Simple Compiler using Flex and Bison

Student Name : ABID ALI

Student ID : 2019380141

Contents

Objectives

1.To know about the basic compiling process.

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 syntax 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.

Features

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.Class and inheritance

12.Try Catch

13.Functions

14.Build in Odd Even, Factorial Function

15.Mathematical Expression

a. Addition

b. Subtraction

c. Multiplication

d. Division

e. Power

f. Log () Operation

g. Sin () operation

h. Tan () operation

I. Cos () operation

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.

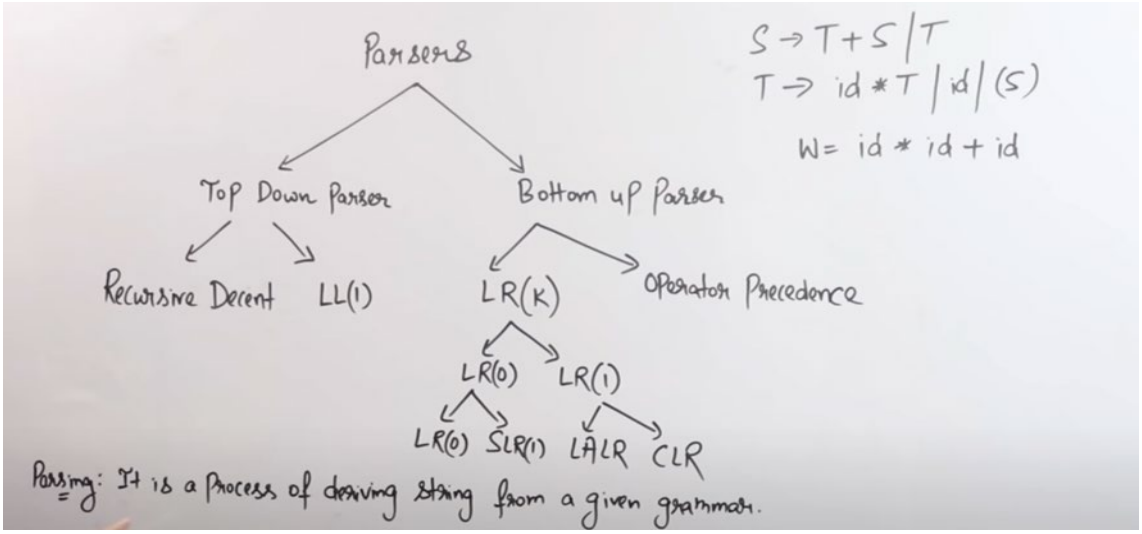
My compiler tokens

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. A context-free grammar is a set of recursive rules used to generate patterns of strings. A context-free grammar can describe all regular languages and more, but they cannot describe *all* possible languages.

Parsers



Parsing:

Parsing is a process of deriving string from a given grammar.

Lexical analysis:

1)This is the first step of compiler.

2)Lexical analysis gives stream of tokens.

Example: Taken characters and convert into stream of tokens.

This stream of tokens will go to parser. This phase is called syntax analysis phase.

Remarks:

The strings that we got, will be checked does that string belongs to particular grammar or not.

We use parser tree/derivative tree/syntax tree

The Grammar that we used is CFG (Context Free Grammar)

Configuration

We need to download bison for windows, download flex for windows and mingw

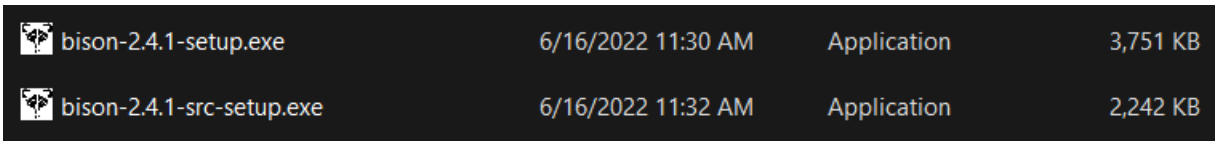


Fig: Bison setup

Fig: Flex setup

Fig: mingw-get-setup

Run the program in terminal

Press the button to run the program

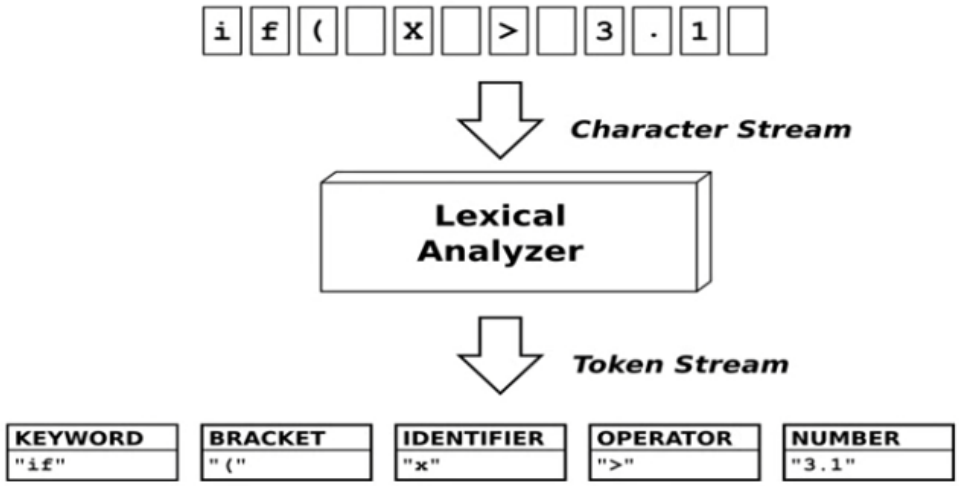
Alternate way to run the program in terminal

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

LEX

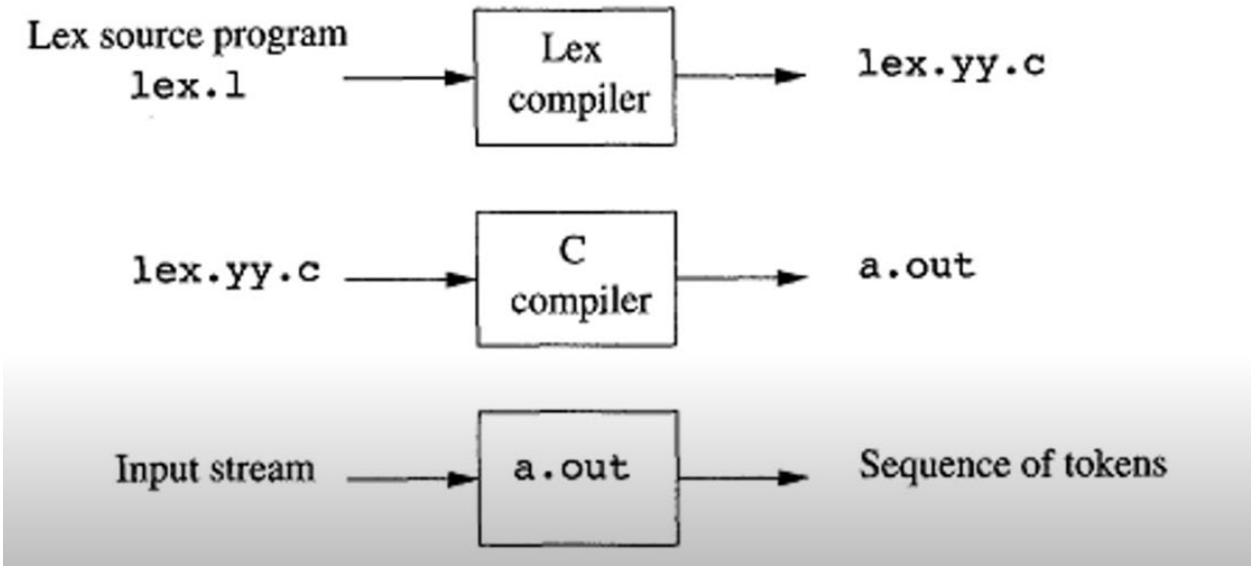
1. Lex is a tool that generate lexical analyser.

2. Lexical analyser is the first phase of compiler which take input as source code and generate output as token.



3. The input that we put is in lex language and tool itself is called lex compiler.

4. The lex compiler transform the input patterns into a transition diagram and generates code in a file called lex.yy.c This part is the lexical scanner created by flex.



Structure of Lex Programs:

A Lex program has the following form:

{declarations}

%%

{translation rules}

%%

{auxiliary functions}

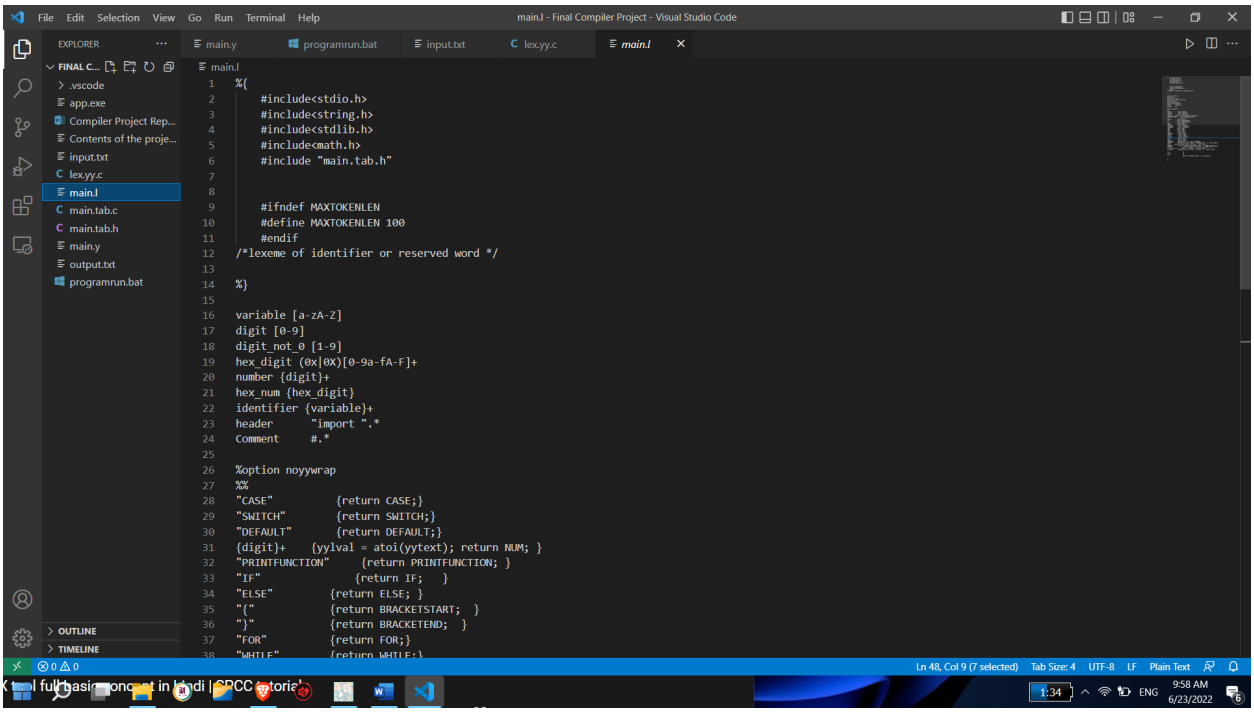


Fig: Lex file

- The declarations section includes declarations of variables
- The translation rules have the form:

  Pattern {Action}
- The third section holds whatever auxiliary functions are used in the actions. Alternatively, these functions can be compiled separately and loaded with the lexical analyser

Declaration

```
variable [a-zA-Z]
digit [0-9]
digit_not_0 [1-9]
hex_digit (0x|0X)[0-9a-fA-F]+
number {digit}+
hex_num {hex_digit}
identifier {variable}+
header          "import ".*
Comment                 #.*
```

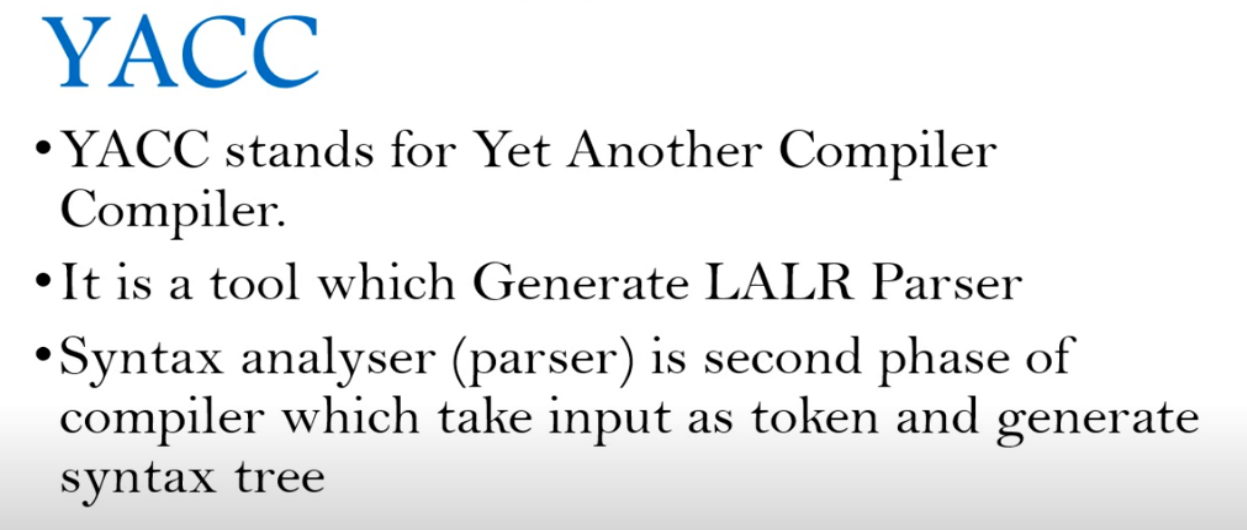
Translation Rules

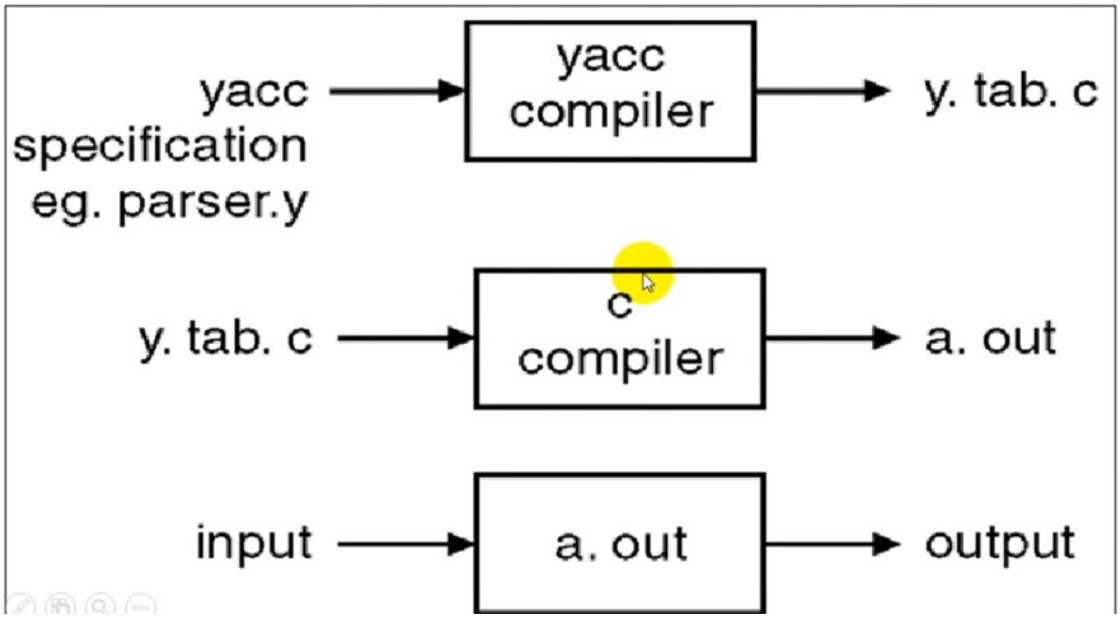
```
%%
"CASE"                          {return CASE;}
"SWITCH"                {return SWITCH;}
"DEFAULT"               {return DEFAULT;}
```

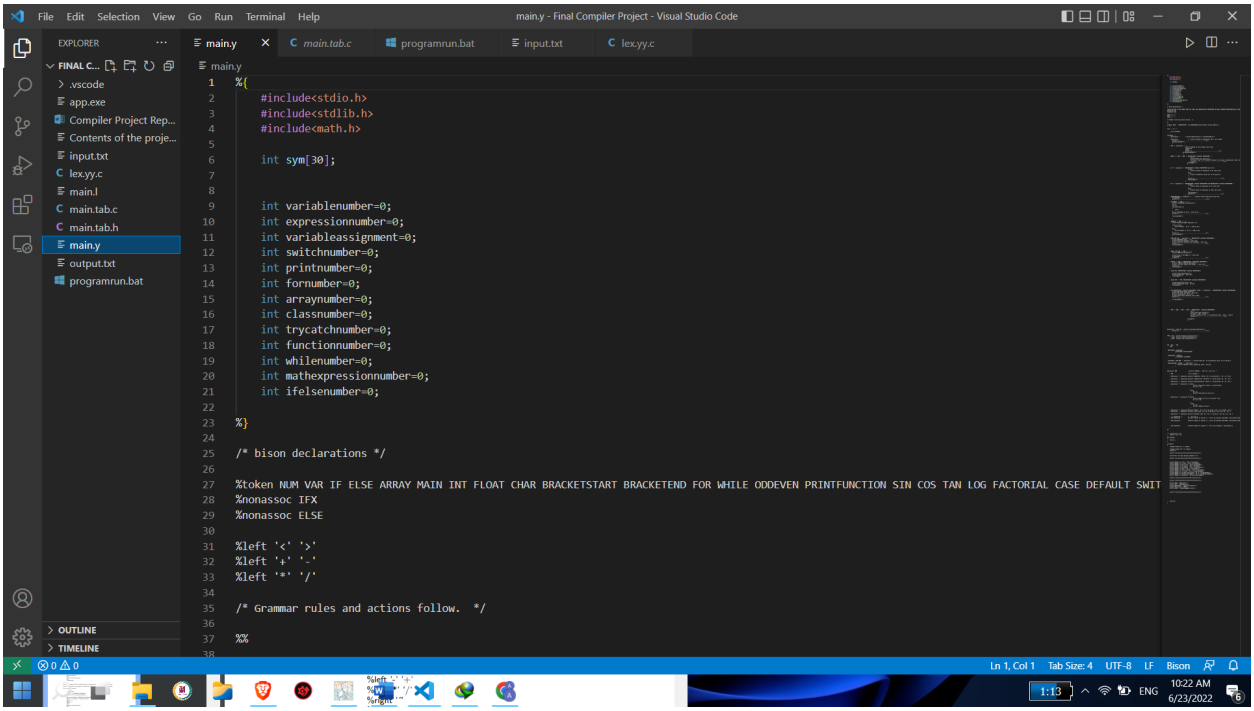
```
{digit}+          {yylval = atoi(yytext); return NUM; }
"PRINTFUNCTION"           {return PRINTFUNCTION; }
"IF"                      {return IF;   }
"ELSE"                    {return ELSE; }
"{"               {return BRACKETSTART;  }
"}"               {return BRACKETEND;    }
"FOR"                     {return FOR;}
"WHILE"                   {return WHILE;}
"FACTORIAL"                     {return FACTORIAL;}
"ODDEVEN"           {return ODDEVEN;}
"SIN"             {return SIN;}
"COS"         {return COS;}
"TAN"         {return TAN;}
"LOG"                   {return LOG;}
"INTEGER"             {return INT;}
"CHAR"                    {return CHAR;}
"FLOAT"                         {return FLOAT;}
"ARRAY"                         {return ARRAY;}
"TRY"                   {return TRY;}
"CATCH"                         {return CATCH;}
"FUNCTION"          {ECHO;printf(" "); return FUNCTION;}
"CLASS"                         {printf("New Class  Name : ");ECHO;printf(" "); return CLASS;}
"Main"    {printf("\nMain Function Start\n"); return MAIN; }
{Comment}             {printf("\nSingle line Comment found :: ");ECHO;printf("\n");}
{header}                                {printf("\nHeader file found :\n");ECHO;printf("\n");}
[-+/*<>=,():;%^]     {yylval = yytext[0];       return *yytext;}
{variable}          {ECHO;printf("\n"); yylval = *yytext - 'a'; return  VAR; }
```

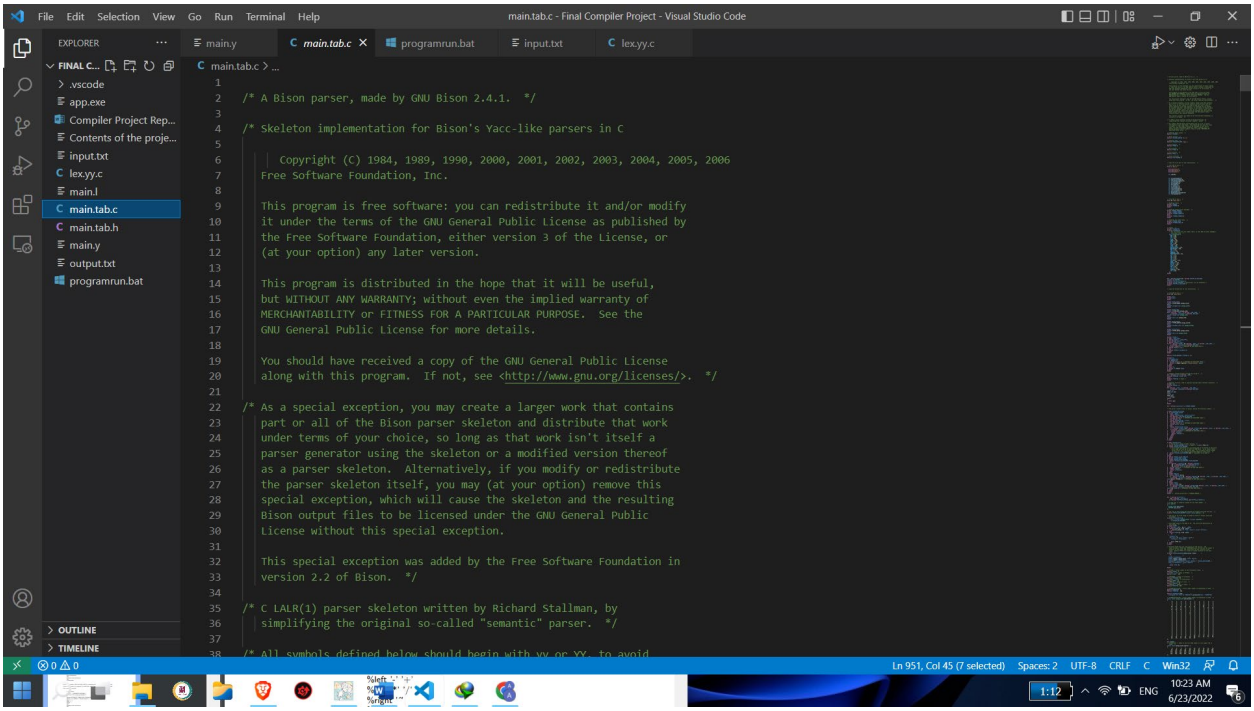
```
[ ]*                    {}
[\n]*                   {}
[\t]*                   {}
.                       {printf("\nUnknown Syntax : %s\n",yytext);}


%%
```

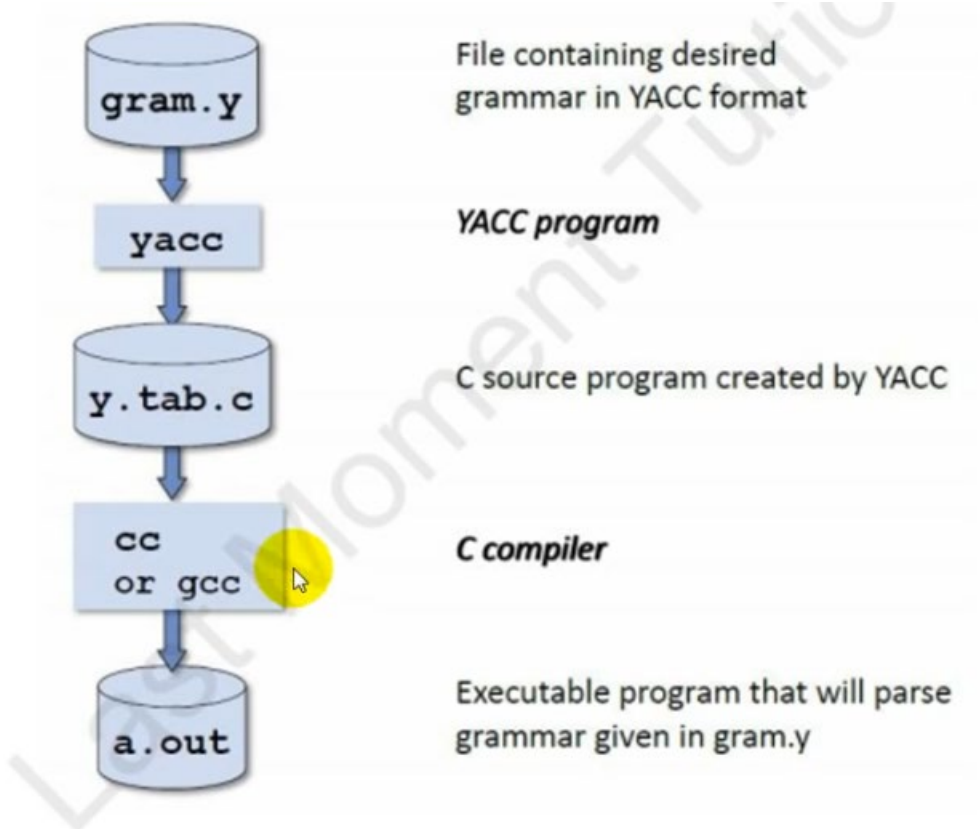
Yacc/Bison




yacc
specification
eg. parser.y
yacc
compiler
y. tab. c
y. tab. c
c
compiler
a. out
input
a. out
output


File Edit Selection View Go Run Terminal Help
main.y - Final Compiler Project - Visual Studio Code
EXPLORER
main.y
main.tab.c
programrun.bat
input.txt
lex.yy.c
FINAL C...
.vscode
app.exe
Compiler Project Rep...
Contents of the proje...
input.txt
lex.yy.c
main.l
main.tab.c
main.tab.h
main.y
output.txt
programrun.bat
OUTLINE
TIMELINE
%{
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int sym[30];
int variablenumber=0;
int expressionnumber=0;
int variableassignment=0;
int switchnumber=0;
int printnumber=0;
int fornumber=0;
int arraynumber=0;
int classnumber=0;
int trycatchnumber=0;
int functionnumber=0;
int whilenumber=0;
int mathexpressionnumber=0;
int ifelsenumber=0;
%}
/* bison declarations */
%token NUM VAR IF ELSE ARRAY MAIN INT FLOAT CHAR BRACKETSTART BRACKETEND FOR WHILE ODDEVEN PRINTFUNCTION SIN COS TAN LOG FACTORIAL CASE DEFAULT SWIT
%nonassoc IFX
%nonassoc ELSE
%left '<' '>'
%left '+' '-'
%left '*' '/'
/* Grammar rules and actions follow. */
%%
Ln 1, Col 1 Tab Size: 4 UTF-8 LF Bison
1:13
ENG
10:22 AM
6/23/2022


File Edit Selection View Go Run Terminal Help
main.tab.c - Final Compiler Project - Visual Studio Code
EXPLORER
main.y
main.tab.c
programrun.bat
input.txt
lex.yy.c
FINAL C...
.vscode
app.exe
Compiler Project Rep...
Contents of the proje...
input.txt
lex.yy.c
main.l
main.tab.c
main.tab.h
main.y
output.txt
programrun.bat
OUTLINE
TIMELINE
/* A Bison parser, made by GNU Bison 2.4.1. */
/* Skeleton implementation for Bison's Yacc-like parsers in C
Copyright (C) 1984, 1989, 1990, 2000, 2001, 2002, 2003, 2004, 2005, 2006
Free Software Foundation, Inc.
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>. */
/* As a special exception, you may create a larger work that contains
part or all of the Bison parser skeleton and distribute that work
under terms of your choice, so long as that work isn't itself a
parser generator using the skeleton or a modified version thereof
as a parser skeleton. Alternatively, if you modify or redistribute
the parser skeleton itself, you may (at your option) remove this
special exception, which will cause the skeleton and the resulting
Bison output files to be licensed under the GNU General Public
License without this special exception.
This special exception was added by the Free Software Foundation in
version 2.2 of Bison. */
/* C LALR(1) parser skeleton written by Richard Stallman, by
simplifying the original so-called "semantic" parser. */
/* All symbols defined below should begin with yy or YY, to avoid
Ln 951, Col 45 (7 selected) Spaces: 2 UTF-8 CRLF C Win32
1:12
ENG
10:23 AM
6/23/2022


gram.y
File containing desired grammar in YACC format
yacc
YACC program
y.tab.c
C source program created by YACC
cc
or gcc
C compiler
a.out
Executable program that will parse grammar given in gram.y

Syntax

Definitions

%%

Rules

%%

Supplementary Code

- **Definition Section**: All code between % and % is copied to the C file. The definitions section is where we configure various parser features such as defining token codes, establishing operator precedence and associativity and setting up variables used to communicate between the scanner and the parser.
- **Rules Section:** The required productions section is where we specify the grammar rule.
- **Supplementary Code Section:** It is used for ordinary C code that we want copied verbatim to the generated C file, declarations are copied to the top of the file, user subroutines to the bottom.

Definition and Declaration

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

```
    #include<math.h>

    int sym[30];


    int variablenumber=0;
    int expressionnumber=0;
    int variableassignment=0;
    int switchnumber=0;
    int printnumber=0;
    int fornumber=0;
    int arraynumber=0;
    int classnumber=0;
    int trycatchnumber=0;
    int functionnumber=0;
    int whilenumber=0;
    int mathexpressionnumber=0;
    int ifelsenumber=0;

%}

/* bison declarations */

%token NUM VAR IF ELSE ARRAY MAIN INT FLOAT CHAR BRACKETSTART BRACKETEND FOR
WHILE ODDEVEN PRINTFUNCTION SIN COS TAN LOG FACTORIAL CASE DEFAULT SWITCH CLASS
TRY CATCH FUNCTION
%nonassoc IFX
%nonassoc ELSE

%left '<' '>'
%left '+' '-'
%left '*' '/'
```

Translation Rules

```
%%

program: MAIN ':' BRACKETSTART  line BRACKETEND {printf("Main function END\n");}
     ;

line: /* NULL */

    | line statement
```

```
    ;

statement: ';'
    | declaration ';'       { printf("Declaration\n"); variablenumber++;}

    | expression ';'            {   printf("\nvalue of expression: %d\n", $1);
$$=$1;
        printf("\n.....................................\n");
        expressionnumber++;
        }

    | VAR '=' expression ';' {
                            printf("\nValue of the variable: %d\n",$3);
                            sym[$1]=$3;
                            $$=$3;
                            printf("\n.....................................\n
");
                        variableassignment++;
                        }

    | WHILE '(' NUM '<' NUM ')' BRACKETSTART statement BRACKETEND  {
                                    int i;
                                    printf("WHILE Loop execution");
                                    for(i=$3 ; i<$5 ; i++) {printf("\nvalue of
the loop: %d expression value: %d\n", i,$8);}
                                    printf("\n.............................
......\n");
                                whilenumber++;
                                }


    | IF '(' expression ')' BRACKETSTART statement BRACKETEND %prec IFX {
                                 if($3){
                                    printf("\nvalue of expression in
IF: %d\n",$6);
                                 }
                                 else{
                                    printf("\ncondition value zero in IF
block\n");
                                 }

                                 printf("\n.................................
..\n");
                                 ifelsenumber++;
                             }
```

```
    | IF '(' expression ')' BRACKETSTART statement BRACKETEND ELSE BRACKETSTART
statement BRACKETEND {
                                if($3){
                                    printf("value of expression in IF: %d\n",$6);
                                }
                                else{
                                    printf("value of expression in
ELSE: %d\n",$11);
                                }
                                ifelsenumber++;
                                printf("\n.....................................
..\n");
                            }
    | PRINTFUNCTION '(' expression ')' ';' {printf("\nPrint Expression %d\n",$3);
        printnumber++;
        printf("\n.....................................\n");}

    | FACTORIAL '(' NUM ')' ';' {
        printf("\nFACTORIAL declaration\n");
        int i;
        int f=1;
        for(i=1;i<=$3;i++)
        {
            f=f*i;
        }
        printf("FACTORIAL of %d is : %d\n",$3,f);
        printf("\n.....................................\n");

        functionnumber++;

        }

    | ODDEVEN '(' NUM ')' ';' {
        printf("Odd Even Number detection \n");

        if($3 %2 ==0){
            printf("Number : %d is -> Even\n",$3);
        }
        else{
            printf("Number is :%d is -> Odd\n",$3);
        }
        printf("\n.....................................\n");
        functionnumber++;
        }
```

```
    | FUNCTION VAR '(' expression ')' BRACKETSTART statement BRACKETEND {
        printf("FUNCTION found :  \n");
        printf("Function Parameter : %d\n",$4);
        printf("Function internal block statement : %d\n",$7);
        printf("\n.....................................\n");
        functionnumber++;


    }

    | ARRAY TYPE VAR '(' NUM ')' ';' {
        printf("ARRAY Declaration\n");

        printf("Size of the ARRAY is : %d\n",$5);
        arraynumber++;
        printf("\n.....................................\n");
    }

    | SWITCH '(' NUM ')' BRACKETSTART  SWITCHCASE BRACKETEND {
        printf("\nSWITCH CASE Declaration\n");
        printf("\nFinally Choose Case number :-> %d\n",$3);
        printf("\n.....................................\n");
        switchnumber++;
    }

    | CLASS VAR  BRACKETSTART statement BRACKETEND {

        printf("Class Declaration\n");
        printf("Expression : %d\n",$4);
        classnumber++;
    }

    | CLASS VAR ':' VAR  BRACKETSTART statement BRACKETEND {

        printf("Inheritance occur \n");
        printf("Expression value : %d",$6);
        classnumber++;

    }

    | TRY BRACKETSTART  statement BRACKETEND  CATCH '(' expression ')'
BRACKETSTART statement BRACKETEND{
        printf("TRY CATCH block found\n");
        printf("TRY Block operation : %d\n",$3);
```

```
        printf("CATCH Value : %d\n",$7);
        printf("Catch Block operation :%d\n",$10);
        printf("\n.....................................\n");

        trycatchnumber++;
    }







    | FOR '(' NUM ',' NUM ',' NUM ')' BRACKETSTART  statement BRACKETEND {
                                    int i;
                                    printf("FOR Loop execution");
                                    for(i=$3 ; i<$5 ; i=i+$7 )
                                    {printf("\nvalue of the  i: %d expression
value : %d\n", i,$10);}
                                    printf("\n.................................
......\n");

                                fornumber++;
                                }


    ;



declaration : TYPE ID1   {printf("\nvariable detection\n");
        printf("\n.....................................\n");}
            ;


TYPE : INT   {printf("interger declaration\n");}
     | FLOAT  {printf("float declaration\n");}
     | CHAR   {printf("char declaration\n");}
     ;



ID1 : ID1 ',' VAR
    |VAR
    ;
```

```
 SWITCHCASE: casegrammer
           |casegrammer defaultgrammer
           ;

 casegrammer: /*empty*/
           | casegrammer casenumber
           ;

 casenumber: CASE NUM ':' expression ';' {printf("Case No : %d & expression
value :%d \n",$2,$4);}
           ;
 defaultgrammer: DEFAULT ':' expression ';' {
                printf("\nDefault case & expression value : %d",$3);
           }
       ;


expression: NUM                { printf("\nNumber :  %d\n",$1 ); $$ = $1;  }

    | VAR                      { $$ = sym[$1]; }

    | expression '+' expression {printf("\nAddition :%d+%d = %d
\n",$1,$3,$1+$3 );  $$ = $1 + $3;}

    | expression '-' expression {printf("\nSubtraction :%d-%d=%d \n ",$1,$3,$1-
$3); $$ = $1 - $3; }

    | expression '*' expression {printf("\nMultiplication :%d*%d \n
",$1,$3,$1*$3); $$ = $1 * $3; }

    | expression '/' expression { if($3){
                                        printf("\nDivision :%d/%d \n
",$1,$3,$1/$3);
                                        $$ = $1 / $3;

                                    }
                                    else{
                                        $$ = 0;
                                        printf("\ndivision by zero\n\t");
                                    }
                                }
    | expression '%' expression { if($3){
                                        printf("\nMod :%d % %d \n",$1,$3,$1 %
$3);
                                        $$ = $1 % $3;
```

```
                                    }
                                    else{
                                        $$ = 0;
                                        printf("\nMOD by zero\n");
                                    }
                                }
    | expression '^' expression {printf("\nPower  :%d ^ %d \n",$1,$3,$1 ^
$3);  $$ = pow($1 , $3);}
    | expression '<' expression {printf("\nLess Than :%d < %d \n",$1,$3,$1 < $3);
$$ = $1 < $3 ; }

    | expression '>' expression {printf("\nGreater than :%d > %d \n ",$1,$3,$1 >
$3); $$ = $1 > $3; }

    | '(' expression ')'        {    $$ = $2; }
    | SIN expression            {printf("\nValue of Sin(%d)
is : %lf\n",$2,sin($2*3.1416/180)); $$=sin($2*3.1416/180);}

    | COS expression            {printf("\nValue of Cos(%d)
is : %lf\n",$2,cos($2*3.1416/180)); $$=cos($2*3.1416/180);}



    | LOG expression            {printf("\nValue of Log(%d)
is : %lf\n",$2,(log($2))); $$=(log($2));}

    ;
%%
```

Testcases

Input

import trycatch

import math

import time

import abid

import 2019380141

import Computer Science

Output

Header file found :

import trycatch

Header file found :

import math

Header file found :

import time

Header file found :

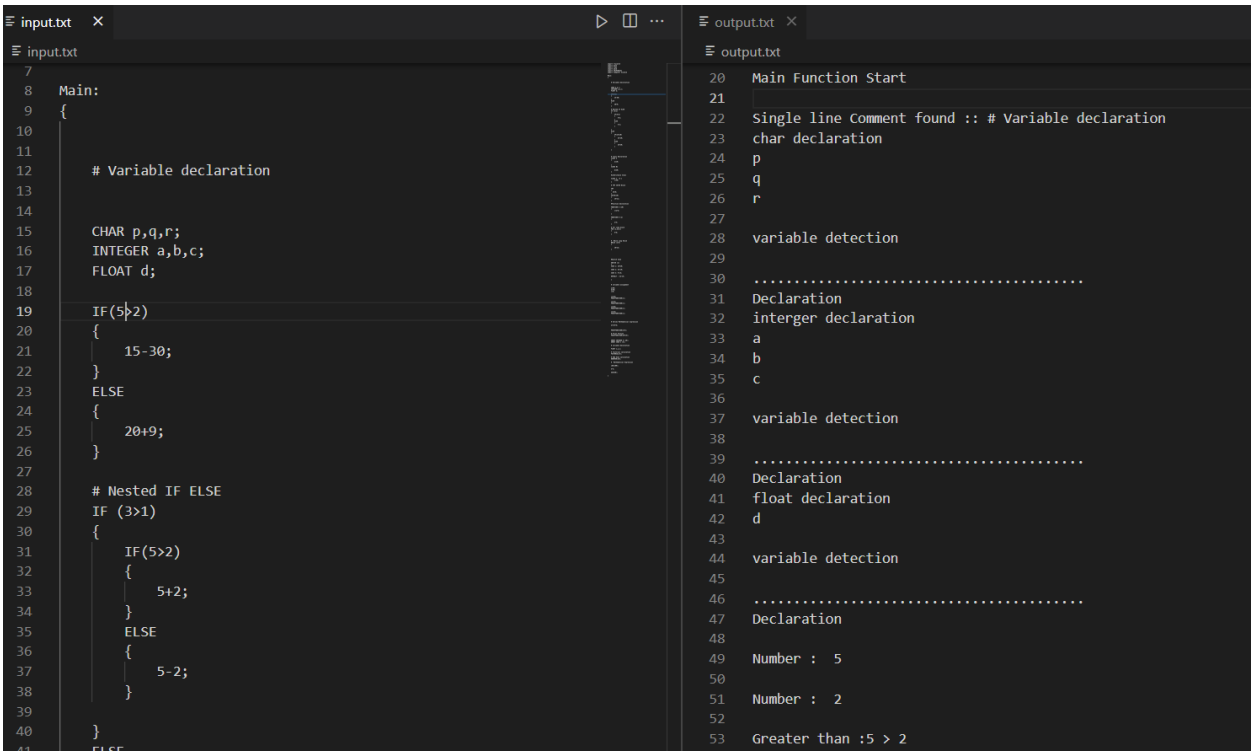
import abid

Header file found :

import 2019380141

Header file found :

import Computer Science



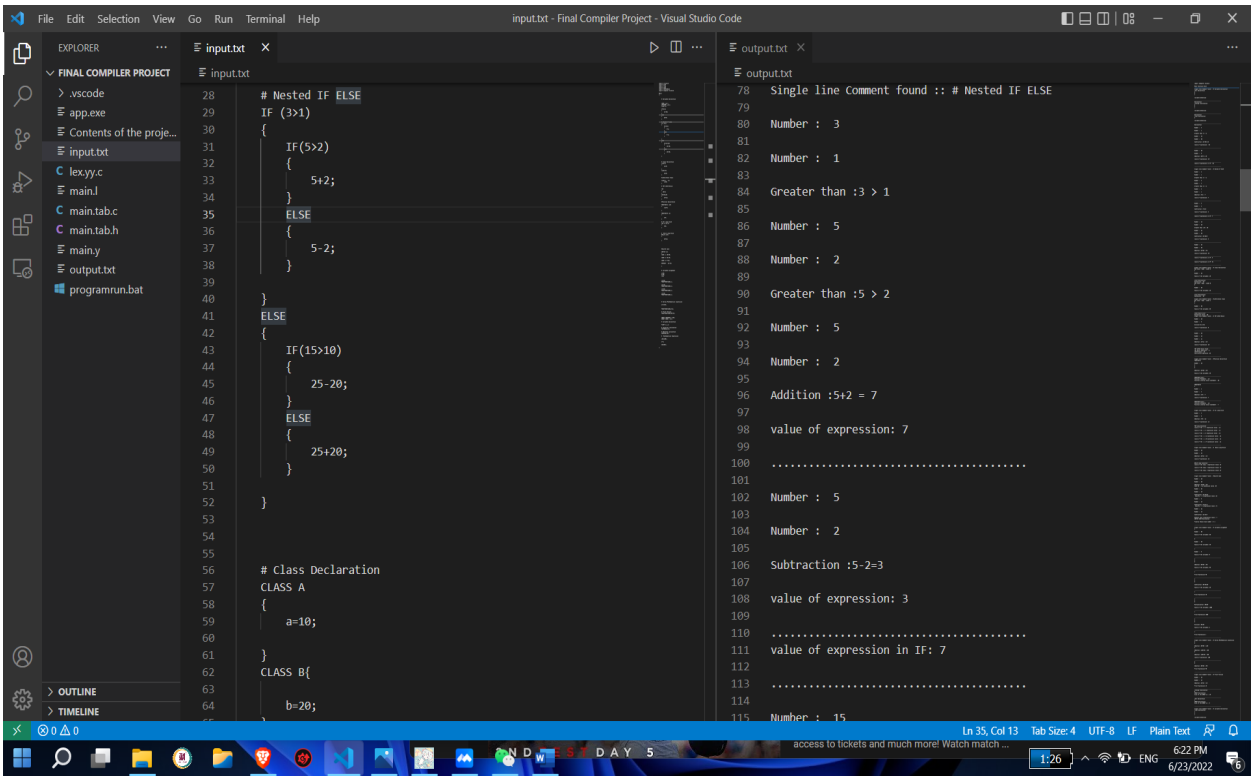
input.txt

```
Main:
{

    # Variable declaration


    CHAR p,q,r;
    INTEGER a,b,c;
    FLOAT d;

    IF(5>2)
    {
        15-30;
    }
    ELSE
    {
        20+9;
    }

    # Nested IF ELSE
    IF (3>1)
    {
        IF(5>2)
        {
            5+2;
        }
        ELSE
        {
            5-2;
        }

    }
```

output.txt

```
Main Function Start

Single line Comment found :: # Variable declaration
char declaration
p
q
r

variable detection

.....
Declaration
interger declaration
a
b
c

variable detection

.....
Declaration
float declaration
d

variable detection

.....
Declaration

Number :  5

Number :  2

Greater than :5 > 2
```

Main function detected, variable declaration, variable showed, IF ELSE statement is working



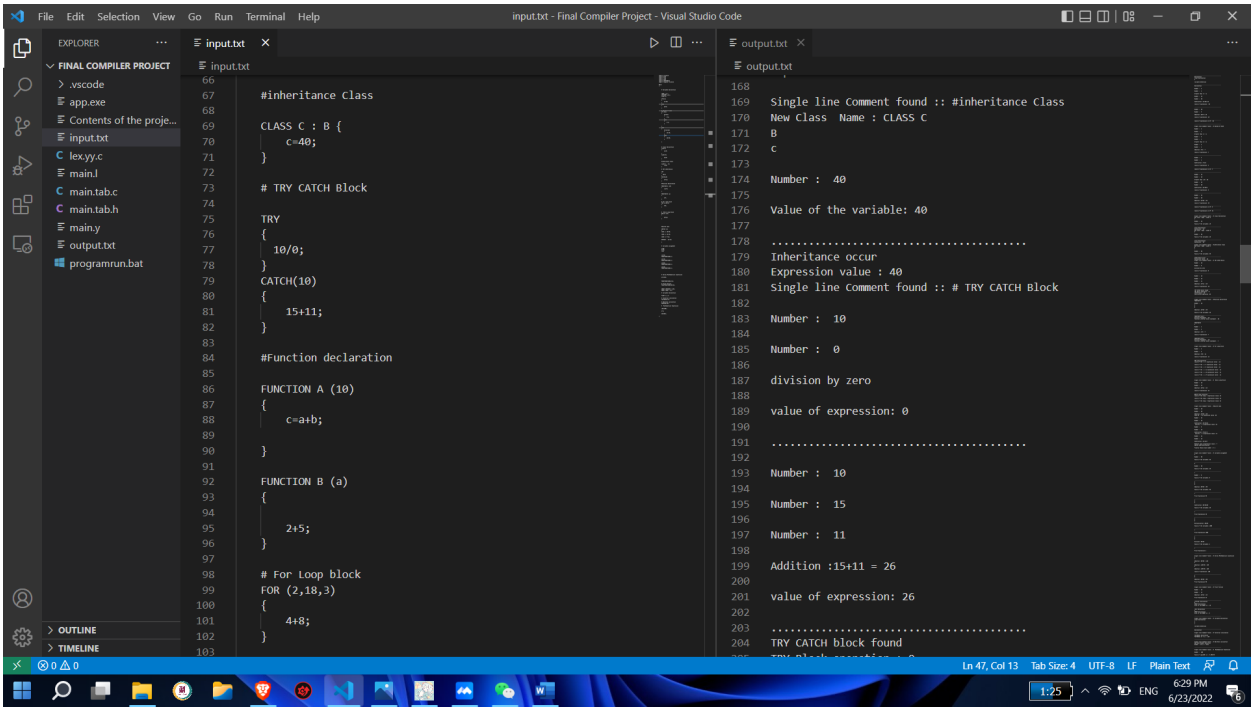
input.txt

```
    # Nested IF ELSE
    IF (3>1)
    {
        IF(5>2)
        {
            5+2;
        }
        ELSE
        {
            5-2;
        }

    }
    ELSE
    {
        IF(15>10)
        {
            25-20;
        }
        ELSE
        {
            25+20;
        }

    }



    # Class Declaration
    CLASS A
    {
        a=10;

    }
    CLASS B{

        b=20;
```

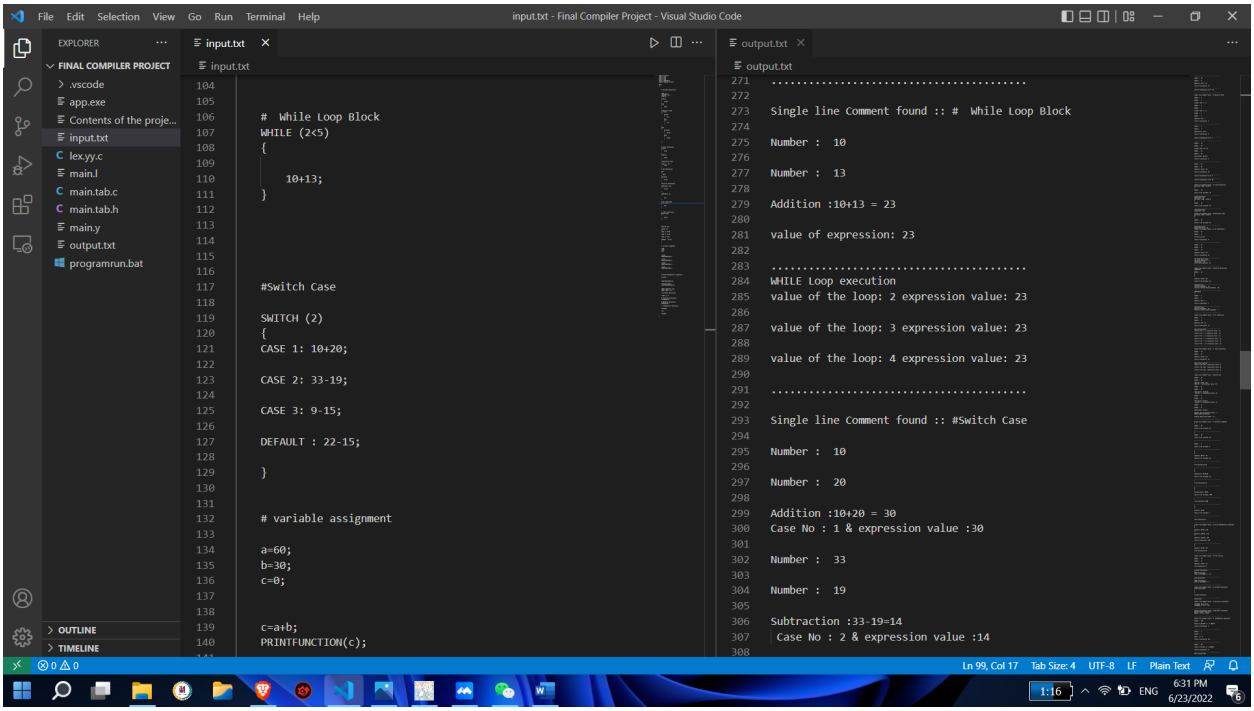
output.txt

```
Single line Comment found :: # Nested IF ELSE

Number :  3

Number :  1

Greater than :3 > 1

Number :  5

Number :  2

Greater than :5 > 2

Number :  5

Number :  2

Addition :5+2 = 7

value of expression: 7

.....

Number :  5

Number :  2

Subtraction :5-2=3

value of expression: 3

.....
value of expression in IF: 7

.....

Number :  15
```

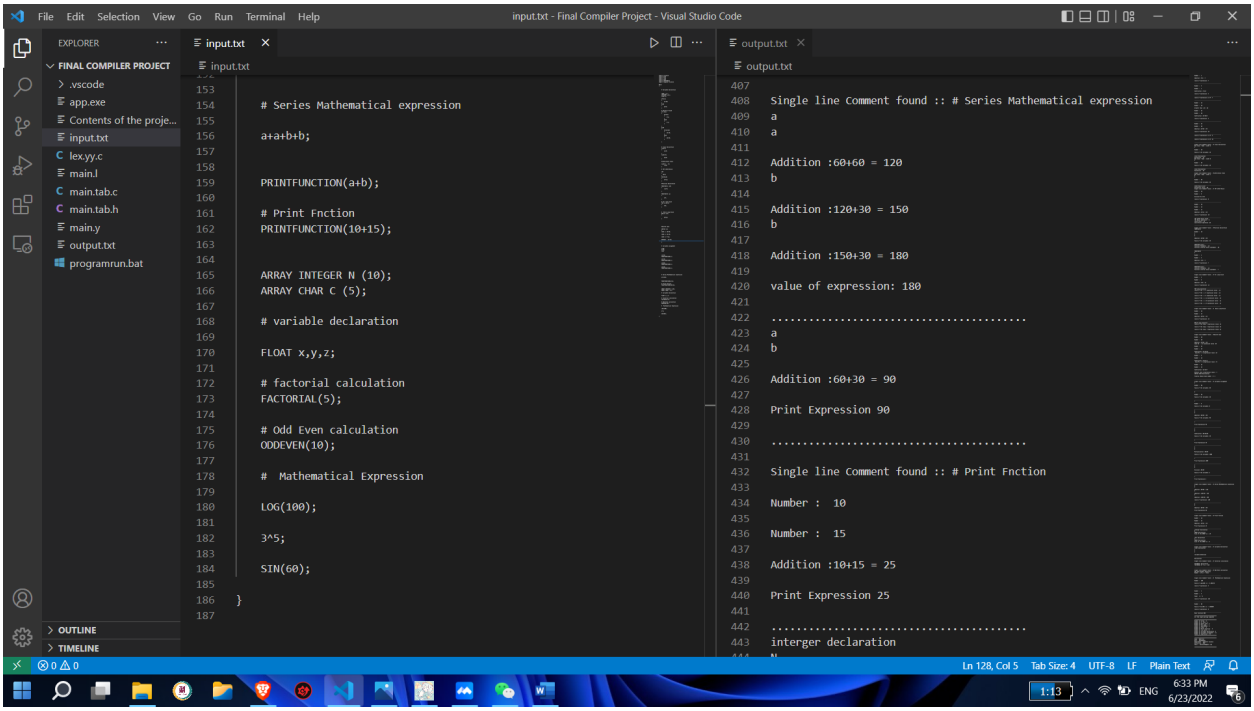
Nested IF ELSE and Class Declaration



Inheritance class, TRY CATCH Block, Function declaration, For Loop block



While Loop Block, Switch Case, variable assignment, PRINTFUNCTION



Series Mathematical expression, ARRAY INTEGER, ARRAY CHAR, FLOAT, FACTORIAL, ODDEVEN, LOG, SIN

Conclusion

I learned in this project the basic compiling process, translation of a high-level language into a low-level language, Flex and Bison for implementation of a compiler using C programming language, create a new language and it's semantic and syntax rules, checking different type of input and their output of the compiler, implementation of the Regular Expression and Context Free Grammar in the compiler.

Features available:

Header file, Main function, Comments, Variable declaration, IF ELSE Block, Variable assignment, Array Declaration, For loop, While loop, Print function, Class and inheritance, Try Catch, Functions, Build in Odd Even, Factorial Function

Mathematical Expression

a. Addition

b. Subtraction

c. Multiplication

d. Division

e. Power

f. Log () Operation

g. Sin () operation

h. Tan () operation

I. Cos () operation

References

1) https://www.gnu.org/software/bison/

2) https://en.wikipedia.org/wiki/GNU_Bison

3) https://en.wikipedia.org/wiki/Flex_(lexical_analyser_generator)

4) http://alumni.cs.ucr.edu/~lgao/teaching/flex.html

5) http://gnuwin32.sourceforge.net/packages/flex.htm

6) https://sourceforge.net/projects/gnuwin32/files/bison/2.4.1/

7) https://sourceforge.net/projects/mingw/

8) https://www.mingw-w64.org/

9)https://www.youtube.com/watch?v=54bo1qaHAfk&list=PLARg2IRFQdoXXMzfWgWZCJqhyFbSpMdKj

10)https://www.youtube.com/watch?v=0Cw658NjZZ4&list=PLbcKbyl11YO4tnneNSFmN6cHDDN7sWBGF