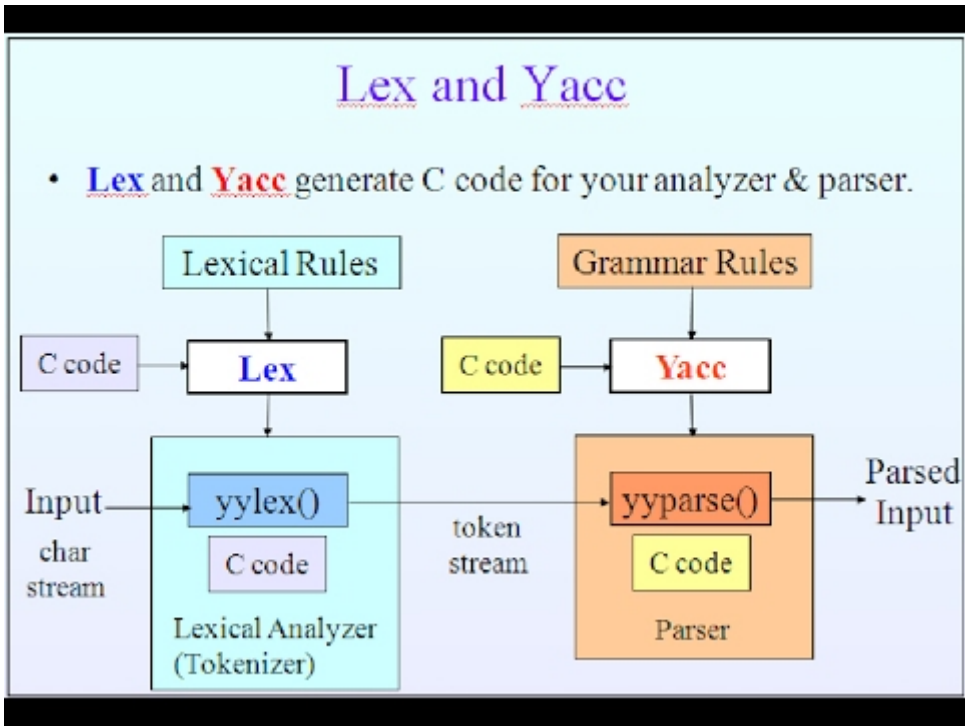
The y.tab.c file contains a function yyparse() which is an implementation (in C) of a push down automation. yyparse() is responsible for parsing the given input file. The function yylex() is invoked by yyparse() to read tokens from the input file. The yyparse() function is automatically generated by YACC in the y.tab.c file.

For Compiling YACC Program:

1. Write lex program in a file file.l and yacc in a file file.y
2. Open Terminal and Navigate to the Directory where you have saved the files.
3. type lex file.l
4. type yacc file.y
5. type cc lex.yy.c y.tab.h -ll
6. type ./a.out



LEX recognizes regular expressions, whereas YACC recognizes entire grammar. LEX divides the input stream into tokens, while YACC uses these tokens and groups them together logically. LEX and YACC work together to analyze the program syntactically. The YACC can report conflicts or ambiguities (if at all) in the form of error messages.



Supporting C-Routines Section:

The third part of a Yacc specification consists of supporting C-routines. YACC generates a single function called yyparse(). This function requires no parameters and returns either a 0 on success, or 1 on failure. If syntax error over its return 1.The special function yyerror() is called when YACC encounters an invalid syntax. The yyerror() is passed a single string (char)argument.
This function just prints user defined message like:

```
yyerror (char err)
{
printf (“Divide by zero”);
}
```

When LEX and YACC work together lexical analyzer using yylex () produce pairs consisting of a token and its associated attribute value. If a token such as DIGIT is returned, the token value associated with a token is communicated to the parser through a YACC defined variable yylval.
We have to return tokens from LEX to YACC, where its declaration is in YACC. To link this LEX program include a y.tab.h file, which is generated after YACC compiler the program using
– d option.

Conclusion:

Thus using yacc tool we’ve implemented a parser to check whether the entered arithmetic expression is valid or not.

Code:

cal.l-

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

%%
```

```
[0-9]+ { yylval=atoi(yytext); return NUMBER; }

[\n] return 0;
. return yytext[0];

%%

int yywrap()
{
return 1;
}
```

cal.y-

```
%{
#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;}
;

%%
int main()
{
  yyparse();
  if(flag==0)
```

```
   printf("\nEntered arithmetic expression is Valid\n\n");
  return 0;
}

int yyerror()
{
   printf("\nEntered arithmetic expression is Invalid\n\n");
   flag=1;
   return 0;
}
```

Output-

```
(base) dell@dell-Inspiron-15-3567:~/BEALL2/Compiler/Compiler$ lex cal.l
(base) dell@dell-Inspiron-15-3567:~/BEALL2/Compiler/Compiler$ yacc -d cal.y
(base) dell@dell-Inspiron-15-3567:~/BEALL2/Compiler/Compiler$ gcc y.tab.c lex.yy.c -ll
-ly -lm
(base) dell@dell-Inspiron-15-3567:~/BEALL2/Compiler/Compiler$ ./a.out
2+3

Result=5

Entered arithmetic expression is Valid

(base) dell@dell-Inspiron-15-3567:~/BEALL2/Compiler/Compiler$ ./a.out

+33

Entered arithmetic expression is Invalid

*/
```