**SYSTEM SOFTWARE**

**MODULE V**

## 1. What is YACC? Explain

- YACC stands for **Yet Another Compiler Compiler**.
- YACC provides a tool to produce a **parser** for a given grammar.
- YACC is a program designed to compile a LALR (1) grammar.
- It is used to produce the source code of the syntactic analyzer of the language produced by LALR (1) grammar.
- The input of YACC is the **rule or grammar** and the output is a **C program**.

Some points about YACC:

**Input: A CFG- file.y**

**Output: A parser y.tab.c (yacc)**

- The output file "file.output" contains the **parsing tables.**
- The file "file.tab.h" contains declarations.
- The parser called the yyparse ().
- Parser expects to use a function called yylex () to get tokens.

The basic operational sequence is as follows:

**gram.y**

This file contains the desired grammar in YACC format.

**yacc**

It shows the YACC program.

**y.tab.c**

It is the c source program created by YACC.

**cc or gcc** C

Compiler

**a.out**

Executable file that will parse grammar given in gram.y

## 2. What is LEX ? Explain.

- Lex is a program that generates lexical analyzer. It is used with YACC parser generator.
- The lexical analyzer is a program that transforms an input stream into a sequence of tokens.
- It reads the input stream and produces the source code as output through implementing the lexical analyzer in the C program.

The function of Lex is as follows:

- Firstly lexical analyzer creates a program lex.1 in the Lex language. Then Lex compiler runs the lex.1 program and produces a C program lex.yy.c.
- Finally C compiler runs the lex.yy.c program and produces an object program a.out.
- a.out is lexical analyzer that transforms an input stream into a sequence of tokens.

Lex file format

A Lex program is separated into **three** sections by %% delimiters. The format of Lex source is as follows:

1. { definitions }
   %%
2. { rules }
   %%
3. { user subroutines }

**Definitions** include declarations of **constant, variable and regular definitions**.

**Rules** define the statement of form p1 {action1} p2 {action2}..... pn {actionn}. Where **pi** describes the regular expression and **action i** describes the actions what action the lexical analyzer should take when pattern pi matches a lexeme.

**User subroutines** are auxiliary procedures needed by the actions. The subroutine can be loaded with the lexical analyzer and compiled separately.

## 3. What is the difference between LEX and YACC?

- Lex is used to split the text into a list of tokens, what text become token can be specified using regular expression in lex file.

- Yacc is used to give some structure to those tokens. For example in

Programming languages, we have assignment statements like int a = 1 + 2; and i want to make sure that the left hand side of '=' be an identifier and the right side be an expression [it could be more complex than this]. This can be coded using a CFG rule and this is what you specify in yacc file and this you cannot do using lex (lex cannot handle recursive languages).

- A typical application of lex and yacc is for implementing programming languages.
- Lex tokenizes the input, breaking it up into keywords, constants, punctuation, etc.
- Yacc then implements the actual computer language; recognizing a for statement, for instance, or a function definition.
- Lex and yacc are normally used together. This is how you usually construct an application using both:
- Input Stream (characters) -> Lex (tokens) -> Yacc (Abstract Syntax Tree) > Your Application

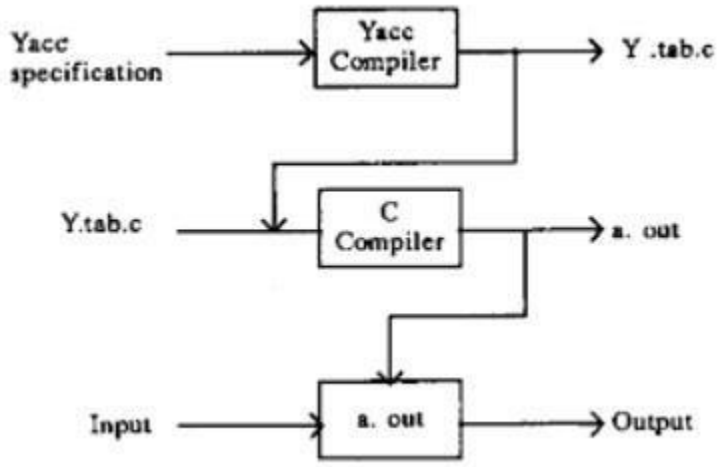
## 4. Explain How does this yacc works?

- yacc is designed for use with C code and generates a parser written in C.
- The parser is configured for use in conjunction with a lex-generated scanner and relies on standard shared features (token types, yylval, etc.) and calls the function yylex as a scanner coroutine.
- You provide a grammar specification file, which is traditionally named using a .y extension.
- You invoke yacc on the .y file and it creates the y.tab.h and y.tab.c files containing a thousand or so lines of intense C code that implements an efficient LALR (1) parser for your grammar, including the code for the actions you specified.
- The file provides an extern function yyparse.y that will attempt to successfully parse a valid sentence.

- You compile that C file normally, link with the rest of your code, and you have a parser! By default, the parser reads from stdin and writes to stdout, just like a lex-generated scanner does.

### ○ What is YACC?

- Yacc is officially known as a "parser".
- It's job is to analyse the structure of the input stream, and operate of the "big picture".
- In the course of it's normal work, the parser also verifies that the input is syntactically sound.
- Consider again the example of a C-compiler. In the C-language, a word can be a function name or a variable, depending on whether it is followed by a (or a = There should be exactly one } for each { in the program.
- YACC stands for "Yet another Compiler Compiler". This is because this kind of analysis of text files is normally associated with writing compilers.




## 6. Explain How does this Lexical analyzer work?

- The lexical analyzer created by Lex behaves in concert with a parser in the following manner.
- When activated by the parser, the lexical analyzer begins reading its remaining input, one character at a time, until it has found the longest prefix of the input that is matched by one of the regular expressions p.
- Then it executes the corresponding action. Typically the action will return control to the parser.
- However, if it does not, then the lexical analyzer proceeds to find more lexemes, until an action causes control to return to the parser.
- The repeated search for lexemes until an explicit return allows the lexical analyzer to process white space and comments conveniently.
- The lexical analyzer returns a single quantity, the token, to the parser. To pass an attribute valu e with information about the lexeme, we can set the global variable yylval.
- e.g. Suppose the lexical analyzer returns a single token for all the relational operators, in which case the parser won't be able to distinguish between " <=","=>","<",">","==" etc. We can set yylval appropriately to specify the nature of the operator.

**The two variables yytext and yyleng**

1. **yytext is a variable that is a pointer to the first character of the lexeme.**
2. **yyleng is an integer telling how long the lexeme is.**

   **(Lexemes** and Tokens. A **Lexeme** is a string of characters that is a lowest-level syntatic unit in the programming language. These are the "words" and punctuation of the programming language. A **Token** is a syntactic category that forms a class of **lexemes)**