**Khulna University of Engineering & Technology**

**Department of Computer Science & Engineering**

**Course No :** CSE 3112

**Course Title :** Compiler Design Laboratory

**Project Name** : Simple Compiler using Bison

**Submission By**

Faisal Ahmed

Roll : 1607048

Section : A

**Objectives**

1.To know about the compiler

2.To know the translation of a high level language into a low level language

3. To know the top down parser and the bottom up parser .

4. To know the Flex and Bison for implementation of a compiler using C programming language .

5.To create a new language and it’s semantic and syntactic rules .

6.To check some different type of input and their output of the compiler .

7.To implement the Regular Expression ,Context Free Grammar in the compiler .

**Introduction**

A compiler is a computer program that translates computer code written in one programming language into another language. The name compiler is primarily used for programs that translate source code from a high-level programming language to a lower level language to create an executable program.

**Flex and Bison**

Lex is a program that generates lexical analyzer. It is used with YACC parser generator. The lexical analyzer is a program that transforms an input stream into a sequence of tokens. .It reads the input stream and produces the source code as output through implementing the lexical analyzer in the C program

Bison is a general-purpose parser generator that converts a grammar description (Bison Grammar Files) for an LALR(1) context-free grammar into a C program to parse that grammar. The Bison parser is a bottom-up parser. ... Compile the code output by Bison, as well as any other source files .

**Run The program in terminal**

1. bison -d main.y
2. flex main.l
3. gcc lex.yy.c main.tab.c -o app
4. app

Procedure

1.The code is divided into two part flex file (.l) and bison file (.y) .

2.Input expression check the lex (.y) file and if the expression satisfies the rule then it check the CFG into the bison file .

3.it’s a bottom up parser and the parser construct the parse tree .firstly ,matches the leaves node with the rules and if the CFG matches then it gradually goes to the root .

Token

A **token** is the smallest element(character) of a computer language program that is meaningful to the **compiler**. The parser has to recognize these as **tokens**: identifiers, keywords, literals, operators, punctuators, and other separators .

My compiler tokens

NUM, VAR ,IF, ELSE, ARRAY ,MAIN ,INT, FLOAT ,CHAR ,START, END ,FOR, WHILE, ODDEVEN, PRINTFUNCTION, SIN, COS ,TAN, LOG ,FACTORIAL ,CASE, DEFAULT, SWITCH

**CFG**

**Context-free grammars** (CFGs) are used to describe [context-free languages](https://brilliant.org/wiki/context-free-languages/). A context-free grammar is a set of recursive rules used to generate patterns of [strings](https://brilliant.org/wiki/strings/). A context-free grammar can describe all [regular languages](https://brilliant.org/wiki/regular-languages/) and more, but they cannot describe all possible languages.

**My compiler CFGs**

program: MAIN ':' START line END

line: /\* NULL \*/

| line statement

;

statement: ';'

| declaration ';'

| expression ';'

|VAR '=' expression ';'

| WHILE '(' NUM '<' NUM ')' START statement END

| IF '(' expression ')' START expression ';' END %prec IFX

| IF '(' expression ')' START expression ';' END ELSE START expression ';' END

| PRINTFUNCTION '(' expression ')' ';'

| FACTORIAL '(' NUM ')' ';'.

| ODDEVEN '(' NUM ')' ';'

| ARRAY TYPE VAR '(' NUM ')' ';'

| SWITCH '(' NUM ')' START SWITCHCASE END

| FOR '(' NUM ',' NUM ',' NUM ')' START statement END

declaration : TYPE ID1

TYPE : INT

| FLOAT

| CHAR

ID1 : ID1 ',' VAR

|VAR

SWITCHCASE: casegrammer

|casegrammer defaultgrammer

casegrammer: /\*empty\*/

| casegrammer casenumber

casenumber: CASE NUM ':' expression ';'

defaultgrammer: DEFAULT ':' expression ';'

expression: NUM

| VAR

| expression '+' expression

| expression '-' expression

| expression '\*' expression

| expression '/' expression

| expression '%' expression

| expression '^' expression

| expression '<' expression

| expression '>' expression

| '(' expression ')'

| SIN expression

| COS expression

| TAN expression

| LOG expression

**Features of this compiler**

1.header file

2. Main function

3.Comments

4.Variable declaration

5. IF ELSE Block

6.Variable assignment

7. Array Declaration

8. For loop

9. While loop

10. Print function

11. Switch Case

12.Mathematical Expression

Addition, Subtraction, Multiplication, Division, Power, Log () Operation, Sin () operation, Tan () operation,

Cos () operation.

**Discussion**

This is a bottom up parser and the parser generate a set of tokens. In a program Conditional logic, Loops, Variable declaration, Mathematical function, array, header file are used. Unfortunately, shift reduce problem occur in the compilation time. If any grammar match with the input text then the compiler shows the token is declared.

**Discussion**

This compiler is similar like the python language and this compiler written into C programming language.

**References and links**

<https://github.com/FaisalAhmedBijoy/Compiler-lab/tree/master/Compiler%20project%20python>