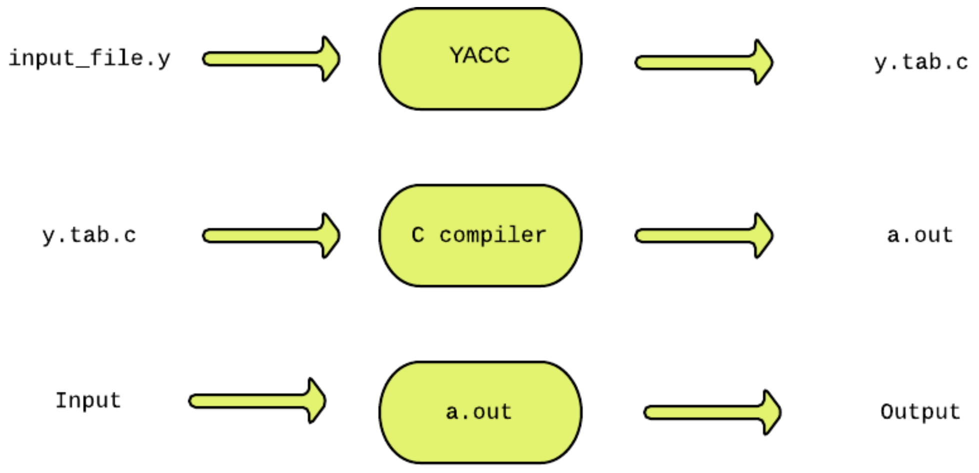
Practical No. 8

Aim: YACC Program specifications

Theory:

1. 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).
2. This C program when compiled, yields an executable parser.



3. 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.
4. The structure of YACC programs: A YACC program consists of three sections: Declarations, Rules and Auxiliary functions.

```
DECLARATIONS

%%

RULES

%%

AUXILIARY FUNCTIONS
```

5. 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.

6. 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. A rule in YACC is of the form:

```
production_head : production_body {action in C } ;
```

7. Productions: Each production consists of a production head and a production body

```
expr : expr '+' expr
```

8. Actions: The action part of a rule consists of C statements enclose within a '{' and '}'. These statements are executed when the input is matched with the body of a production and a reduction takes place. The notion of a reduction will be explained later.

```
expr: DIGIT {printf("NUM%d ",pos);}
```

9. 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 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. You will need to write your supporting functions later in this project.
10. Infix to Postfix program: When yacc_file.y is fed to YACC, it generates a y.tab.c file. When compiled, this program yields a parser. The generated parser uses shift-reduce parsing to parse the given input. Yacc copies the C declarations (in the Declaration section of input_file.y) and all the auxiliary functions (in the Auxiliary functions section of input_file.y) directly into y.tab.c without any modification. In addition to these, YACC generates the definition of yyparse() in y.tab.c.
11. It is important to understand that, y.tab.c contains the following :
    a. The C declarations from the input_file.y file.
    b. Generated yyparse() definition.
    c. All the auxiliary functions from the input_file.y
12. yyparse() is the function that parses the given input using shift-reduce parsing. When the reduction of a handle takes place, yyparse() executes the action (specified in the action part of the rule) corresponding to the handle's production in the yacc program. On successful parsing of the given input, yyparse() returns 0. If yyparse() fails to parse the given input, it returns 1.

Program for YACC:

```
%{
#include <ctype.h>
#include <stdio.h>
#define YYSTYPE double /* double type for yacc stack */
%}
%%
Lines : Lines S '\n' { printf("OK \n"); }
     | S '\n’
     | error '\n' {yyerror("Error: reenter last line:");
                                                   yyerrok; };
S    : '(' S ')’
     | '[' S ']’
     | /* empty */ ;
%%

#include "lex.yy.c"
void yyerror(char * s)
/* yacc error handler */
{
fprintf (stderr, "%s\n", s);
}
int main(void)
{
return yyparse();
}
```

Program for lex:

```
%{
%}
%%
[ \t] { /* skip blanks and tabs */ }
\n|. { return yytext[0]; }
%%
```

Conclusion:

So, in this practical we have successfully analyzed and created a detailed report on YACC parser generator that represents the structure of the YACC program and explains each part of its specification.