Exp No: 6.1

STUDY OF YACC

AIM :

To study the working of YACC

INTRODUCTION:

Yacc is the abbreviation of ***Yet Another Compiler-Compiler.*** It is a LALR parser generator, generating a parser, the part of a compiler that tries to make syntactic sense of the source code. Yacc provides a general tool for imposing structure on the input to a computer program. The Yacc user prepares a specification of the input process; this includes rules describing the input structure, code to be invoked when these rules are recognized, and a low-level routine to do the basic input. Yacc then generates a function to control the input process. This function, called a parser, calls the user-supplied low-level input routine (the lexical analyzer) to pick up the basic items (called tokens) from the input stream. These tokens are organized according to the input structure rules, called grammar rules; when one of these rules has been recognized, then user code supplied for this rule, an action, is invoked; actions have the ability to return values and make use of the values of other actions.

Using the lex Program with the yacc Program

When used alone, the **lex** program generator makes a lexical analyzer that recognizes simple, one-word input or receives statistical input. You can also use the **lex** program with a parser generator, such as the **yacc** command. The **yacc** command generates a program, called a parser, that analyzes the construction of more than one-word input. This parser program operates well with the lexical analyzers that the **lex** command generates. The parsers recognize many types of grammar with no regard to context. These parsers need a preprocessor to recognize input tokens such as the preprocessor that the **lex** command produces.

The **lex** program recognizes only extended regular expressions and formats them into character packages called tokens, as specified by the input file. When using the **lex** program to make a lexical analyzer for a parser, the lexical analyzer (created from the **lex** command) partitions the input stream. The parser (from the **yacc** command) assigns structure to the resulting pieces. You can also use other programs along with the programs generated by either the **lex** or **yacc** commands.

A token is the smallest independent unit of meaning as defined by either the parser or the lexical analyzer. A token can contain data, a language keyword, an identifier, or other parts of a language syntax.

The **yacc** program looks for a lexical analyzer subroutine named **yylex**, which is generated by the **lex** command. Normally, the default main program in the **lex** library calls the **yylex** subroutine. However, if the **yacc** command is loaded and its main program is used, **yacc** calls the **yylex** subroutine. In this case, each **lex** rule should end with:

return(token);

,where the appropriate token value is returned.

The **yacc** command assigns an integer value to each token defined in the **yacc** grammar file through a **#define** preprocessor statement. The lexical analyzer must have access to these macros to return the tokens to the parser. Use the **yacc -d** option to create a **y.tab.h** file, and include the **y.tab.h** file in the **lex** specification file by adding the following lines to the definition section of the **lex** specification file:

```
%{
#include "y.tab.h"
%}
```

Alternately, you can include the **lex.yy.c** file in the **yacc** output file by adding the following line after the second %% (percent sign, percent sign) delimiter in the **yacc** grammar file:

#include "lex.yy.c"

The **yacc** library should be loaded before the **lex** library to get a main program that invokes the **yacc** parser. You can generate **lex** and **yacc** programs in either order.

Creating a Parser with the yacc Program:

The **yacc** program creates parsers that define and enforce structure for character input to a computer program. To use this program, you must supply the following inputs:

Grammar file A source file that contains the specifications for the language to recognize.

This file also contains the **main**, **yyerror**, and **yylex** subroutines. You must supply these subroutines.

main	A C language subroutine that, as a minimum, contains a call to the **yyparse** subroutine generated by the **yacc** program. A limited form of this subroutine is available in the **yacc** library.
yyerror	A C language subroutine to handle errors that can occur during parser operation. A limited form of this subroutine is available in the **yacc** library.
yylex	A C language subroutine to perform lexical analysis on the input stream and pass tokens to the parser. You can generate this lexical analyzer subroutine using the **lex** command.

When the **yacc** command gets a specification, it generates a file of C language functions called **y.tab.c**. When compiled using the **cc** command, these functions form the **yyparse** subroutine and return an integer. When called, the **yyparse** subroutine calls the **yylex** subroutine to get input tokens. The **yylex** subroutine continues providing input until either the parser detects an error or the **yylex** subroutine returns an end-marker token to indicate the end of operation. If an error occurs and the **yyparse** subroutine cannot recover, it returns a value of 1 to **main**. If it finds the end-marker token, the **yyparse** subroutine returns a value of 0 to **main**.

Using the yacc Grammar File:

A **yacc** grammar file consists of three sections:

- Declarations
- Rules
- Programs

Two adjacent %% (double percent signs) separate each section of the grammar file. To make the file easier to read, put the %% on a line by themselves. A complete grammar file looks like:

```
declarations
%%
rules
%%
programs
```

The declarations section may be empty. If you omit the programs section, omit the second set of %%. Therefore, the smallest **yacc** grammar file is:

```
%%
rules
```

Steps for Compiling the Program:

Perform the following steps, in order.

1: Write the YACC grammer file with the **.y** extension **(example calc.y)** and the lex specification file with the extension **.l.**

2: Process the **yacc** grammar file using the **-d** optional flag (which tells the **yacc** command to create a file that defines the tokens used in addition to the C language source code):

yacc -d calc.y

1. Use the **ls** command to verify that the following files were created:

   y.tab.c The C language source file that the **yacc** command created for the parser.
   y.tab.h A header file containing defines statements for the tokens used by the parser.

2. Process the **lex** specification file:

   lex calc.l

3. Use the **ls** command to verify that the following file was created:

   lex.yy.c The C language source file that the **lex** command created for the lexical analyzer.

4. Compile and link the two C language source files:

   gcc y.tab.c lex.yy.c -lfl

5. Use the **ls** command to verify that the following files were created:

   y.tab.o The object file for the **y.tab.c** source file
   lex.yy.o The object file for the **lex.yy.c** source file
   a.out The executable program file

6. To then run the program directly from the **a.out** file, enter **./ a.out**

RESULT:

Studied the operations of YACC