**TASK 1:**

**recur.l:**

%{

#include "recur.tab.h"

%}

%%

[ \t\n]      ;

[0-9]+       { yylval = atoi(yytext); return NUMBER; }

"+"          { return '+'; }

"-"          { return '-'; }

"\*"          { return '\*'; }

"/"          { return '/'; }

"("          { return '('; }

")"          { return ')'; }

.            ;

%%

int yywrap() {

    return 1;

}

**recur.y:**

%{

#include <stdio.h>

extern int yylex();

extern int yylval;

float result;

void yyerror(const char \*s);

%}

%token NUMBER

%%

input: expression {printf("Result: %d\n",$1);}

expression: term

          | expression '+' term { $$ = $1 + $3; }

          | expression '-' term { $$ = $1 - $3; }

          ;

term: factor

    | term '\*' factor { $$ = $1 \* $3; }

    | term '/' factor { $$ = $1 / $3; }

    ;

factor: NUMBER         { $$ = $1; }

      | '(' expression ')' { $$ = $2; }

      ;

%%

int main() {

    printf("\nEnter Expression: ");

    yyparse();

    return 0;

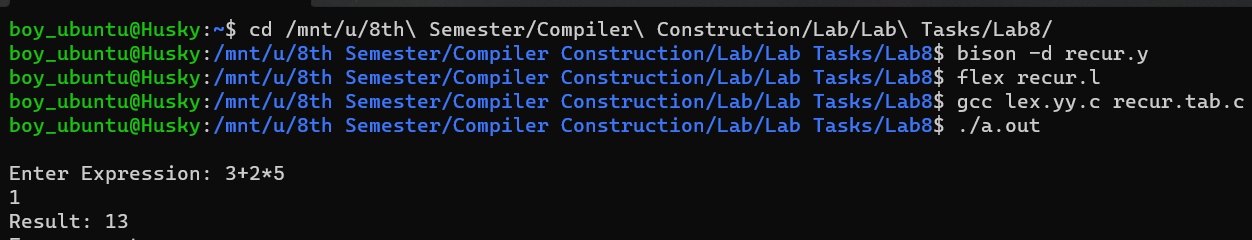
}

void yyerror(const char \*s) {

    fprintf(stderr, "Error: %s\n", s);

}

**OUTPUT:**



**TASK 2:**

**recur.c:**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

int evaluate\_expression(const char \*expression);

int evaluate\_term(const char \*\*expr);

int evaluate\_factor(const char \*\*expr);

int main() {

    char expression[100];

    printf("Enter an arithmetic expression: ");

    fgets(expression, sizeof(expression), stdin);

    int result = evaluate\_expression(expression);

    printf("Result: %d\n", result);

    return 0;

}

int evaluate\_expression(const char \*expression) {

    int result = evaluate\_term(&expression);

    while (\*expression == '+' || \*expression == '-') {

        char op = \*expression++;

        int right = evaluate\_term(&expression);

        if (op == '+') {

            result += right;

        } else {

            result -= right;

        }

    }

    return result;

}

int evaluate\_term(const char \*\*expr) {

    int result = evaluate\_factor(expr);

    while (\*\*expr == '\*' || \*\*expr == '/') {

        char op = \*(\*expr)++;

        int right = evaluate\_factor(expr);

        if (op == '\*') {

            result \*= right;

        } else {

            result /= right;

        }

    }

    return result;

}

int evaluate\_factor(const char \*\*expr) {

    if (\*\*expr == '(') {

        (\*expr)++;

        int result = evaluate\_expression(\*expr);

        while (\*\*expr != ')') {

            (\*expr)++;

        }

        (\*expr)++;

        return result;

    } else {

        int result = 0;

        while (isdigit(\*\*expr)) {

            result = result \* 10 + (\*\*expr - '0');

            (\*expr)++;

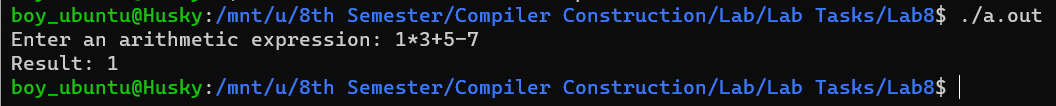
        }

        return result;

    }

}

**OUTPUT:**

****