

### Practical-5

**AIM:** Understand the structure of the YACC program with example.

#### Theory:

#### Q-1 What is YACC Compiler:

YACC in compiler design, also known as Yet Another Compiler Compiler is used to produce the source code of the syntactic analyzer of the language produced by Look Ahead Left to Right LALR (1) parser generator. As an input, a parser generator takes input a syntax specification and produces a procedure for recognizing that language as output.

Stephen C. Johnson developed YACC in compiler design in the early 1970s. Initially, the YACC was written in the B programming language and was soon rewritten in C. It was originally designed for being complemented by Lex.

In addition to that, YACC was also rewritten in OCaml, Ratfor, ML, ADA, Pascal, Java < Python, Ruby and Go.

#### Q-2 Structure of yacc:

##### Structure:

A YACC program consists of three sections: Declarations, Rules and Auxiliary functions.

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

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

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

**Definitions:** these include the header files and any token information used in the syntax. These are located at the top of the input file. Here, the tokens are defined using a modulus sign. In the YACC, numbers are automatically assigned for tokens.

#### Example:

```
%token ID
{% #include <stdio.h> %}
```

Rules: The rules are defined between `%%` and `%%`. These rules define the actions for when the token is scanned and are executed when a token matches the grammar.

Auxiliary Routines: Auxiliary routines contain the function required in the rules section. This Auxiliary section includes the `main()` function, where the `yyparse()` function is always called.

This `yyparse()` function plays the role of reading the token, performing actions and then returning to the `main()` after the execution or in the case of an error.

0 is returned after successful parsing and 1 is returned after an unsuccessful parsing.

### Q-3 Working of YACC?

Solution:

YACC in compiler design is set to work in C programming language along with its parser generator.

- An input with a `.y` extension is given.
- The file is invoked, and 2 files, `y.tab.h` and `y.tab.c`, are created. These files contain long codes implementing the LARL (1) Parser for grammar.
- This file then provides `yyparse.y`, which tries to parse a valid sentence successfully.

For the output files.

- If called with the `-d` option in the command line, YACC produces `y.tab.h` with all its specific definitions.
- If called with the `-v` option, YACC produces `y.output` having a textual description of the LALR(1) parsing table.

### Q-4 Example of YACC?

```
%{
#include <stdio.h>
#include <stdlib.h>

// Declare the type of the tokens
#define ID 258
#define NUM 259
#define NL 260

void yyerror(const char *s);
int yylex(void);
}%

%token NUM ID NL
%%
```

```
// Grammar rules
input:
    | input line
    ;

line:
    expr NL { printf("Result: %d\n", $1); }
    ;

expr:
    NUM      { $$ = $1; }
    | ID      { printf("Identifier: %s\n", $1); $$ = 0; } // For demonstration
    | expr '+' expr { $$ = $1 + $3; }
    | expr '-' expr { $$ = $1 - $3; }
    | '(' expr ')' { $$ = $2; }
    ;

%%

// Error handling function
void yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
}

int main() {
    printf("Enter expressions (Ctrl+D to exit):\n");
    return yyparse();
}
```

### Q-5 How to Execute file?

- Save the program code to a file with a .y extension, such as calculator.y
- Install Yacc/Bison on your system, if it is not already installed.
- Open a terminal or command prompt and navigate to the directory where the .y file is saved.
- Run the following command to generate a C source file from the Yacc/Bison grammar file  
Yacc -d calculator.y
- This will create two output files: y.tab.c (the C source file) and y.tab.h (the header file).
- Compile the generated C source file and link it with the Yacc/Bison runtime library using a C compiler such as GCC :  
Gcc -o calculator y.tab.c -ly

- Run the compiled program by typing its name and press enter :  
bash  
./calculator

**Program:****Lex file:**

```
%{
#include <stdlib.h>
#include <stdio.h>
#include "y.tab.h"
}%

%%

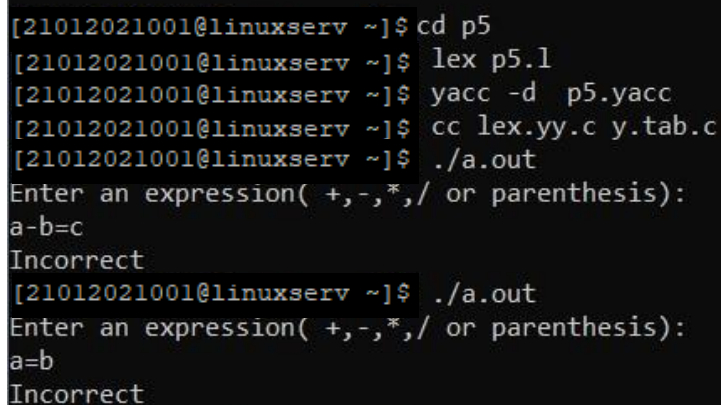
[0-9]+      { yylval = atoi(yytext); return NUM; }
[\t]        ; // Ignore whitespace
[a-zA-Z][a-zA-Z0-9]* { return ID; }
[\n]        { return NL; }
.           { return yytext[0]; }

%%

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

**Yacc file:**

Mentioned in Q-4

**Output:**

```
[21012021001@linuxserv ~]$ cd p5
[21012021001@linuxserv ~]$ lex p5.l
[21012021001@linuxserv ~]$ yacc -d p5.yacc
[21012021001@linuxserv ~]$ cc lex.yy.c y.tab.c
[21012021001@linuxserv ~]$ ./a.out
Enter an expression( +,-,*,/ or parenthesis):
a-b=c
Incorrect
[21012021001@linuxserv ~]$ ./a.out
Enter an expression( +,-,*,/ or parenthesis):
a=b
Incorrect
```