Practical:9

Aim: Introduction to YACC and generate calculator program.

YACC:

YACC (Yet Another Compiler Compiler) is a tool used for generating parsers, which are components of compilers that analyze the syntactic structure of a given input based on a formal grammar. It is often used in the process of creating programming languages or interpreters for languages. YACC is primarily used in conjunction with C, but the concepts can be applied to other languages as well.

Key Concepts of YACC:

Parser Generator: YACC takes a grammar specification (written in a specific syntax) and generates a parser in C code. This parser reads input and checks if it conforms to the rules specified in the grammar.

Grammar: The grammar defined in YACC is usually written in a form known as BNF (Backus-Naur Form) or a variation thereof. It consists of:

Terminals: The basic symbols from which strings (sentences) in the language are constructed. For example, keywords, operators, and identifiers.

Non-Terminals: Symbols that can be expanded into sequences of terminals and non-terminals according to the production rules.

Production Rules: Define how non-terminals can be replaced by sequences of terminals and/or other non-terminals.

Start Symbol: The initial symbol from which parsing starts.

Actions: Actions are C code snippets embedded in the grammar rules. They are executed when the corresponding rule is successfully matched.

Parse Table: YACC generates a parse table (often a set of tables for shift and reduce operations) which is used by the parser to decide what actions to take during parsing.

Basic Structure of a YACC File:

```
/* definitions */

....

%%

/* rules */

....

%%

/* auxiliary routines */

....
```

Code:

lex code:

```
 #include<stdio.h>
 #include "y.tab.h"
 #include<ctype.h>
 extern int yylval;
%%
[0-9]+ {
        yylval=atoi(yytext);
        return NUMBER;
      }
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
 return 1;
}
```

YACC code:

```
%{
#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;} ;
%%
void main()
{
printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Division, Modulus and Round brackets:\n");
yyparse();
if(flag==0)
printf("\nEntered arithmetic expression is Valid\n\n");
}
void yyerror()
{
printf("\nEntered arithmetic expression is Invalid\n\n");
flag=1;
}
```

Output:

```
ssasit@ssasit-Veriton-Series:~/Desktop$ cd 131
ssasit@ssasit-Veriton-Series:~/Desktop/131$ cd pr9
ssasit@ssasit-Veriton-Series:~/Desktop/131/pr9$ yacc -d pr9.y
ssasit@ssasit-Veriton-Series:~/Desktop/131/pr9$ lex pr9.l
ssasit@ssasit-Veriton-Series:~/Desktop/131/pr9$ gcc lex.yy.c y.tab.c -w
ssasit@ssasit-Veriton-Series:~/Desktop/131/pr9$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction,
 Multiplication, Division, Modulus and Round brackets:
6%56

Result=6

Entered arithmetic expression is Valid

ssasit@ssasit-Veriton-Series:~/Desktop/131/pr9$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction,
 Multiplication, Division, Modulus and Round brackets:
5*(4%456)+45

Result=65

Entered arithmetic expression is Valid

ssasit@ssasit-Veriton-Series:~/Desktop/131/pr9$
```

Practical:13

Aim: Implement a c program to implement operator precedence parsing.

Code:

```
#include<stdio.h>
#include<string.h>
void main() {
  char stack[20], ip[20], opt[10][10][1], ter[10];
  int i, j, k, n, top = 0, col, row;
  for (i = 0; i < 10; i++) {
    stack[i] = NULL;
    ip[i] = NULL;
    for (j = 0; j < 10; j++) {
      opt[i][j][1] = NULL;
    }
  }
  printf("Enter the no.of terminals :\n");
  scanf("%d", & n);
  printf("\nEnter the terminals :\n");
  scanf("%s", & ter);
  printf("\nEnter the table values :\n");
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
      printf("Enter the value for %c %c:", ter[i], ter[j]);
      scanf("%s", opt[i][j]);
    }
  }
  printf("\n**** OPERATOR PRECEDENCE TABLE ****\n");
  for (i = 0; i < n; i++) {
    printf("\t%c", ter[i]);
```

```
}
printf("\n");
for (i = 0; i < n; i++) {
  printf("\n%c", ter[i]);
  for (j = 0; j < n; j++) {
    printf("\t%c", opt[i][j][0]);
  }
}
stack[top] = '$';
printf("\nEnter the input string:");
scanf("%s", ip);
i = 0;
printf("\nSTACK\t\t\tINPUT STRING\t\t\tACTION\n");
printf("\n%s\t\t\t%s\t\t\t", stack, ip);
while (i <= strlen(ip)) {
  for (k = 0; k < n; k++) {
    if (stack[top] == ter[k])
      col = k;
    if (ip[i] == ter[k])
      row = k;
  }
  if ((stack[top] == '$') && (ip[i] == '$')) {
    printf("String is accepted\n");
    break;
  } else if ((opt[col][row][0] == '<') || (opt[col][row][0] == '=')) {
    stack[++top] = opt[col][row][0];
    stack[++top] = ip[i];
    printf("Shift %c", ip[i]);
    i++;
  } else {
    if (opt[col][row][0] == '>') {
```

```
        while (stack[top] != '<') {
          --top;
        }
        top = top - 1;
        printf("Reduce");
      } else {
        printf("\nString is not accepted");
        break;
      }
    }
    printf("\n");
    for (k = 0; k <= top; k++) {
      printf("%c", stack[k]);
    }
    printf("\t\t\t");
    for (k = i; k < strlen(ip); k++) {
      printf("%c", ip[k]);
    }
    printf("\t\t\t");
  }
}
```

Output:

```
ssasit@ssasit-Veriton-Series:~/Desktop/131$ ./a.out
Enter the no.of terminals :
4

Enter the terminals :
+*i$

Enter the table values :
Enter the value for + +:>
Enter the value for + *:<
Enter the value for + i:<
Enter the value for + $:>
Enter the value for * +:>
Enter the value for * *:>
Enter the value for * i:<
Enter the value for * $:>
Enter the value for i +:>
Enter the value for i *:>
Enter the value for i i:=
Enter the value for i $:>
Enter the value for $ +:<
Enter the value for $ *:<
Enter the value for $ i:<
Enter the value for $ $:A

**** OPERATOR PRECEDENCE TABLE ****
        +       *       i       $

+       >       <       <       >
*       >       >       <       >
i       >       >       =       >
$       <       <       <       A
Enter the input string:i+i*i+i$

STACK                   INPUT STRING                    ACTION

$                       i+i*i+i$                        Shift i
$<i                     +i*i+i$                 Reduce
$                       +i*i+i$                 Shift +
$<+                     i*i+i$                  Shift i
$<+<i                   *i+i$                   Reduce
$<+                     *i+i$                   Shift *
$<+<*                   i+i$                    Shift i
$<+<*<i                 +i$                     Reduce
$<+<*                   +i$                     Reduce
$<+                     +i$                     Reduce
$                       +i$                     Shift +
$<+                     i$                      Shift i
$<+<i                   $                       Reduce
$<+                     $                       Reduce
$                       $                       String is accepted
ssasit@ssasit-Veriton-Series:~/Desktop/131$
```