Practical No. 8:

* Title: Desk calculator ~~and related~~ with error recovery.

* Objective: Students will learn and implement
  i) Parser for desk calculator and related error recovery
  ii) Rules section for such a scanner and a parser and their working in synchronization.

* Description:

i) Create a LEX file first & RE for the digit

ii) In the rules section, write rules for number identification, newline and other details apart from this.

iii) Then create a YACC file. In declaration part of the YACC file include header file like stdio.h, ctype.h and define. YYSTYPE as double.

iv) Declare ID as a number, and precedence for token related arithmetic operations like +, -, * and / etc.

v) In the rules section of YACC file, add rules for start symbol which will print the value of the answer, write rules for catching the error like if the expression is entered is wrong then ask user to enter it once more.

vi) Also write rules for different arithmetic operations like +, -, * and /, unary - and for number also in the rules section of the YACC file.

vii) Then include lex.yy.c file which acts as a connector b/w LEX and YACC files. Lastly, call main routine for printing the value of the expression through yyparse().

Code for LEX:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h" // Include generated header from yacc
%}

%%
[0-9]+(\.[0-9]+)?         { yylval.dval = atof(yytext); return NUMBER; }
[ \t]+                    ; // Ignore whitespace
\n                        { return '\n'; } // Return newline as a token
"("                       { return LPAREN; }
")"                       { return RAREN; } // Ensure this matches your
token name
"+"                       { return PLUS; }
"-"                       { return MINUS; }
"*"                       { return MULTIPLY; }
"/"                       { return DIVIDE; }
.                         { return yytext[0]; } // Any other character

%%

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

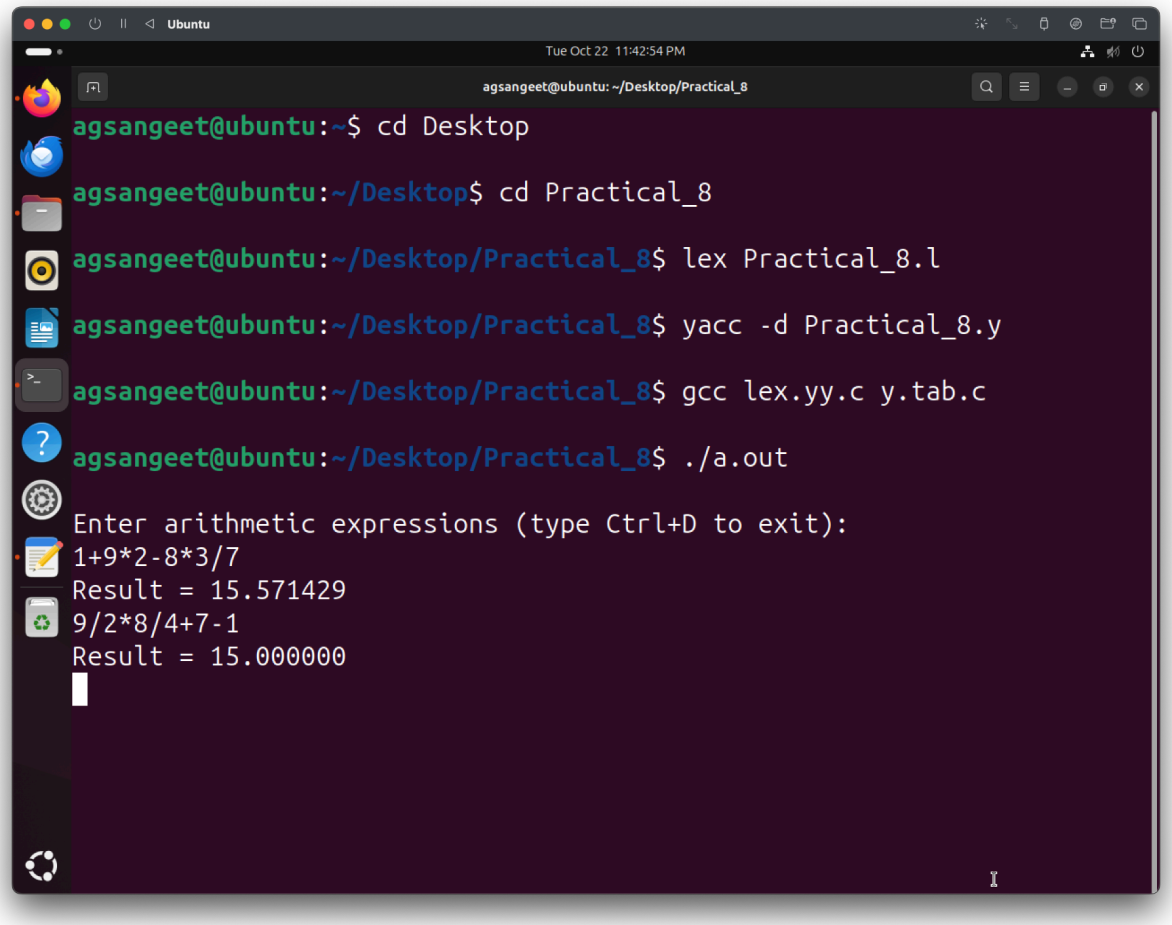
Code for YACC:

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

void yyerror(const char *s); // Declare the error handling function
int yylex(void); // Declare lexer function

%}

%union {
    double dval; // Store double values from expressions
```

```
}

%token <dval> NUMBER
%token LPAREN RAREN PLUS MINUS MULTIPLY DIVIDE

%left PLUS MINUS
%left MULTIPLY DIVIDE
%nonassoc UMINUS // Unary minus precedence
%nonassoc ERROR // Error recovery

%type <dval> expression

%%

// Grammar rules
calculation:
    /* empty */
    | calculation expression '\n' { printf("Result = %lf\n", $2); }
    ;

expression:
    NUMBER { $$ = $1; }
    | expression PLUS expression { $$ = $1 + $3; }
    | expression MINUS expression { $$ = $1 - $3; }
    | expression MULTIPLY expression { $$ = $1 * $3; }
    | expression DIVIDE expression {
        if ($3 == 0) {
            yyerror("Error: Divide by zero");
            YYERROR; // Trigger a syntax error
        } else {
            $$ = $1 / $3;
        }
    }
    | LPAREN expression RAREN { $$ = $2; }
    | MINUS expression %prec UMINUS { $$ = -$2; }
    ;

%%

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

```
}

int main(void) {
    printf("Enter arithmetic expressions (type Ctrl+D to exit):\n");
    return yyparse(); // Start the parser
}
```

Output:



* Conclusion: Thus, we have implemented the parser for desk calculator.