
# NOTRE DAME UNIVERSITY

# BANGLADESH

# CSE-3104 LAB Report-03

**Course Title: Compiler Design Lab**

**Course Code: CSE-3104**

## Submitted by:

**Name: Istiak Alam**

**ID: 0692230005101005**

**Batch: CSE-20**

**Submission Date: 24-11-24**

**Lab Task Topic: Compiler Problem Solving using Lex**

## Submitted to:

**Khorshed Alam**
**Lecturer, NDUB**

## Problem:

Write a YACC program to implement Grammar for a simple calculator.

## Solution:

## Code : –

```
// Lex File -→ Calc.l //
%{
#include "y.tab.h"  // Include Bison-generated header for token definitions
#include <stdlib.h>
extern int yylex(void);  // Declare yylex(), generated by Flex
extern int yyerror(const char *s);

%}

%%

[ \t]+              { /* Ignore whitespace */ }
[0-9]+(\.[0-9]+)?([eE][+-]?[0-9]+)?  { yylval = atof(yytext); return NUMBER; }  // Match numbers, both
integer and floating-point

"+"                   { return PLUS; }
"-"                   { return MINUS; }
"*"                   { return TIMES; }
"/"                   { return DIVIDE; }
"^"                   { return POWER; }
"("                   { return LEFT; }
")"                   { return RIGHT; }
\n                    { return END; }

.                     { return 0; }  // Catch any unrecognized characters

%%

int yywrap(void) {
    return 1;  // End of input
}
```

```
// Yacc File → Calc.y //
%{
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "y.tab.h"  // Include Flex-generated header for token definitions

#define YYSTYPE int
%}

%token NUMBER
%token PLUS MINUS TIMES DIVIDE POWER
%token LEFT RIGHT
%token END

%left PLUS MINUS
%left TIMES DIVIDE
%left NEG
%right POWER

%start Input

%%
```

```
Input:
    | Input Line
    ;

Line:
    END
    | Expression END { printf("Result: %d\n", $1); }
    ;

Expression:
    NUMBER { $$ = $1; }
    | Expression PLUS Expression { $$ = $1 + $3; }
    | Expression MINUS Expression { $$ = $1 - $3; }
    | Expression TIMES Expression { $$ = $1 * $3; }
    | Expression DIVIDE Expression { $$ = $1 / $3; }
    | MINUS Expression %prec NEG { $$ = -$2; }
    | Expression POWER Expression { $$ = pow($1, $3); }
    | LEFT Expression RIGHT { $$ = $2; }
    ;

%%

int yyerror(const char *s) {
     fprintf(stderr, "%s\n", s);
    return 0;
}

int main() {
    printf("Enter an expression:\n");
    if (yyparse()) {
        fprintf(stderr, "Parsing failed.\n");
    } else {
        fprintf(stderr, "Parsing successful.\n");
    }
    return 0;
}
```

## Code Processing –

Step 1 : Open terminal in Linux and create a lex file named **calc.l**

⇒ **`touch calc.l`**

Step 1 : Open terminal in Linux and create a yacc file named **calc.y**

⇒ **`touch calc.y`**

Step 2 : After Writing those code save it and type command in terminal :

⇒ **`lex calc.l`**

⇒ **`bison -d cal.y`**

Step 3 : It will create a lex.yy.c & yyfile. Then next type command in terminal

⇒ **`gcc -o calculator lex.yy.c cal.tab.c -lfl -lm`**

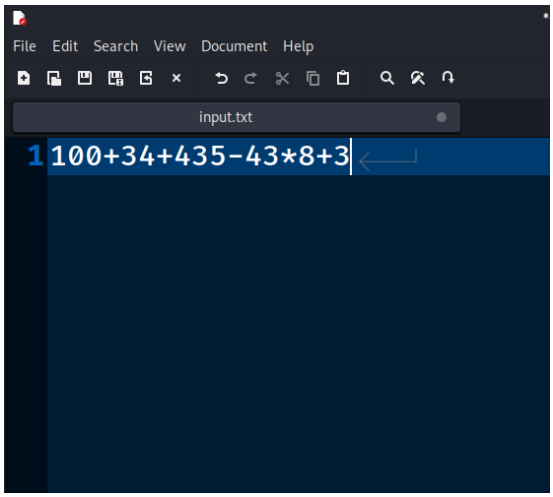
Step 4 : Then it will create an executable file named **calculator.exe**.

Step 5 : Next type command in terminal and run the exe file :

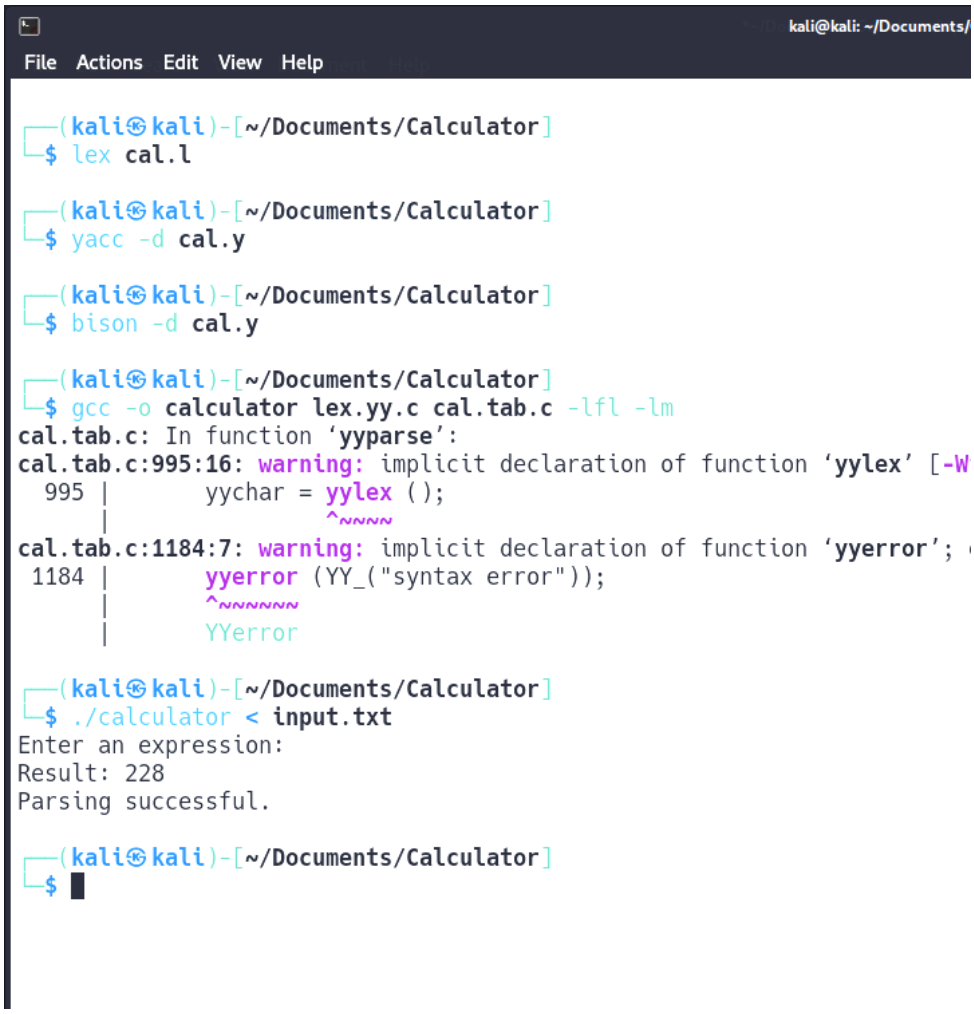
⇒ **`./calculator < input.txt`**

## Input and Output:

1. We are using **input.txt** file for checking out inputs in this lexical analyzer.
2. In the **input.txt** file there are a simple program language –




3. After injecting the input, the output is –




*Output*