// NAME: AHAN BANDYOPADHYAY

//ROLL No. 211210008

//Compiler Design Lab 7-8

1. Consider the example of simple desk calculator that performs simple operations on integer expressions with the grammar:

exp -> exp addop term | term addop -> + | -

term -> term mulop factor | factor mulop -> *

factor -> (exp) | number

number -> number digit | digit

digit -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

You are required to write YACC specifications for this grammar so that the parser evaluates any arithmetic expressions and the output shows each grammar rule as it is applied in the parsing process. Show your parsing sequence for the input string: (2+(3*4))

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

%token NUMBER
%token PLUS MINUS TIMES LPAREN RPAREN
%left PLUS MINUS
%left TIMES
%start exp

%%

exp : exp PLUS term  { printf("exp -> exp + term\n"); $$ = $1 + $3; }
   | exp MINUS term { printf("exp -> exp - term\n"); $$ = $1 - $3; }
   | term           { printf("exp -> term\n"); $$ = $1; }
   ;

term : term TIMES factor { printf("term -> term * factor\n"); $$ = $1 * $3; }
    | factor          { printf("term -> factor\n"); $$ = $1; }
    ;

factor : LPAREN exp RPAREN { printf("factor -> (exp)\n"); $$ = $2; }
      | NUMBER            { printf("factor -> number\n"); $$ = $1; }
      ;

NUMBER: DIGIT+
DIGIT: '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
PLUS: '+'
MINUS: '-'
TIMES: '*'
LPAREN: '('
```

```
RPAREN: ')'

%%

int yylex() {
    int c = getchar();
    if (isdigit(c)) {
        yylval = c - '0';
        return NUMBER;
    } else if (c == '+') {
        return PLUS;
    } else if (c == '-') {
        return MINUS;
    } else if (c == '*') {
        return TIMES;
    } else if (c == '(') {
        return LPAREN;
    } else if (c == ')') {
        return RPAREN;
    } else if (c == EOF) {
        return 0;
    } else {
        return -1; // Error
    }
}

int main() {
    yyparse();
    return 0;
}
```

OUTPUT:
Input: (2+(3*4))

```
factor -> (exp)
exp -> term
term -> term * factor
exp -> exp + term
exp -> term
term -> factor
factor -> number
factor -> number
factor -> number
```

2. The following grammar describes a boolean expression (exp) consisting of operators "&", "|", "!", "==", "! =", brackets "(" and ")", and identifiers.

exp : exp_2 | exp '&' exp_2 ;

exp_2 : exp_3 | exp_3 '|' exp_2 ;

exp_3 : exp_4 | exp_4 '==' exp_4 | exp_4 '!=' exp_4 ; exp_4 : exp_5 | '!' exp_5 ;

exp_5 : identifier | '(' exp ')' ;

Write Yacc code to recognize a series of expressions, each on a new line. Each expression should unambiguously demonstrate precedence & associativity of the operators, by showing the orders in which operators would be evaluated. Also, give a parse sequence for each of the expressions.

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
%}

%token IDENTIFIER
%token AND OR NOT EQUAL NOTEQUAL LPAREN RPAREN
%left AND
%left OR
%left EQUAL NOTEQUAL
%right NOT

%start exp_list

%%
exp_list : exp '\n' { printf("\n"); }
        | exp_list exp '\n' { printf("\n"); };

exp : exp AND exp_2 { printf("exp -> exp & exp_2\n"); }
   | exp_2 { printf("exp -> exp_2\n"); } ;

exp_2 : exp_3 OR exp_2 { printf("exp_2 -> exp_3 | exp_2\n"); }
     | exp_3 { printf("exp_2 -> exp_3\n"); } ;

exp_3 : exp_4 EQUAL exp_4 { printf("exp_3 -> exp_4 == exp_4\n"); }
     | exp_4 NOTEQUAL exp_4 { printf("exp_3 -> exp_4 != exp_4\n"); }
     | exp_4 { printf("exp_3 -> exp_4\n"); };

exp_4 : NOT exp_5 { printf("exp_4 -> ! exp_5\n"); }
     | exp_5 { printf("exp_4 -> exp_5\n"); };

exp_5 : IDENTIFIER { printf("exp_5 -> identifier\n"); }
     | LPAREN exp RPAREN { printf("exp_5 -> ( exp )\n"); };
```

```
IDENTIFIER : [a-zA-Z_][a-zA-Z0-9_]*
AND : '&'
OR : '|'
NOT : '!'
EQUAL : "=="
NOTEQUAL : "!="
LPAREN : '('
RPAREN : ')'

%%
int yylex() {
    int c = getchar();
    if (c == '&') return AND;
    if (c == '|') return OR;
    if (c == '!') return NOT;
    if (c == '=') {
        if ((c = getchar()) == '=') return EQUAL;
        ungetc(c, stdin);
        return '=';
    }
    if (c == '!') {
        if ((c = getchar()) == '=') return NOTEQUAL;
        ungetc(c, stdin);
        return '!';
    }
    if (c == '(') return LPAREN;
    if (c == ')') return RPAREN;
    if (isalpha(c) || c == '_') {
        char buffer[100];
        int i = 0;
        buffer[i++] = c;
        while ((c = getchar()) != EOF && (isalnum(c) || c == '_')) {
            buffer[i++] = c;
        }
        buffer[i] = '\0';
        ungetc(c, stdin);
        yylval = strdup(buffer);
        return IDENTIFIER;
    }
    if (c == '\n' || c == EOF) return 0; // End of input
    return -1; // Error
}
```

```
int main() {
   yyparse();
   return 0;
}
```

OUTPUT

1. Expression: "a & b | c == d"

```
exp -> exp & exp_2
exp -> exp_2
exp_2 -> exp_3 | exp_2
exp_2 -> exp_3
exp_3 -> exp_4 == exp_4
exp_3 -> exp_4
exp_4 -> exp_5
exp_5 -> identifier
exp_5 -> identifier
exp_5 -> identifier
exp_5 -> identifier
```

2. Expression: "!a | b != c & d"

```
exp -> exp_2
exp_2 -> exp_3 | exp_2
exp_2 -> exp_3
exp_3 -> exp_4 != exp_4
exp_3 -> exp_4
exp_4 -> exp_5
exp_4 -> ! exp_5
exp_5 -> identifier
exp_5 -> identifier
exp_5 -> identifier
exp_5 -> identifier
```

3. Expression: "a != b | !c & d == e"

```
exp -> exp_2
exp_2 -> exp_3 | exp_2
exp_2 -> exp_3
exp_3 -> exp_4 == exp_4
exp_3 -> exp_4
exp_4 -> ! exp_5
exp_4 -> exp_5
exp_5 -> identifier
exp_5 -> identifier
exp_5 -> identifier
exp_5 -> identifier
exp_5 -> identifier
```

3. For the grammar below, write a YACC program to compute the decimal value of an input string in binary. For example the translation of the string 101.101 should be the decimal number 5.625.

S→L.L|L L→LB|B B→0|1

```
%{
#include <stdio.h>
#include <math.h>
%}

%token ZERO ONE DOT
%start S

%%
S : L DOT L { printf("Decimal value: %f\n", $1 + $3); };;

L : L B { $$ = $1 * 2 + $2; }
  | B { $$ = $1; };

B : ZERO { $$ = 0; }
  | ONE { $$ = 1; };

ZERO : '0';
ONE : '1';
DOT : '.';
%%

int main() {
    yyparse();
    return 0;
}
int yyerror(const char *s) {
    printf("Error: %s\n", s);
    return 0;
```

```
}

int yylex() {
    int c = getchar();
    if (c == '0')
        return ZERO;
    else if (c == '1')
        return ONE;
    else if (c == '.')
        return DOT;
    else if (c == '\n' || c == EOF)
        return 0;
    else
        yyerror("Invalid character");
}
```

Input: 101.101

Output:

L -> L B (1st digit: 1)

L -> L B (2nd digit: 10)

L -> L B (3rd digit: 101)

L -> L . L (Dot encountered)

L -> L . L B (1st digit after dot: 1)

L -> L . L B (2nd digit after dot: 10)

L -> L . L B (3rd digit after dot: 101)

Decimal value: 5.625