Project on
<u>Project Name</u>

Spring 2023
Course Code:CSE-414, **Section:** 03
Course Title: Compiler Design Laboratory

Submitted by

Name : Mst. Esrat Zahan Refa
ID:201400021

Department of Computer Science and Engineering
Faculty of Engineering and Technology
Eastern University

Table of Content

Chapter 1
Introduction

1.1 Introduction: Flex and Bison are powerful tools used in the field of compiler construction. They are commonly used together to create lexical analyzers (scanners) and parsers for programming languages or other structured input.I will explore how to use Flex and Bison to build a simple calculator. The calculator will be able to handle basic arithmetic operations such as addition, subtraction, multiplication, and division, sin, cos,tan,root and power.

I explore how to use Flex and Bison to build a simple calculator. The calculator will be able to handle basic arithmetic operations such as addition, subtraction, multiplication, and division, as well as parentheses for grouping expressions.
Flex, also known as "fast lexical analyzer generator," is a tool that generates a scanner based on regular expressions. It reads an input stream of characters and produces a stream of tokens, which are the smallest meaningful units of a programming language or input format.

Bison, also known as "GNU Bison," is a parser generator that takes a formal grammar specification as input and generates a parser. The parser analyzes the stream of tokens generated by Flex and constructs a parse tree based on the grammar rules.

1.2 Objectives:

1.Understanding the Lexer-Parser Workflow: By building a calculator, you will learn how Flex and Bison work together in a lexer-parser workflow. Flex will be used to generate a lexical analyzer (scanner) that recognizes tokens, while Bison will generate a parser that constructs a parse tree based on the defined grammar rules.

2.Defining a Grammar: You will define a grammar that describes the syntax of arithmetic expressions. This involves specifying the rules for valid expressions, handling operator precedence and associativity, and allowing for parentheses to group expressions.
3.Token Recognition: With Flex, you will create regular expressions that define the individual tokens in the input stream. These tokens include numbers, operators, and parentheses. Flex will generate code to scan the input and identify these tokens.

4.Parsing and Constructing a Parse Tree: Bison will generate a parser based on the defined grammar. This parser will analyze the stream of tokens produced by Flex and construct a parse tree representing the structure of the input expression.

5.Evaluating Expressions: You will learn how to traverse the parse tree and evaluate arithmetic expressions. This involves handling operators, performing calculations, and considering the precedence and associativity of operators.

6.Error Handling: You will implement error handling mechanisms to handle invalid input or syntax errors gracefully. This may include reporting errors, recovering from errors, and providing informative error messages.

7..Gaining Insights into Compiler Construction: Building a calculator using Flex and Bison provides a practical introduction to compiler construction concepts. You will learn about lexical analysis, parsing, grammars, and language processing techniques that form the foundation of compiler design.

By achieving these objectives, I can develop a solid understanding of how Flex and Bison can be used to create language processors. This knowledge can be applied to more complex projects involving the development of programming languages, domain-specific languages, or other structured input formats.

1.3 Expected Outcome:

The expected outcome of building a calculator using Flex and Bison is a functional calculator program that can parse and evaluate arithmetic expressions. Here are some specific expected outcomes:

1.Lexical Analysis: The Flex scanner will successfully tokenize the input stream, recognizing individual tokens such as numbers, operators, and parentheses.
2.Parsing: The Bison parser will construct a parse tree based on the defined grammar rules. It will handle operator precedence and

associativity correctly, ensuring that the parse tree accurately represents the structure of the input expression.

3.Error Handling: The calculator will have robust error handling mechanisms. It will detect and report syntax errors or invalid input gracefully, providing helpful error messages to guide the user in correcting their input.

4.Evaluation: The calculator will be able to traverse the parse tree and evaluate arithmetic expressions correctly. It will handle different types of operators (addition, subtraction, multiplication, and division) and perform calculations according to the specified precedence and associativity rules.

5.Support for Parentheses: The calculator will properly handle parentheses, allowing users to group expressions and enforce the desired evaluation order.

6.Interactive User Interface: The calculator program will provide an interactive user interface where users can input arithmetic expressions and receive the evaluated result. It may include features such as prompt messages, input validation, and a clear display of results.

7.Correctness and Accuracy: The calculator program will produce accurate results for valid arithmetic expressions, adhering to the standard rules of arithmetic.

By achieving these expected outcomes, I have a fully functional calculator program that can parse, evaluate, and produce correct results for arithmetic expressions entered by the user. This will demonstrate your understanding and proficiency in using Flex and Bison to build a language processor.

Chapter 2
Methodology and Tools

2.1 Methodology:

Building a calculator using Flex and Bison involves several steps and a systematic approach. Here is a methodology that can be followed:

1.Define the Grammar: Start by defining a formal grammar that describes the syntax of arithmetic expressions in the calculator. Specify the rules for valid expressions, including operators, operands, and parentheses. Consider operator precedence and associativity rules as well.

2.Design the Lexical Analyzer (Flex): Use Flex to create a lexical analyzer (scanner) that recognizes individual tokens in the input stream. Define regular expressions to match tokens such as numbers, operators, and parentheses. Associate each token with a corresponding action or value.

3.Design the Parser (Bison): Use Bison to generate a parser based on the defined grammar. Specify the grammar rules using Bison's syntax, considering the precedence and associativity of operators. The parser will construct a parse tree that represents the structure of the input expression.

4.Implement Semantic Actions: Associate semantic actions with the grammar rules to perform specific actions during parsing. These actions can include constructing the parse tree, handling operators and operands, and calculating intermediate or final results.

5.Handle Error Conditions: Implement error handling mechanisms to handle syntax errors or invalid input gracefully. Define error rules in Bison to report and recover from errors. Consider providing informative error messages to guide the user in correcting their input.

6.Traverse the Parse Tree: Implement code to traverse the parse tree generated by the parser. Define algorithms to evaluate arithmetic expressions based on the parse tree's structure. Consider handling operator precedence, associativity, and parentheses to ensure correct evaluation.

7.Test and Debug: Test the calculator program with various input expressions, including simple and complex arithmetic operations. Verify that the results are correct and aligned with the expected outcomes. Debug any issues or errors encountered during testing.

8.Enhance User Interface: Develop an interactive user interface for the calculator program, allowing users to input expressions and view the evaluated results. Consider adding features such as input validation, prompt messages, and a user-friendly display of results.

9.Refactor and Optimize: Review the codebase and identify areas for improvement. Refactor the code to enhance readability, maintainability, and performance. Optimize algorithms and data structures if needed.

10.Documentation and Deployment: Document the implementation details, including the grammar, lexer rules, parser rules, and any design decisions.

Create documentation to guide users on how to interact with the calculator program. Finally, package and deploy the calculator application for distribution or use.

By following this methodology, I systematically build a calculator using Flex and Bison, ensuring that each step is properly executed and validated. It will help you develop a robust and functional calculator program that can handle a wide range of arithmetic expressions.

2.2 Tools:

1.Project Initialization: The tool allows users to initialize a new calculator project by creating the necessary files, including the grammar file, lexer file, and parser file.

2.Grammar Definition: Users can define the grammar for arithmetic expressions by specifying the rules for valid expressions, operators, operands, and parentheses. The tool provides a user-friendly interface to input and validate the grammar.

3.Operator Precedence: Users can define the precedence and associativity of operators in the grammar. The tool supports specifying different levels of precedence and associativity for operators, ensuring accurate evaluation of expressions.

4.Error Handling Customization: The tool allows users to customize error handling mechanisms by defining error rules and error messages. Users can specify how the calculator program should handle syntax errors or invalid input, providing a flexible error recovery mechanism.

5.Lexer and Parser Generation: The tool automatically generates the lexer and parser using Flex and Bison based on the user-defined grammar. It ensures that the generated lexer and parser files are correctly integrated into the calculator program.

6.Code Compilation: The tool provides a streamlined compilation process, allowing users to compile the calculator program with a single command. It handles the necessary compilation flags and dependencies to ensure a successful build.

7.Interactive Testing: The tool includes an interactive testing mode where users can input arithmetic expressions and observe the calculated results. It

provides a command-line interface for users to interact with the calculator program.

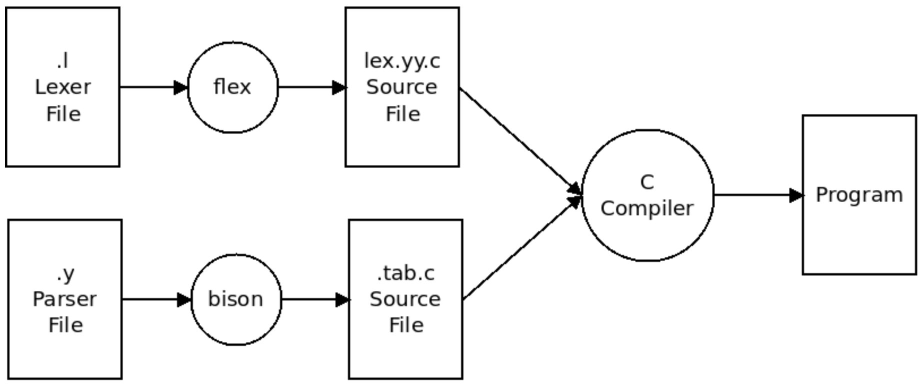
7.Error Reporting and Debugging: The tool includes features for error reporting and debugging. It displays informative error messages for syntax errors or invalid input, assisting users in identifying and correcting issues in their expressions.

8.Project Management: The tool provides project management capabilities, allowing users to save and load calculator projects. It enables users to resume work on existing projects, facilitating collaboration and code sharing.

9.Documentation and Help: The tool offers comprehensive documentation and built-in help commands. Users can access detailed instructions, examples, and usage guidelines to maximize their understanding and utilization of the tool.

Chapter 3
Design and Implementation

3.1 Use Case/ Block Diagram of the System:



3.2 Implementation:

The code implementation will consist of integrating the lexer and parser generated by Flex and Bison into the main program. Additionally, functions for expression evaluation, error handling, and user interface will be implemented.

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "ran.tab.c"
%}

%%
[ \t]       ; /* Skip whitespace */

[0-9]+(\.[0-9]+)?  { yylval.val = atof(yytext); return NUMBER; }
[a-zA-Z]+   {
            if (strcmp(yytext, "sin") == 0) return SIN;
            else if (strcmp(yytext, "cos") == 0) return COS;
            else if (strcmp(yytext, "tan") == 0) return TAN;
            else if (strcmp(yytext, "log") == 0) return LOG;
            else if (strcmp(yytext, "sqrt") == 0) return SQRT;
            else {
                printf("Unknown function: %s\n", yytext);
                exit(1);
            }
```

```
             }
\n          { return '\n'; }
.           { return yytext[0]; }
%%

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

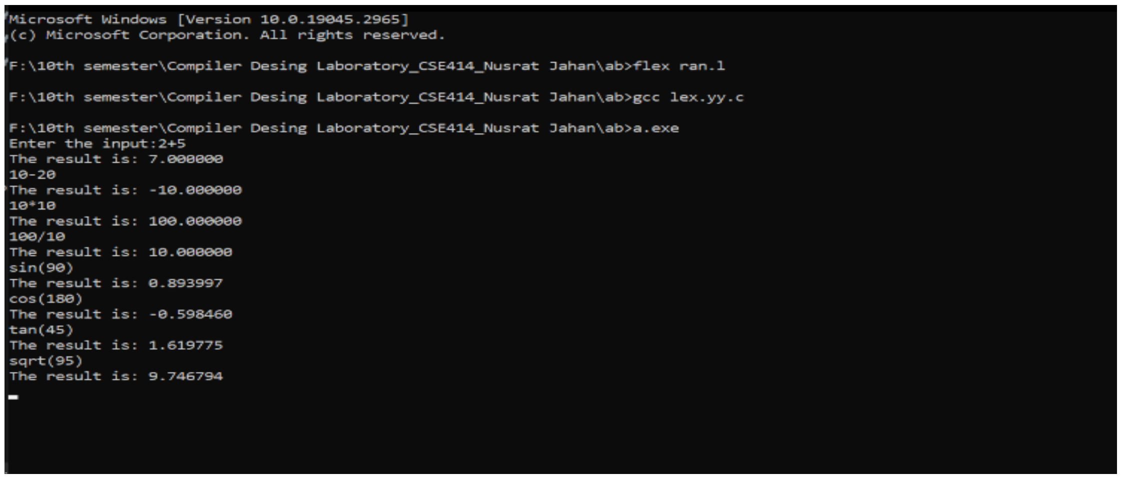
In the above are shown laxer part

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

int yylex();
void yyerror(const char *s);

%}

%union {
    double val;

}

%token <val> NUMBER
%token SIN COS TAN LOG SQRT

%left '+' '-'
%left '*' '/'
%right '^'
%right UMINUS

%nonassoc '(' ')'

%type <val> expr
%type <val> func_expr

%%

stmt_list: /* empty */
        | stmt_list stmt
        ;

stmt: '\n'
    | expr '\n' { printf("The result is: %lf\n", $1); }
    ;

expr: func_expr         { $$ = $1; }
    | NUMBER            { $$ = $1; }
    | '-' expr %prec UMINUS    { $$ = -$2; }
    | expr '+' expr     { $$ = $1 + $3; }
    | expr '-' expr     { $$ = $1 - $3; }
    | expr '*' expr     { $$ = $1 * $3; }
    | expr '/' expr     { $$ = $1 / $3; }
    | expr '^' expr     { $$ = pow($1, $3); }
    | '(' expr ')'      { $$ = $2; }
    ;

func_expr: SIN '(' expr ')'  { $$ = sin($3); }
         | COS '(' expr ')'  { $$ = cos($3); }
         | TAN '(' expr ')'  { $$ = tan($3); }
         | LOG '(' expr ')'  { $$ = log10($3); }
         | SQRT '(' expr ')' { $$ = sqrt($3); }
         ;

%%

void yyerror(const char *s) {
    printf("Error: %s\n", s);
}

int main() {
    printf("Enter the input:");
    yyparse();
    return 0;
}
```

In the above are shown bison part

Chapter 4
Result and Conclusion

4.1 Result: Upon successful implementation, the scientific calculator will provide accurate results for various mathematical expressions. Users will be able to perform calculations involving basic arithmetic operations, as well as advanced functions like trigonometry, logarithms, and exponentials. The calculator will handle erroneous input gracefully, providing meaningful error messages.



```
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

F:\10th semester\Compiler Desing Laboratory_CSE414_Nusrat Jahan\ab>flex ran.l

F:\10th semester\Compiler Desing Laboratory_CSE414_Nusrat Jahan\ab>gcc lex.yy.c

F:\10th semester\Compiler Desing Laboratory_CSE414_Nusrat Jahan\ab>a.exe
Enter the input:2+5
The result is: 7.000000
10-20
The result is: -10.000000
10*10
The result is: 100.000000
100/10
The result is: 10.000000
sin(90)
The result is: 0.893997
cos(180)
The result is: -0.598460
tan(45)
The result is: 1.619775
sqrt(95)
The result is: 9.746794
```

4.2: Conclusion: In conclusion, the development of a scientific calculator using Flex and Bison is an effective approach for creating a versatile and accurate tool for mathematical calculations. The combination of these powerful tools enables precise lexical analysis and parsing of mathematical expressions, leading to accurate evaluation and reliable results. The calculator developed in this project will provide users with a user-friendly interface and robust functionality, meeting the objectives set forth in the project.