Report on

C++ Mini Compiler

Submitted in partial fulfillment of the requirements for ***Sem VI***

Compiler Design Laboratory

Bachelor of Technology
in
Computer Science & Engineering

Submitted by:

Kavya P K	**PES1201800151**
Roshni Poddar	**PES1201800161**
Neeli Krishna Dheeraj	**PES1201800182**
Shraddha Bharadwaj	**PES1201800306**

Under the guidance of

Madhura V
Assistant Professor
Department of CSE
PES University, Bengaluru

January – May 2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

TABLE OF CONTENTS

Chapter No.	Title	Page No.
1.	INTRODUCTION	3
1.1	Language	3
1.2	Sample Input File	3
1.3	Sample Output	4
2.	ARCHITECTURE OF LANGUAGE:	7
2.1	Syntax	7
2.2	Semantics	8
3.	LITERATURE SURVEY	8
4.	CONTEXT FREE GRAMMAR	9
5.	DESIGN STRATEGY	14
5.1	Lexical analysis	14
5.2	Syntax Analysis	15
5.3	Semantic analysis and Expression Evaluation	15
5.4	Intermediate Code Generation	16
5.5	Code Optimisation	16
5.6	Symbol Table	17
6.	IMPLEMENTATION DETAILS	18
6.1	Lexical analysis	18
6.2	Syntax Analysis	19
6.3	Semantic analysis and Expression Evaluation	19
6.4	Intermediate Code Generation	20
6.5	Code Optimisation	20
6.6	Instructions to run C++ Mini compiler	20
7.	RESULTS AND SHORTCOMINGS	20
8.	SNAPSHOTS	21
9.	CONCLUSIONS	38
10.	FURTHER ENHANCEMENTS	39
REFERENCES/BIBLIOGRAPHY		39

1. INTRODUCTION

1.1 Language

This project is a Mini compiler developed for C-like object oriented programming language, C++. The compiler designed takes care of basic syntax and semantics, along with if construct and switch construct.

The aim of this project is to generate an optimized intermediate code for a given C++ input source code.

The phases of the project are as follows:

1. Lexical Analysis
2. Semantic Analysis
3. Construction of Symbol table
4. Generating Three Address Code, and Quadruple format TAC
5. Code Optimisation

The project has been developed using LEX to identify tokens for patterns matched, YACC for parsing and semantic meaning and to generate Three Address Code, and finally using Python to optimise the TAC.

1.2 Sample Input

Sample input file that is successfully compiled by this project:

```
#include<iostream>
#include<stdlib.h>
using namespace std;
float a = 10;
int func()
{
    int a = 5;
    int c = a + 3 * 5 / 10;
}
class class1
{
   private: int b;
   public:
       int class_function(int d)
```

```
            {
                cout<<"In function "<<d<<endl;
            }
};
int main()
{
   class1 obj1;
   obj1.class_function(10);
       int a1[5]={1,2,3,4,5};
       int b1[]={1,2,3,4,5};
       int a = 9;
       int b = 10;

       int add = 3 + 5 + 5 + 8;
       int sub = 3 - 5;
       int op1 = 5;
       int op2 = 10;
       bool gt = op1 > op2;
       bool lteq = op1 <= op2;
       bool log_and = op1 && op2;
       bool log_or = op1 || op2;
       if(a <= b)
       {
       int day = 4;
       switch (day)
               {
                   case 1: {
                               int case1 = 4;
                               break;
                           }
                   case 2:
                   case 3: break;
                   case 4: cout << "CASE 4\n";
               }
       }
}
```

1.3 Sample Output

The TAC generated for this sample input:

```
a = 10
func{
a = 5
t1 = 3 * 5
t2 = t1 / 10
t3 = a + t2
c = t3
}
class1{
class1.class_function{
PARAM "INVOKED CLASS FUNCTION WITH VALUE ="
PARAM d
PARAM endl
call ( cout , 3 )
}
main{
PARAM 10
call ( obj1.class_function , 1 )
a1[0] = 1
a1[4] = 2
a1[8] = 3
a1[12] = 4
a1[16] = 5
b1[0] = 1
b1[4] = 2
b1[8] = 3
b1[12] = 4
b1[16] = 5
a = 9
b = 10
t4 = 3 + 5
t5 = t4 + 5
t6 = t5 + 8
add = t6
t7 = 3 - 5
sub = t7
op1 = 5
op2 = 10
t8 = op1 > op2
gt = t8
t9 = op1 <= op2
lteq = t9
t10 = op1 && op2
log_and = t10
t11 = op1 || op2
log_or = t11
t12 = a <= b
if t12 goto L1
goto L2
L1 :
day = 4
t13 = day == 1
```

```
if t13 goto L3
goto L4
L3 :
case1 = 4
L4 :
t14 = day == 2
if t14 goto L5
goto L6
L5 :
L6 :
t15 = day == 3
if t15 goto L7
goto L8
L7 :
call ( break , 0 )
L8 :
t16 = day == 4
if t16 goto L9
goto L10
L9 :
PARAM "CASE 4\n"
call ( cout , 1 )
L10 :
L2 :
}
```

In quadruple format:

```
|op           |op1          |op2          |result       |
-------------------------------------------------------------------
|=            |a            |             |10           |
|=            |a            |             |5            |
|*            |3            |5            |t1           |
|/            |t1           |10           |t2           |
|+            |a            |t2           |t3           |
|=            |c            |             |t3           |
|PARAM        |"In function "|            |             |
|PARAM        |d            |             |             |
|PARAM        |endl         |             |             |
|call         |cout         |3            |             |
|PARAM        |10           |             |             |
|call         |class_function|1           |             |
|=            []|1          |0            |0            |
|=            []|2          |4            |0            |
|=            []|3          |8            |0            |
|=            []|4          |12           |0            |
|=            []|5          |16           |a1           |
|=            []|1          |0            |0            |
|=            []|2          |4            |0            |
|=            []|3          |8            |0            |
|=            []|4          |12           |0            |
|=            []|5          |16           |b1           |
```

```
|=          |a          |           |9          |
|=          |b          |           |10         |
|+          |3          |5          |t4         |
|+          |t4         |5          |t5         |
|+          |t5         |8          |t6         |
|=          |add        |           |t6         |
|-          |3          |5          |t7         |
|=          |sub        |           |t7         |
|=          |op1        |           |5          |
|=          |op2        |           |10         |
|>          |op1        |op2        |t8         |
|=          |gt         |           |t8         |
|<=         |op1        |op2        |t9         |
|=          |lteq       |           |t9         |
|&&         |op1        |op2        |t10        |
|=          |log_and   |        |t10         |
|||         |op1        |op2        |t11        |
|=          |log_or     |           |t11        |
|<=         |a          |b          |t12        |
|if         |t12        |goto       |L1         |
|goto       |           |           |L2         |
|Label      |           |           |L1         |
|=          |day        |           |4          |
|==         |day        |1          |t13        |
|=          |case1      |           |4          |
|Label      |           |           |L4         |
|==         |day        |2          |t14        |
|if         |t14        |goto       |L5         |
|goto       |           |           |L6         |
|Label      |           |           |L5         |
|Label      |           |           |L6         |
|==         |day        |3          |t15        |
|Label      |           |           |L7         |
|Label      |           |           |L8         |
|==         |day        |4          |t16        |
|PARAM      |"CASE 4\n"|        |           |
|call       |cout       |1          |           |
|Label      |           |           |L10        |
|Label      |           |           |L2         |
```

2. ARCHITECTURE OF LANGUAGE

2.1 Syntax

The project supports compilation of the basic syntax and constructs in C++, including:

1. Global/local variables declarations and initializations,
2. Functions
3. Classes

4. Variables types - int, float, double, char, bool
5. Arrays and pointers
6. Arithmetic and Boolean expressions
7. If else construct
8. Switch construct

2.2 Semantics

The project takes care of the following cases:

1. Type checking - The variables are checked for appropriate types.
2. Declare variables before use - If an undeclared variable is used, appropriate error is displayed.
3. Variables declared can be used only in ways that are acceptable for the declared type. The variables present in an expression are type-compatible.
   For example,
   ```
   int a =5;
   int b = a + "hello"; // this throws error
   ```
4. The test expression used in an if and switch statement evaluates to a Boolean value if the values of the variables in the expression is known.
5. New declarations don't conflict with earlier declarations - if they do, error is displayed.
6. Break statements only appear in constructs.
7. The error statements reported include variable redeclaration, using variable without declaration, and type checking. This is done using checks with symbol tables, and error-specific statements in the grammar and yyerror.

3. LITERATURE SURVEY

This project has been developed by taking reference from various websites as listed in the reference section.

It also required a clear understanding of lex and yacc concepts as well as regex concepts, taking reference from material provided in the course, which include:

1. Lex and Yacc: A Brisk Tutorial By Saumya K. Debray, The University of Arizona
2. Mastering Regular Expressions by Jeffrey E. F. Friedl

4. CONTEXT FREE GRAMMAR

The context free grammar used for this project is as follows:

```
S     : START;

START : INCLUDE T_lt H T_gt START
      | INCLUDE "\"" H "\""  START
      | INCLUDE T_lt ID T_gt  START
      | INCLUDE "\"" ID "\""  START
      | INCLUDE T_lt H T_gt START_CODE
      | INCLUDE "\"" H "\"" START_CODE
      | INCLUDE T_lt ID T_gt START_CODE
      | INCLUDE "\"" ID "\"" START_CODE
      ;

START_CODE : USING NAMESPACE ID T_semi GLOBAL
           | GLOBAL
           ;

GLOBAL  : ASSIGN_EXPR T_semi  GLOBAL
                | EXPR T_semi  GLOBAL
                | DECLARATION T_semi  GLOBAL
                | FUNCTION_DEF GLOBAL
                | CLASS_DEF GLOBAL
                | PTR_EXPR T_semi  GLOBAL
                | OBJ_EXPR T_semi  GLOBAL
                | ARR_EXPR T_semi  GLOBAL
                | MAIN
                ;

ASSIGN_TOK  : ASSIGN_EXPR T_semi;

MAIN : TYPE MAINTOK  BODY_MAIN
      | TYPE MAINTOK BODY_MAIN   FUNC_AND_DEC
      | TYPE MAINTOK   BODY_MAIN  CLASS_REC
      | TYPE MAINTOK    BODY_MAIN FUNC_AND_DEC CLASS_REC
      | TYPE MAINTOK  BODY_MAIN   CLASS_REC FUNC_AND_DEC
      ;

BODY_MAIN  : flower_paran_o
            | flower_paran_o flower_paran_c
            ;

CLASS_REC: CLASS_REC CLASS_DEF
      | CLASS_DEF
      ;

FUNC_AND_DEC: ASSIGN_TOK FUNC_AND_DEC
      | EXPR_TOK FUNC_AND_DEC
      | ASSIGN_TOK
      | EXPR_TOK
      | FUNCTION_DEF
      | FUNCTION_DEF FUNC_AND_DEC
      | DECL_TOK FUNC_AND_DEC
      | DECL_TOK
      ;
```

```
EXPR_TOK: EXPR T_semi
         ;

DECL_TOK : DECLARATION T_semi
        ;

func_paran_o:   T_lround

func_paran_c:   T_rround

flower_paran_o: T_lflower

flower_paran_c: T_rflower

BODY  : flower_paran_o C flower_paran_c
     | flower_paran_o flower_paran_c
     ;

PARAMS  : TYPE ID
         | TYPE ID T_comma PARAMS
         | CHAR_PTR ID
         | CHAR_PTR ID T_comma PARAMS
         | PTR_TYPE ID
         | PTR_TYPE ID T_comma PARAMS
         ;

FUNCTION_DEF  : TYPE ID func_paran_o PARAMS func_paran_c
                     | TYPE ID func_paran_o func_paran_c
                     | TYPE ID func_paran_o PARAMS func_paran_c T_semi
                     | TYPE ID func_paran_o func_paran_c T_semi
                     | VOID ID func_paran_o PARAMS func_paran_c
                     | VOID ID func_paran_o func_paran_c
                     | VOID ID func_paran_o PARAMS func_paran_c T_semi
                     | VOID ID func_paran_o func_paran_c T_semi
                     | PTR_TYPE ID func_paran_o PARAMS func_paran_c
                     | PTR_TYPE ID func_paran_o func_paran_c
                     | PTR_TYPE ID func_paran_o PARAMS func_paran_c T_semi
                     | PTR_TYPE ID func_paran_o func_paran_c T_semi
                     | CHAR_PTR ID func_paran_o PARAMS func_paran_c T_semi
                     | CHAR_PTR ID func_paran_o func_paran_c T_semi
                     | CHAR_PTR ID func_paran_o PARAMS func_paran_c
                     | CHAR_PTR ID func_paran_o func_paran_c
                     ;

C     : C statement T_semi
     | C LOOPS
     | statement T_semi
     | LOOPS
     | C ID func_paran_o
     | C ID func_paran_o  func_paran_c T_semi
     | ID func_paran_o
     | ID func_paran_o  func_paran_c T_semi
     | error
     ;


CLASS_DEF : CLASS ID flower_paran_o
             | CLASS ID flower_paran_o flower_paran_c T_semi
             ;

CLASS_BODY  : PUBLIC T_col FUNC_AND_DEC
                 | PUBLIC T_col FUNC_AND_DEC CLASS_BODY
                 | PRIVATE T_col FUNC_AND_DEC
                 | PRIVATE T_col FUNC_AND_DEC CLASS_BODY
                 | PROTECTED T_col FUNC_AND_DEC
                 | PROTECTED T_col FUNC_AND_DEC CLASS_BODY
                 | PUBLIC T_col CLASS_BODY
                 | PRIVATE T_col CLASS_BODY
```

```
                 | PROTECTED T_col CLASS_BODY
                 | PUBLIC T_col
                 | PRIVATE T_col
                 | PROTECTED T_col
                 ;

LOOPS : SWITCH T_lround LIT T_rround flower_paran_o SWITCHBODY flower_paran_c
     | IF_HEAD LOOPBODY
     | IF_HEAD LOOPBODY ELSE
     ;

IF_HEAD
     :IF T_lround COND T_rround

SWITCHBODY
     : SWITCHBODY CASE LIT T_col
     | CASE LIT T_col
     | SWITCHBODY CASE LIT T_col BREAK
     | SWITCHLOOP1 BREAK  T_semi
     | SWITCHLOOP1
     | SWITCHLOOP2 BREAK   T_semi
     | SWITCHLOOP2
     | SWITCHLOOP3 BREAK
     | SWITCHLOOP3
     | SWITCHLOOP4 BREAK  T_semi
     | SWITCHLOOP4
     ;
SWITCHLOOP1
     :SWITCHBODY  CASE LIT T_col  LOOPBODY ;

SWITCHLOOP2
     : CASE LIT T_col LOOPBODY;

SWITCHLOOP3
      :DEFAULT T_col  LOOPBODY ;

SWITCHLOOP4
      :DEFAULT T_col
      ;

LOOPBODY  : flower_paran_o C flower_paran_c
       | flower_paran_o C BREAK T_semi flower_paran_c
       | T_semi
       | statement T_semi
       | flower_paran_o flower_paran_c
       | error
       ;

statement : DECLARATION
            | ASSIGN_EXPR
            | EXPR
            | TERNARY_EXPR
            | PTR_EXPR
            | ARR_EXPR
            | OBJ_EXPR
            | PRINT
            | RETURN
            | RETURN T_lround EXPR T_rround
            | RETURN EXPR
            | error
            ;

COND  : LIT RELOP LIT
     | LIT
     | LIT RELOP LIT bin_boolop LIT RELOP LIT
     | un_boolop LIT RELOP LIT
     | LIT bin_boolop LIT
     | un_boolop T_lround LIT T_rround
```

```
     | un_boolop LIT
     | error
     ;

ASSIGN_EXPR
     : ID T_eq EXPR
     | TYPE ID T_eq EXPR
     | CHAR_PTR ID T_eq STRING_LIT
     ;

DECLARATION
     : TYPE ID
     | DECLARATION T_comma ID
     | CHAR_PTR ID
     ;

TERNARY_EXPR
     : T_lround COND T_rround '?' statement T_col statement
     ;

REP_PRINT:
       T_lt T_lt LIT
     | REP_PRINT  T_lt T_lt LIT
     | T_lt T_lt ENDL
     |  REP_PRINT T_lt T_lt ENDL
          ;

PRINT
     : COUT  REP_PRINT
     ;
LIT
     : ID
     | STRING_LIT
     | CHAR_CONST
     | T_DIGIT
     | T_REAL
     | T_TRUE
     | T_FALSE
     ;

TYPE
     : INT
     | CHAR
     | FLOAT
     | STRING
     | BOOL
     ;

PTR_TYPE : INT_PTR
     | FLOAT_PTR
     ;

PTR_EXPR: PTR_TYPE ID
     | PTR_TYPE ID T_eq T_amp ID
     | CHAR_PTR ID T_eq T_amp ID
     | ID T_eq T_amp ID
     ;

ARR_EXPR: TYPE ID T_lsq T_DIGIT T_rsq
     | TYPE ID T_lsq T_DIGIT T_rsq T_eq
     | TYPE ID T_sq T_eq
     | ID T_lsq T_DIGIT T_rsq T_eq
     ;

OBJ_EXPR : ID ID
     | ID T_dot ID T_eq LIT
     | ID T_dot ID T_lround T_rround
     | ID T_dot ID T_lround ARRAY T_rround
```

```
     ;

ARRAY : LIT
     | ARRAY T_comma
     ;

EXPR
   : EXPR bin_boolop EXP
   | EXP
   ;

EXP: EXP RELOP T
   | T
   ;

T
   : T T_pl F
   | T T_min F
   | F
   ;

F
   : F T_mul G
   | F T_div G
   | G
   ;

G
   : LIT
   ;

RELOP
     : T_lt
     | T_gt
     | T_lteq
     | T_gteq
     | T_neq
     | T_eqeq
     ;

bin_boolop
     : T_and
     | T_or
     ;

un_boolop
     : T_not
     ;
```

5. DESIGN STRATEGY

5.1 Lexical analysis

1. The LEX tool was used to create a scanner for C++ language.
2. The scanner transforms the source file from a stream of bits and bytes into a series of meaningful tokens containing information that will be used by the later stages of the compiler.

3. The scanner also scans for the comments (single-line and multiline comments) and writes the source file without comments onto an output file which is used in the further stages.
4. All tokens included are of the form T_.Eg: T_pl for '+',T_min for '-', T_lt for '<' etc.
5. Using yylval and yytext for token value , token type, and yylineno for line number where the token occurs and the column number in that line. For example:

```
Token: int          Token Type: T_INT       Line number: 7          Column number: 0
Token: s            Token Type: T_ID        Line number: 7          Column number: 3
Token: =            Token Type: T_eq        Line number: 7          Column number: 4
Token: 4            Token Type: T_DIGIT     Line number: 7          Column number: 5
```

6. Skipping over white spaces and recognizing all keywords, operators, variables and constants is handled in this phase.
7. Scanning error is reported when the input string does not match any rule in the lex file using yyerror.
8. The rules are regular expressions which have corresponding actions that execute on a match with the source input.

5.2 Syntax Analysis

1. Syntax analysis is responsible for verifying that the sequence of tokens forms a valid sentence given the definition of your Programming Language grammar.
2. The design implementation supports
    a. Variable declarations and initializations.
    b. Variables of type int,float,char,bool.
    c. Arithmetic and boolean expressions Arrays .
3. Constructs handled are if-else and switch constructs.
4. YACC tool is used for parsing. It reports shift-reduce and reduce-reduce conflicts on parsing an ambiguous grammar. All conflicts that arose while writing the grammar were resolved.
5. Appropriate Context Free Grammar is written to parse the tokens in order to check the syntax and on mismatch in the parsing, error is displayed.

6. After encountering an error, the compiler continues parsing after displaying the line number at which the error occurs.

5.3 Semantic analysis and expression evaluation

1. Operands in an arithmetic expression are checked for type compatibility and appropriate message is displayed in case of type mismatch.
2. Temporaries are generated using the right precedence and associativity.
3. Symbol table creation and expression evaluation:
    a. For each scope a symbol table is generated, eg,:

```
Name    Type    Line number value
b        float   6       56.0
s        int     7       4
d        float   8       20.0
p        int     10      2
t1       int     11      20.000000
mr       int     11      t1
t2       int     12      5.000000
dr       int     12      t2
ro       int*    13      NOT_DEFINED
test_func        int_func        14      A_FUNCTION
class1   class   19      A_CLASS
```

Figure: Example of Symbol Table Generated

    b. Symbol table stores the name of the identifier , the type and the value.
    c. In case of an array, the value is not defined.
    d. In case of a function type is defined as <return type>_func.
    e. By keeping track of the current block in which the current token being parsed exists in:
        i. A token in the scope as current scope is added to the same table
        ii. If it enters a different scope a new table for that scope is created and until that scope is exited tokens are added to that scope itself.
        iii. On exit from that scope the pointer moves to the previous scope and the same continues.

5.4 Intermediate code generation

1. Intermediate code is generated after the tokens are parsed through the yacc file.
2. Intermediate code tends to be machine independent code.
3. Three Address Code – A statement involving no more than three references (two for operands and one for result) is known as three address statements. A sequence of three address statements is known as three address code.
4. Three address statements are of the form `x = y op z,` here x, y, z will have an address (memory location).
5. Example – The three address code for the expression

   `a + b * c + d :`

   i. `T 1 = b * c`

   ii. `T 2 = a + T 1`

   iii. `T 3 = T 2 + d`
6. T1, T2, T3 are temporary variables. The format used to represent Three address Code is Quadruples. It is shown with 4 columns- operator, operand1, operand2, and result.

5.5 Code Optimisation

1. Code optimisation takes intermediate code as the input and using a python script following optimisation techniques are carried out:
   a. **Dead code elimination** - Any statements that occur after a return statement will not be executed and can be removed from the code so as to curb unwanted propagation of values.
   b. **Strength reduction** - Multiplication and Division by 2 is replaced by their bitwise operation which proves to be less expensive
   c. **Constant folding** - any arithmetic or logical expression with constants as operands can be computed beforehand so as to propagate the values
   d. **Constant propagation** - any expression with a variable whose value is available is substituted and computed.

e. Constant folding and propagation occurs in a loop until no other changes can be made in a code.
f. Output is optimised code as TAC.

5.6 Symbol Table

Symbol Table is made using a linked list indexed with scope and each node in the linked list contains all the variables in the scope with name, type , value stored as attributes.
The following methods are allowed on the Symbol Table.

Insert: If a variable is declared the insert function is called , which checks if the variable is already declared if not variable is added into the symbol table at given scope

Change_value : After some assignment of each variable the type of each variable is checked if matched the value of the variable is changed to assigned expression

Display: Display function iterates through the entire linked list and displays all the variables declared in it titled with scope

6. IMPLEMENTATION DETAILS

6.1 Lexical analysis

The tools used are LEX and YACC. Lexical analysis is done by using regex patterns to identify various tokens.

The lex file is as follows:

```
%{
   #include<string.h>
   #include<stdio.h>
   #define YYSTYPE YACC
   unsigned int SourceCol=0;
   #include"y.tab.h"
%}

%e  1019
%p  2807
%n  371
%k  284
%a  1213
%o  1117
%x ML_COMMENT

NL  [\n]
alpha [A-Za-z_]
digit [0-9]
digits {digit}+
opFraction \.{digits}
opExponent [Ee][+-]?{digits}
real {digits}{opFraction}?{opExponent}?
delim [\t|' ']
whitespace {delim}+
%option yylineno
%%
"//".*        { printf("Comment at line %d\n", yylineno); }
"/*"          { printf("Multi line comment starts from %d ", yylineno); BEGIN(ML_COMMENT); }
<ML_COMMENT>"*/"    { printf("Multi line comment ends at %d\n", yylineno); BEGIN(INITIAL); }
<ML_COMMENT>[^*\n]+
```

```
<ML_COMMENT>"*"
<ML_COMMENT>"\n"    { ; }
{NL}     {SourceCol=0;}
[ \t\n]
"while"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);SourceCol+=yyleng;return WHILE;}
"for"        {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return FOR;}
"class"     {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return CLASS;}
"public"     {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return PUBLIC;}
"private"    {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return PRIVATE;}
"protected"  {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);printf("Token: %s \t \t Token Type: %s \t Line number: %d
\t Column number: %d \t \n",yylval.string,"T_PROTECTED",yylineno,SourceCol);SourceCol+=yyleng;return PROTECTED;}
"if"        {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return IF;}
"else"       {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);SourceCol+=yyleng;return ELSE;}
"switch"     {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);printf("Token: %s \t \t Token Type: %s \t Line number: %d
\t Column number: %d \t \n",yylval.string,"T_SWITCH",yylineno,SourceCol);SourceCol+=yyleng;return SWITCH;}
"case"       {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);printf("Token: %s \t \t Token Type: %s \t Line number: %d
\t Column number: %d \t \n",yylval.string,"T_CASE",yylineno,SourceCol);SourceCol+=yyleng;return CASE;}
"default"    {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);printf("Token: %s \t \t Token Type: %s \t Line number: %d
\t Column number: %d \t \n",yylval.string,"T_DEFAULT",yylineno,SourceCol);SourceCol+=yyleng;return DEFAULT;}
"cout"       {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return COUT; }
"endl"       {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return ENDL;}
"break"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return BREAK;}
"continue"   {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return CONTINUE;}
"return"     {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return RETURN;}

"int"        {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return INT;}
"float"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return FLOAT;}
"char"       {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return CHAR;}
"void"       {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return VOID;}
"string"     {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return STRING;}
"bool"       {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return BOOL;}
"int*"       {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return INT_PTR;}
"float*"     {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return FLOAT_PTR;}
"char*"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);SourceCol+=yyleng;return CHAR_PTR;}
"#include"   {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return INCLUDE;}
"main()"     {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return MAINTOK;}
"using"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return USING;}
"namespace"  {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return NAMESPACE;}
"true"       {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_TRUE;}
"false"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_FALSE;}
{digits}    {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext); return T_DIGIT;}
{real}      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_REAL;}
\".*\"       {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return STRING_LIT;}
\'.\'        {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return CHAR_CONST;}
{alpha}({alpha}|{digit}){0,31}    {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return ID;}
{alpha}({alpha}|{digit})*"\.h"?   {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext); return H;}
"<"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_lt;}
">"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_gt;}
"="      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_eq;}
"<="     {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_lteq;}
">="     {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_gteq;}
"=="     {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_eqeq;}
"!="     {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext); return T_neq;}
"+"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_pl;}
"-"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext); return T_min;}
"*"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext); return T_mul;}
"/"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_div;}
"++"     {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_incr;}
"--"     {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_decr;}
"!"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_neq;}
"||"     {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);  return T_or;}
"&&"     {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_and;}
"[]"     {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext); return T_sq;}
"["      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext); return T_lsq;}
"]"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_rsq;}
"&"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_amp;}
":"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_col;}
"("      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext); return T_lround;}
")"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_rround;}
"{"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_lflower;}
"}"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext); return T_rflower;}
";"      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext);return T_semi;}
","      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext); return T_comma;}
"."      {yylloc.first_line=yylineno;yylloc.first_column = SourceCol; yylloc.last_column = SourceCol+yyleng;yylval.string=strdup(yytext); return T_dot;}
.       {printf("Invalid character %s \n",yytext);return yytext[0];}
%%
```

6.2 Syntax and Semantic Analysis

The context free grammar (as mentioned above) was written. Tools used - YACC.

Each grammar contains an associated semantic action that evaluates the parsed tokens and adds variables to the symbol table, or if wrong, displays error.

6.3 Symbol Table

The symbol table has been developed using a linked list of nodes with the structure:

```
typedef struct NODE
     {
     char* name;
     char* type;
     int lineno;
     char value[40];
     }NODE;

typedef struct FINAL
{
     int scope_arr[100];
     int len;
     int count;
     NODE data_arr[100];
     struct FINAL *next;
     }FINAL;

typedef struct LIST{
     FINAL *head;
     FINAL *tail;
     int count;
     }LIST;
```

6.4 Intermediate Code Generation

Intermediate code generation is done for all correct code in the yacc file. This is done by adding appropriate print statements to a file for each line in the grammar. Quadruple format is also obtained in this manner, containing op, op1,op2,result.

6.5 Code Optimization

Code Optimisation has been done using python. The three address code is passed to a python script that performs constant folding, propagation, strength reduction and dead code elimination.

6.6 Instructions to Run the C++ Mini Compiler

Execute the following commands:

```
lex lexer.l
```

```
yacc -d parser.y
gcc y.tab.c -ll -ly
./a.out input_file.cpp
python3 optimization.py
```

7. RESULTS

The results obtained from the mini compiler consist of outputs at the end of each phase:

1. List of tokens, line number, column number and value at the end of the lexical phase.
2. Symbol table containing variables, values, line used and scope at the end of syntax and semantic analysis.
3. Intermediate Code in the form of TAC and in Quadruple format.

Possible Shortcomings of this Project:

1. Takes care of only if and switch constructs.
2. Does not display error messages for all possible errors. Scope of Project is limited.

8. SNAPSHOTS

1. Lexical Analysis

Input File:

```
#include<iostream>
#include<stdlib.h>
using namespace std;

int func()
{
    float var = 5e4;
        string s = "abc";
        char c = 'c';
}

class class1
{
   private:
   public:
};

int main()
{
       int a1[5] = {1,2,3,4,5};
       int op1 = 3 + 5 + 7;
       int op2 = 3 - 5;
```

```
        bool gt = op1 > op2;
        bool lteq = op1 <= op2;
        bool log_and = op1 && op2;
        bool log_or = op1 || op2;
        int day = 4;
        switch (day)
        {
                    case 4: cout << "CASE 4\n";
                    case 5: cout << "CASE 5\n";
        }
        if(op1 >= op2)
        {
            class1 obj1;
        }
        else{}
}
```

Tokens Generated:

```
Token: }                Token Type: T_rflower   Line number: 16         Column number: 0
Token: ;                Token Type: T_semi      Line number: 16         Column number: 1
Token: int              Token Type: T_INT       Line number: 18         Column number: 0
Token: main()           Token Type: T_MAINTOK   Line number: 18         Column number: 3
Token: {                Token Type: T_lflower   Line number: 19         Column number: 0
Token: int              Token Type: T_INT       Line number: 20         Column number: 0
Token: a1               Token Type: T_ID        Line number: 20         Column number: 3
Token: [                Token Type: T_lsq       Line number: 20         Column number: 5
Token: 5                Token Type: T_DIGIT     Line number: 20         Column number: 6
Token: ]                Token Type: T_rsq       Line number: 20         Column number: 7
Token: =                Token Type: T_eq        Line number: 20         Column number: 8
Token: {                Token Type: T_lflower   Line number: 20         Column number: 9
Token: 1                Token Type: T_DIGIT     Line number: 20         Column number: 10
Token: ,                Token Type: T_comma     Line number: 20         Column number: 11
Token: 2                Token Type: T_DIGIT     Line number: 20         Column number: 12
Token: ,                Token Type: T_comma     Line number: 20         Column number: 13
Token: 3                Token Type: T_DIGIT     Line number: 20         Column number: 14
Token: ,                Token Type: T_comma     Line number: 20         Column number: 15
Token: 4                Token Type: T_DIGIT     Line number: 20         Column number: 16
Token: ,                Token Type: T_comma     Line number: 20         Column number: 17
Token: 5                Token Type: T_DIGIT     Line number: 20         Column number: 18
Token: }                Token Type: T_rflower   Line number: 20         Column number: 19
Token: ;                Token Type: T_semi      Line number: 20         Column number: 20
Token: int              Token Type: T_INT       Line number: 21         Column number: 0
Token: op1              Token Type: T_ID        Line number: 21         Column number: 3
Token: =                Token Type: T_eq        Line number: 21         Column number: 6
Token: 3                Token Type: T_DIGIT     Line number: 21         Column number: 7
Token: +                Token Type: T_pl        Line number: 21         Column number: 8
Token: 5                Token Type: T_DIGIT     Line number: 21         Column number: 9
Token: +                Token Type: T_pl        Line number: 21         Column number: 10
Token: 7                Token Type: T_DIGIT     Line number: 21         Column number: 11
Token: ;                Token Type: T_semi      Line number: 21         Column number: 12
Token: int              Token Type: T_INT       Line number: 22         Column number: 0
Token: op2              Token Type: T_ID        Line number: 22         Column number: 3
Token: =                Token Type: T_eq        Line number: 22         Column number: 6
Token: 3                Token Type: T_DIGIT     Line number: 22         Column number: 7
Token: -                Token Type: T_min       Line number: 22         Column number: 8
Token: 5                Token Type: T_DIGIT     Line number: 22         Column number: 9
Token: ;                Token Type: T_semi      Line number: 22         Column number: 10
Token: bool             Token Type: T_BOOL      Line number: 24         Column number: 0
Token: gt               Token Type: T_ID        Line number: 24         Column number: 4
Token: :                Token Type: T_col       Line number: 14         Column number: 7
Token: public           Token Type: T_PUBLIC    Line number: 15         Column number: 0
Token: :                Token Type: T_col       Line number: 15         Column number: 6
```

```
Token: <                Token Type: T_lt        Line number: 31         Column number: 10
Token: <                Token Type: T_lt        Line number: 31         Column number: 11
Token: "CASE 4\n"               Token Type: T_STRINGLIT         Line number: 31         Column number: 12
Token: ;                Token Type: T_semi      Line number: 31         Column number: 22
Token: case             Token Type: T_CASE      Line number: 32         Column number: 0
Token: 5                Token Type: T_DIGIT     Line number: 32         Column number: 4
Token: :                Token Type: T_col       Line number: 32         Column number: 5
Token: cout             Token Type: T_COUT      Line number: 32         Column number: 6
Token: <                Token Type: T_lt        Line number: 32         Column number: 10
Token: <                Token Type: T_lt        Line number: 32         Column number: 11
Token: "CASE 5\n"               Token Type: T_STRINGLIT         Line number: 32         Column number: 12
Token: ;                Token Type: T_semi      Line number: 32         Column number: 22
Token: }                Token Type: T_rflower   Line number: 33         Column number: 0
Token: if               Token Type: T_IF        Line number: 34         Column number: 0
Token: (                Token Type: T_lround    Line number: 34         Column number: 2
Token: op1              Token Type: T_ID        Line number: 34         Column number: 3
Token: >=               Token Type: T_gteq      Line number: 34         Column number: 6
Token: op2              Token Type: T_ID        Line number: 34         Column number: 8
Token: )                Token Type: T_rround    Line number: 34         Column number: 11
Token: {                Token Type: T_lflower   Line number: 35         Column number: 0
Token: class1           Token Type: T_ID        Line number: 36         Column number: 0
Token: obj1             Token Type: T_ID        Line number: 36         Column number: 6
Token: ;                Token Type: T_semi      Line number: 36         Column number: 10
Token: }                Token Type: T_rflower   Line number: 37         Column number: 0
Token: else             Token Type: T_ELSE      Line number: 38         Column number: 0
Token: {                Token Type: T_lflower   Line number: 38         Column number: 4
Token: }                Token Type: T_rflower   Line number: 38         Column number: 5
Token: }                Token Type: T_rflower   Line number: 39         Column number: 0
Input accepted.
```

2. Syntax and Semantic Analysis

```
Token: gt               Token Type: T_ID        Line number: 24         Column number: 4
Token: =                Token Type: T_eq        Line number: 24         Column number: 6
Token: op1              Token Type: T_ID        Line number: 24         Column number: 7
Token: >                Token Type: T_gt        Line number: 24         Column number: 10
Token: op2              Token Type: T_ID        Line number: 24         Column number: 11
Token: ;                Token Type: T_semi      Line number: 24         Column number: 14
Token: bool             Token Type: T_BOOL      Line number: 25         Column number: 0
Token: lteq             Token Type: T_ID        Line number: 25         Column number: 4
Token: =                Token Type: T_eq        Line number: 25         Column number: 8
Token: op1              Token Type: T_ID        Line number: 25         Column number: 9
Token: <=               Token Type: T_lteq      Line number: 25         Column number: 12
Token: op2              Token Type: T_ID        Line number: 25         Column number: 14
Token: ;                Token Type: T_semi      Line number: 25         Column number: 17
Token: bool             Token Type: T_BOOL      Line number: 26         Column number: 0
Token: log_and          Token Type: T_ID        Line number: 26         Column number: 4
Token: =                Token Type: T_eq        Line number: 26         Column number: 11
Token: op1              Token Type: T_ID        Line number: 26         Column number: 12
Token: &&               Token Type: T_and       Line number: 26         Column number: 15
Token: op2              Token Type: T_ID        Line number: 26         Column number: 17
Token: ;                Token Type: T_semi      Line number: 26         Column number: 20
Token: bool             Token Type: T_BOOL      Line number: 27         Column number: 0
Token: log_or           Token Type: T_ID        Line number: 27         Column number: 4
Token: =                Token Type: T_eq        Line number: 27         Column number: 10
Token: op1              Token Type: T_ID        Line number: 27         Column number: 11
Token: ||               Token Type: T_or        Line number: 27         Column number: 14
Token: op2              Token Type: T_ID        Line number: 27         Column number: 16
Token: ;                Token Type: T_semi      Line number: 27         Column number: 19
Token: int              Token Type: T_INT       Line number: 28         Column number: 0
Token: day              Token Type: T_ID        Line number: 28         Column number: 3
Token: =                Token Type: T_eq        Line number: 28         Column number: 6
Token: 4                Token Type: T_DIGIT     Line number: 28         Column number: 7
Token: ;                Token Type: T_semi      Line number: 28         Column number: 8
Token: switch           Token Type: T_SWITCH    Line number: 29         Column number: 0
Token: (                Token Type: T_lround    Line number: 29         Column number: 6
Token: day              Token Type: T_ID        Line number: 29         Column number: 7
Token: )                Token Type: T_rround    Line number: 29         Column number: 10
Token: {                Token Type: T_lflower   Line number: 30         Column number: 0
Token: case             Token Type: T_CASE      Line number: 31         Column number: 0
Token: 4                Token Type: T_DIGIT     Line number: 31         Column number: 4
Token: :                Token Type: T_col       Line number: 31         Column number: 5
Token: cout             Token Type: T_COUT      Line number: 31         Column number: 6
```

I. Syntax Analysis:
Output: Throws error with line number

Input File:

```
#include<iostream>
#include<stdlib.h>
using namespace std;

int func()
{
    float 12var = 5e4; //error
        string s = "abc";
        char c = 'c';
}

class class1
{
   private:
   public:
};

int main()
{
       int a1[5 = {1,2,3,4,5}; //error
       int op1 = 3 + 5 + 7;
       int op2 = 3 - 5;
       bool gt = op1 >| op2; //error
       bool lteq = op1 <= op2;
       bool log_and = op1 && op2;
       bool log_or = op1 || op2;
       int day = 4;
       switch (day)
       {
                   CASE 3: break;
                   case 4: cout << "CASE 4\n"; //error
       }
       if(op1 >= op2)
       {
           class1 obj1;
       }
       else{}
}
```

```
(base) roshni@roshni-Latitude-7490:~/CD PROJECT$ ./a.out < syntax.cpp
Token: #include          Token Type: T_INCLUDE   Line number: 1          Column number: 0
Token: <                 Token Type: T_lt        Line number: 1          Column number: 8
Token: iostream                  Token Type: T_ID        Line number: 1          Column number: 9
Token: >                 Token Type: T_gt        Line number: 1          Column number: 17
Token: #include          Token Type: T_INCLUDE   Line number: 2          Column number: 0
Token: <                 Token Type: T_lt        Line number: 2          Column number: 8
Token: stdlib.h          Token Type: T_H         Line number: 2          Column number: 9
Token: >                 Token Type: T_gt        Line number: 2          Column number: 17
Token: using             Token Type: T_USING     Line number: 3          Column number: 0
Token: namespace         Token Type: T_NAMESPACE         Line number: 3          Column number: 5
Token: std               Token Type: T_ID        Line number: 3          Column number: 14
Token: ;                 Token Type: T_semi      Line number: 3          Column number: 17
Token: int               Token Type: T_INT       Line number: 5          Column number: 0
Token: func              Token Type: T_ID        Line number: 5          Column number: 3
Token: (                 Token Type: T_lround    Line number: 5          Column number: 7
Token: )                 Token Type: T_rround    Line number: 5          Column number: 8
Token: {                 Token Type: T_lflower   Line number: 6          Column number: 0
Token: float             Token Type: T_FLOAT     Line number: 7          Column number: 0
Token: 12                Token Type: T_DIGIT     Line number: 7          Column number: 5
Error at line 7
Token: var               Token Type: T_ID        Line number: 7          Column number: 7
Token: =                 Token Type: T_eq        Line number: 7          Column number: 10
Token: 5e4       Token Type: T_REAL      Line number: 7          Column number: 11
Token: ;                 Token Type: T_semi      Line number: 7          Column number: 14
Token: string            Token Type: T_STRING    Line number: 8          Column number: 0
Token: s                 Token Type: T_ID        Line number: 8          Column number: 6
Token: =                 Token Type: T_eq        Line number: 8          Column number: 7
Token: "abc"             Token Type: T_STRINGLIT         Line number: 8          Column number: 8
Token: ;                 Token Type: T_semi      Line number: 8          Column number: 13
Token: char              Token Type: T_CHAR      Line number: 9          Column number: 0
Token: c                 Token Type: T_ID        Line number: 9          Column number: 4
Token: =                 Token Type: T_eq        Line number: 9          Column number: 5
Token: 'c'               Token Type: T_CHARCONST         Line number: 9          Column number: 6
Token: ;                 Token Type: T_semi      Line number: 9          Column number: 9
Token: }                 Token Type: T_rflower   Line number: 10         Column number: 0
Token: class             Token Type: T_CLASS     Line number: 12         Column number: 0
Token: class1            Token Type: T_ID        Line number: 12         Column number: 5
Token: {                 Token Type: T_lflower   Line number: 13         Column number: 0
Token: private           Token Type: T_PRIVATE   Line number: 14         Column number: 0
Token: :                 Token Type: T_col       Line number: 14         Column number: 7
Token: public            Token Type: T_PUBLIC    Line number: 15         Column number: 0
Token: :                 Token Type: T_col       Line number: 15         Column number: 6
Token: }                 Token Type: T_rflower   Line number: 16         Column number: 0
Token: ;                 Token Type: T_semi      Line number: 16         Column number: 1
Token: int               Token Type: T_INT       Line number: 18         Column number: 0
Token: main()            Token Type: T_MAINTOK   Line number: 18         Column number: 3
Token: {                 Token Type: T_lflower   Line number: 19         Column number: 0
Token: int               Token Type: T_INT       Line number: 20         Column number: 0
Token: a1                Token Type: T_ID        Line number: 20         Column number: 3
```

```
Token: log_and           Token Type: T_ID        Line number: 25         Column number: 4
Token: =                 Token Type: T_eq        Line number: 25         Column number: 11
Token: op1               Token Type: T_ID        Line number: 25         Column number: 12
Token: &&                Token Type: T_and       Line number: 25         Column number: 15
Token: op2               Token Type: T_ID        Line number: 25         Column number: 17
Token: ;                 Token Type: T_semi      Line number: 25         Column number: 20
Token: bool              Token Type: T_BOOL      Line number: 26         Column number: 0
Token: log_or            Token Type: T_ID        Line number: 26         Column number: 4
Token: =                 Token Type: T_eq        Line number: 26         Column number: 10
Token: op1               Token Type: T_ID        Line number: 26         Column number: 11
Token: ||                Token Type: T_or        Line number: 26         Column number: 14
Token: op2               Token Type: T_ID        Line number: 26         Column number: 16
Token: ;                 Token Type: T_semi      Line number: 26         Column number: 19
Token: int               Token Type: T_INT       Line number: 27         Column number: 0
Token: day               Token Type: T_ID        Line number: 27         Column number: 3
Token: =                 Token Type: T_eq        Line number: 27         Column number: 6
Token: 4                 Token Type: T_DIGIT     Line number: 27         Column number: 7
Token: ;                 Token Type: T_semi      Line number: 27         Column number: 8
Token: switch            Token Type: T_SWITCH    Line number: 28         Column number: 0
Token: (                 Token Type: T_lround    Line number: 28         Column number: 6
Token: day               Token Type: T_ID        Line number: 28         Column number: 7
Token: )                 Token Type: T_rround    Line number: 28         Column number: 10
Token: {                 Token Type: T_lflower   Line number: 29         Column number: 0
Token: CASE              Token Type: T_ID        Line number: 30         Column number: 0
Error at line 30
Token: 3                 Token Type: T_DIGIT     Line number: 30         Column number: 4
Token: :                 Token Type: T_col       Line number: 30         Column number: 5
Token: break             Token Type: T_BREAK     Line number: 30         Column number: 6
Token: ;                 Token Type: T_semi      Line number: 30         Column number: 11
Token: case              Token Type: T_CASE      Line number: 31         Column number: 0
Token: 4                 Token Type: T_DIGIT     Line number: 31         Column number: 4
Token: :                 Token Type: T_col       Line number: 31         Column number: 5
Token: cout              Token Type: T_COUT      Line number: 31         Column number: 6
Token: <                 Token Type: T_lt        Line number: 31         Column number: 10
Token: <                 Token Type: T_lt        Line number: 31         Column number: 11
Token: "CASE 4\n"                Token Type: T_STRINGLIT         Line number: 31         Column number: 12
Token: ;                 Token Type: T_semi      Line number: 31         Column number: 22
Token: }                 Token Type: T_rflower   Line number: 32         Column number: 0
Token: if                Token Type: T_IF        Line number: 33         Column number: 0
Input accepted.

SYMBOL TABLE


scope of table 1.1.

Name    Type    Line number value
s        string          8       "abc"
c        char    9       'c'
```

II. Semantic Analysis:

A) Throws error when a variable is used before declaration or if a variable is redeclared.

Input file:

```
#include<iostream>
#include<stdlib.h>
using namespace std;
a = 3; //error
int main()
{
	int a1[5];
	int op1 = 3 + 5 + 7;
	int op1 = 7.0; //error
}
```

Tokens:

```
(base) roshni@roshni-Latitude-7490:~/CD PROJECT$ ./a.out < semantic.cpp
Token: #include          Token Type: T_INCLUDE   Line number: 1          Column number: 0
Token: <                 Token Type: T_lt        Line number: 1          Column number: 8
Token: iostream                  Token Type: T_ID        Line number: 1          Column number: 9
Token: >                 Token Type: T_gt        Line number: 1          Column number: 17
Token: #include          Token Type: T_INCLUDE   Line number: 2          Column number: 0
Token: <                 Token Type: T_lt        Line number: 2          Column number: 8
Token: stdlib.h          Token Type: T_H         Line number: 2          Column number: 9
Token: >                 Token Type: T_gt        Line number: 2          Column number: 17
Token: using             Token Type: T_USING     Line number: 3          Column number: 0
Token: namespace         Token Type: T_NAMESPACE         Line number: 3          Column number: 5
Token: std               Token Type: T_ID        Line number: 3          Column number: 14
Token: ;                 Token Type: T_semi      Line number: 3          Column number: 17
Token: a                 Token Type: T_ID        Line number: 5          Column number: 0
Token: =                 Token Type: T_eq        Line number: 5          Column number: 1
Token: 3                 Token Type: T_DIGIT     Line number: 5          Column number: 2
Token: ;                 Token Type: T_semi      Line number: 5          Column number: 3
Error: Variable a used before declaring
Token: int               Token Type: T_INT       Line number: 7          Column number: 0
Token: main()            Token Type: T_MAINTOK   Line number: 7          Column number: 3
Token: {                 Token Type: T_lflower   Line number: 8          Column number: 0
Token: int               Token Type: T_INT       Line number: 9          Column number: 0
Token: a1                Token Type: T_ID        Line number: 9          Column number: 3
Token: [                 Token Type: T_lsq       Line number: 9          Column number: 5
Token: 5                 Token Type: T_DIGIT     Line number: 9          Column number: 6
Token: ]                 Token Type: T_rsq       Line number: 9          Column number: 7
Token: ;                 Token Type: T_semi      Line number: 9          Column number: 8
Token: int               Token Type: T_INT       Line number: 10         Column number: 0
Token: op1               Token Type: T_ID        Line number: 10         Column number: 3
Token: =                 Token Type: T_eq        Line number: 10         Column number: 6
Token: 3                 Token Type: T_DIGIT     Line number: 10         Column number: 7
Token: +                 Token Type: T_pl        Line number: 10         Column number: 8
Token: 5                 Token Type: T_DIGIT     Line number: 10         Column number: 9
Token: +                 Token Type: T_pl        Line number: 10         Column number: 10
Token: 7                 Token Type: T_DIGIT     Line number: 10         Column number: 11
Token: ;                 Token Type: T_semi      Line number: 10         Column number: 12
Token: int               Token Type: T_INT       Line number: 11         Column number: 0
Token: op1               Token Type: T_ID        Line number: 11         Column number: 3
Token: =                 Token Type: T_eq        Line number: 11         Column number: 6
Token: 7.0       Token Type: T_REAL      Line number: 11         Column number: 7
Token: ;                 Token Type: T_semi      Line number: 11         Column number: 10
Error: Variable op1 Redeclared
Token: }                 Token Type: T_rflower   Line number: 12         Column number: 0
Input accepted.
```

B) The datatype of each variable in the expression is checked to evaluate if they are compatible

Output: The code gives an error message containing incompatible operand and operator

For example:

Test Case 1: String used in expressions give appropriate errors

Input file:

```
#include<iostream>
#include<stdlib.h>
using namespace std;
int main()
{
      int a = "abc" + 7; //error
      int b = 6 + "def" + 8; //error
      float c = 7e4 && "ghi"; //error
      char d = 'c' >= "jkl"; //error
      float f = 3 && 2 + "pqr"; //error
      bool g = "stu" / 56 || 23.90; //error
}
```

```
(base) roshni@roshni-Latitude-7490:~/CD PROJECT$ ./a.out < type_check.cpp
Token: #include          Token Type: T_INCLUDE   Line number: 1          Column number: 0
Token: <                 Token Type: T_lt        Line number: 1          Column number: 8
Token: iostream                  Token Type: T_ID        Line number: 1          Column number: 9
Token: >                 Token Type: T_gt        Line number: 1          Column number: 17
Token: #include          Token Type: T_INCLUDE   Line number: 2          Column number: 0
Token: <                 Token Type: T_lt        Line number: 2          Column number: 8
Token: stdlib.h          Token Type: T_H         Line number: 2          Column number: 9
Token: >                 Token Type: T_gt        Line number: 2          Column number: 17
Token: using             Token Type: T_USING     Line number: 3          Column number: 0
Token: namespace         Token Type: T_NAMESPACE         Line number: 3          Column number: 5
Token: std               Token Type: T_ID        Line number: 3          Column number: 14
Token: ;                 Token Type: T_semi      Line number: 3          Column number: 17
Token: int               Token Type: T_INT       Line number: 5          Column number: 0
Token: main()            Token Type: T_MAINTOK   Line number: 5          Column number: 3
Token: {                 Token Type: T_lflower   Line number: 6          Column number: 0
Token: int               Token Type: T_INT       Line number: 7          Column number: 0
Token: a                 Token Type: T_ID        Line number: 7          Column number: 3
Token: =                 Token Type: T_eq        Line number: 7          Column number: 4
Token: "abc"             Token Type: T_STRINGLIT         Line number: 7          Column number: 5
Token: +                 Token Type: T_pl        Line number: 7          Column number: 10
Token: 7                 Token Type: T_DIGIT     Line number: 7          Column number: 11
Token: ;                 Token Type: T_semi      Line number: 7          Column number: 12
Error: Invalid operand: "abc" used for addition

Token: int               Token Type: T_INT       Line number: 8          Column number: 0
Token: b                 Token Type: T_ID        Line number: 8          Column number: 3
Token: =                 Token Type: T_eq        Line number: 8          Column number: 4
Token: 6                 Token Type: T_DIGIT     Line number: 8          Column number: 5
Token: +                 Token Type: T_pl        Line number: 8          Column number: 6
Token: "def"             Token Type: T_STRINGLIT         Line number: 8          Column number: 7
Token: +                 Token Type: T_pl        Line number: 8          Column number: 12
Error: Invalid operand: "def" used for addition

Token: 8                 Token Type: T_DIGIT     Line number: 8          Column number: 13
Token: ;                 Token Type: T_semi      Line number: 8          Column number: 14
Token: float             Token Type: T_FLOAT     Line number: 9          Column number: 0
Token: c                 Token Type: T_ID        Line number: 9          Column number: 5
Token: =                 Token Type: T_eq        Line number: 9          Column number: 6
Token: 7e4       Token Type: T_REAL      Line number: 9          Column number: 7
Token: &&                Token Type: T_and       Line number: 9          Column number: 10
```

```
Token: "ghi"            Token Type: T_STRINGLIT         Line number: 9          Column number: 12
Token: ;                Token Type: T_semi      Line number: 9          Column number: 17
Error: Invalid operand: "ghi" used for &&

Token: char             Token Type: T_CHAR      Line number: 10         Column number: 0
Token: d                Token Type: T_ID        Line number: 10         Column number: 4
Token: =                Token Type: T_eq        Line number: 10         Column number: 5
Token: 'c'              Token Type: T_CHARCONST         Line number: 10         Column number: 6
Token: >=               Token Type: T_gteq      Line number: 10         Column number: 9
Token: "jkl"            Token Type: T_STRINGLIT         Line number: 10         Column number: 11
Token: ;                Token Type: T_semi      Line number: 10         Column number: 16
Error: Invalid operand: "jkl" used for >=

Token: float            Token Type: T_FLOAT     Line number: 11         Column number: 0
Token: f                Token Type: T_ID        Line number: 11         Column number: 5
Token: =                Token Type: T_eq        Line number: 11         Column number: 6
Token: 3                Token Type: T_DIGIT     Line number: 11         Column number: 7
Token: &&               Token Type: T_and       Line number: 11         Column number: 8
Token: 2                Token Type: T_DIGIT     Line number: 11         Column number: 10
Token: +                Token Type: T_pl        Line number: 11         Column number: 11
Token: "pqr"            Token Type: T_STRINGLIT         Line number: 11         Column number: 12
Token: ;                Token Type: T_semi      Line number: 11         Column number: 17
Error: Invalid operand: "pqr" used for addition

Token: bool             Token Type: T_BOOL      Line number: 12         Column number: 0
Token: g                Token Type: T_ID        Line number: 12         Column number: 4
Token: =                Token Type: T_eq        Line number: 12         Column number: 5
Token: "stu"            Token Type: T_STRINGLIT         Line number: 12         Column number: 6
Token: /                Token Type: T_div       Line number: 12         Column number: 11
Token: 56               Token Type: T_DIGIT     Line number: 12         Column number: 12
Error: Invalid operand: "stu" used for division

Token: ||               Token Type: T_or        Line number: 12         Column number: 14
Token: 23.90     Token Type: T_REAL      Line number: 12         Column number: 16
Token: ;                Token Type: T_semi      Line number: 12         Column number: 21
Token: }                Token Type: T_rflower   Line number: 13         Column number: 0
Input accepted.

SYMBOL TABLE
(base) roshni@roshni-Latitude-7490:~/CD PROJECT$
```

Test Case 2: Correct expressions with different datatypes

Output: Successfully parsed with icg generated based on precedence and associativity of operations.

Input File:

```
#include<iostream>
#include<stdlib.h>
using namespace std;

int main()
{
        int a = true + 7;
        int b = 6 + 'c' + 8;
        float c = 7e4 && 3;
        char d = 'c' >= 56;
        float f = 3 && 2 + 45;
        bool g = false / 56 || 23.90;
}
```

```
(base) roshni@roshni-Latitude-7490:~/CD PROJECT$ ./a.out < type_check2.cpp
Token: #include          Token Type: T_INCLUDE   Line number: 1          Column number: 0
Token: <                 Token Type: T_lt        Line number: 1          Column number: 8
Token: iostream                  Token Type: T_ID        Line number: 1          Column number: 9
Token: >                 Token Type: T_gt        Line number: 1          Column number: 17
Token: #include          Token Type: T_INCLUDE   Line number: 2          Column number: 0
Token: <                 Token Type: T_lt        Line number: 2          Column number: 8
Token: stdlib.h          Token Type: T_H         Line number: 2          Column number: 9
Token: >                 Token Type: T_gt        Line number: 2          Column number: 17
Token: using             Token Type: T_USING     Line number: 3          Column number: 0
Token: namespace         Token Type: T_NAMESPACE         Line number: 3          Column number: 5
Token: std               Token Type: T_ID        Line number: 3          Column number: 14
Token: ;                 Token Type: T_semi      Line number: 3          Column number: 17
Token: int               Token Type: T_INT       Line number: 5          Column number: 0
Token: main()            Token Type: T_MAINTOK   Line number: 5          Column number: 3
Token: {                 Token Type: T_lflower   Line number: 6          Column number: 0
Token: int               Token Type: T_INT       Line number: 7          Column number: 0
Token: a                 Token Type: T_ID        Line number: 7          Column number: 3
Token: =                 Token Type: T_eq        Line number: 7          Column number: 4
Token: true              Token Type: T_TRUE      Line number: 7          Column number: 5
Token: +                 Token Type: T_pl        Line number: 7          Column number: 9
Token: 7                 Token Type: T_DIGIT     Line number: 7          Column number: 10
Token: ;                 Token Type: T_semi      Line number: 7          Column number: 11
Token: int               Token Type: T_INT       Line number: 8          Column number: 0
Token: b                 Token Type: T_ID        Line number: 8          Column number: 3
Token: =                 Token Type: T_eq        Line number: 8          Column number: 4
Token: 6                 Token Type: T_DIGIT     Line number: 8          Column number: 5
Token: +                 Token Type: T_pl        Line number: 8          Column number: 6
Token: 'c'               Token Type: T_CHARCONST         Line number: 8          Column number: 7
Token: +                 Token Type: T_pl        Line number: 8          Column number: 10
Token: 8                 Token Type: T_DIGIT     Line number: 8          Column number: 11
Token: ;                 Token Type: T_semi      Line number: 8          Column number: 12
Token: float             Token Type: T_FLOAT     Line number: 9          Column number: 0
Token: c                 Token Type: T_ID        Line number: 9          Column number: 5
Token: =                 Token Type: T_eq        Line number: 9          Column number: 6
Token: 7e4       Token Type: T_REAL      Line number: 9          Column number: 7
Token: &&                Token Type: T_and       Line number: 9          Column number: 10
Token: 3                 Token Type: T_DIGIT     Line number: 9          Column number: 12
Token: ;                 Token Type: T_semi      Line number: 9          Column number: 13
Token: char              Token Type: T_CHAR      Line number: 10         Column number: 0
Token: d                 Token Type: T_ID        Line number: 10         Column number: 4
Token: =                 Token Type: T_eq        Line number: 10         Column number: 5
Token: 'c'               Token Type: T_CHARCONST         Line number: 10         Column number: 6
Token: >=                Token Type: T_gteq      Line number: 10         Column number: 9
Token: 56                Token Type: T_DIGIT     Line number: 10         Column number: 11
Token: ;                 Token Type: T_semi      Line number: 10         Column number: 13
Token: float             Token Type: T_FLOAT     Line number: 11         Column number: 0
Token: f                 Token Type: T_ID        Line number: 11         Column number: 5
Token: =                 Token Type: T_eq        Line number: 11         Column number: 6
Token: 3                 Token Type: T_DIGIT     Line number: 11         Column number: 7
```

```
Token: &&                Token Type: T_and       Line number: 11         Column number: 8
Token: 2                 Token Type: T_DIGIT     Line number: 11         Column number: 10
Token: +                 Token Type: T_pl        Line number: 11         Column number: 11
Token: 45                Token Type: T_DIGIT     Line number: 11         Column number: 12
Token: ;                 Token Type: T_semi      Line number: 11         Column number: 14
Token: bool              Token Type: T_BOOL      Line number: 12         Column number: 0
Token: g                 Token Type: T_ID        Line number: 12         Column number: 4
Token: =                 Token Type: T_eq        Line number: 12         Column number: 5
Token: false             Token Type: T_FALSE     Line number: 12         Column number: 6
Token: /                 Token Type: T_div       Line number: 12         Column number: 11
Token: 56                Token Type: T_DIGIT     Line number: 12         Column number: 12
Token: ||                Token Type: T_or        Line number: 12         Column number: 14
Token: 23.90     Token Type: T_REAL      Line number: 12         Column number: 16
Token: ;                 Token Type: T_semi      Line number: 12         Column number: 21
Token: }                 Token Type: T_rflower   Line number: 13         Column number: 0
Input accepted.

SYMBOL TABLE


scope of table 1.1.

Name    Type    Line number value
t1       int     7
a        int     7       t1
t2       int     8
t3       int     8
b        int     8       t3
t4       float   9
c        float   9       t4
t5       int     10
d        char    10      t5
t6       int     11
t7       int     11
f        float   11      t7
t8       int     12
t9       float   12
g        bool    12      t9
(base) roshni@roshni-Latitude-7490:~/CD PROJECT$
```

```
(base) roshni@roshni-Latitude-7490:~/CD PROJECT$ cat icg.txt
main{
t1 = true + 7
a = t1
t2 = 6 + 'c'
t3 = t2 + 8
b = t3
t4 = 7e4 && 3
c = t4
t5 = 'c' >= 56
d = t5
t6 = 2 + 45
t7 = 3 && t6
f = t7
t8 = false / 56
t9 = t8 || 23.90
g = t9
}
(base) roshni@roshni-Latitude-7490:~/CD PROJECT$
```

C) Check if the type of the RHS can be cast to the datatype of LHS

Output: Throws error if incorrect casting is present

```
#include<iostream>
#include<stdlib.h>
using namespace std;

int main()
{
        bool d = "def";
        int a = "abc"; //error
        float b = "ghi"; //error
        char c = "jkl"; //error
        string d = false; //error
        string e = 3.50; //error
}
```

```
(base) roshni@roshni-Latitude-7490:~/CD PROJECT$ ./a.out < type_check3.cpp
Token: #include          Token Type: T_INCLUDE   Line number: 1          Column number: 0
Token: <                 Token Type: T_lt        Line number: 1          Column number: 8
Token: iostream                Token Type: T_ID        Line number: 1          Column number: 9
Token: >                 Token Type: T_gt        Line number: 1          Column number: 17
Token: #include          Token Type: T_INCLUDE   Line number: 2          Column number: 0
Token: <                 Token Type: T_lt        Line number: 2          Column number: 8
Token: stdlib.h          Token Type: T_H         Line number: 2          Column number: 9
Token: >                 Token Type: T_gt        Line number: 2          Column number: 17
Token: using             Token Type: T_USING     Line number: 3          Column number: 0
Token: namespace         Token Type: T_NAMESPACE         Line number: 3          Column number: 5
Token: std               Token Type: T_ID        Line number: 3          Column number: 14
Token: ;                 Token Type: T_semi      Line number: 3          Column number: 17
Token: int               Token Type: T_INT       Line number: 5          Column number: 0
Token: main()            Token Type: T_MAINTOK   Line number: 5          Column number: 3
Token: {                 Token Type: T_lflower   Line number: 6          Column number: 0
Token: bool              Token Type: T_BOOL      Line number: 7          Column number: 0
Token: d                 Token Type: T_ID        Line number: 7          Column number: 4
Token: =                 Token Type: T_eq        Line number: 7          Column number: 5
Token: "def"             Token Type: T_STRINGLIT         Line number: 7          Column number: 6
Token: ;                 Token Type: T_semi      Line number: 7          Column number: 11
Token: int               Token Type: T_INT       Line number: 8          Column number: 0
Token: a                 Token Type: T_ID        Line number: 8          Column number: 3
Token: =                 Token Type: T_eq        Line number: 8          Column number: 4
Token: "abc"             Token Type: T_STRINGLIT         Line number: 8          Column number: 5
Token: ;                 Token Type: T_semi      Line number: 8          Column number: 10
Error: The given expression cannot be casted to int

Token: float             Token Type: T_FLOAT     Line number: 9          Column number: 0
Token: b                 Token Type: T_ID        Line number: 9          Column number: 5
Token: =                 Token Type: T_eq        Line number: 9          Column number: 6
Token: "ghi"             Token Type: T_STRINGLIT         Line number: 9          Column number: 7
Token: ;                 Token Type: T_semi      Line number: 9          Column number: 12
Error: The given expression cannot be casted to float

Token: char              Token Type: T_CHAR      Line number: 10         Column number: 0
Token: c                 Token Type: T_ID        Line number: 10         Column number: 4
Token: =                 Token Type: T_eq        Line number: 10         Column number: 5
Token: "jkl"             Token Type: T_STRINGLIT         Line number: 10         Column number: 6
Token: ;                 Token Type: T_semi      Line number: 10         Column number: 11
Error: The given expression cannot be casted to char

Token: string            Token Type: T_STRING    Line number: 11         Column number: 0
Token: d                 Token Type: T_ID        Line number: 11         Column number: 6
Token: =                 Token Type: T_eq        Line number: 11         Column number: 7
Token: false             Token Type: T_FALSE     Line number: 11         Column number: 8
Token: ;                 Token Type: T_semi      Line number: 11         Column number: 13
Error: The given expression cannot be casted to string

Token: string            Token Type: T_STRING    Line number: 12         Column number: 0
```

```
Token: e                Token Type: T_ID        Line number: 12         Column number: 6
Token: =                Token Type: T_eq        Line number: 12         Column number: 7
Token: 3.50      Token Type: T_REAL      Line number: 12         Column number: 8
Token: ;                Token Type: T_semi      Line number: 12         Column number: 12
Error: The given expression cannot be casted to string

Token: }                Token Type: T_rflower   Line number: 13         Column number: 0
Input accepted.

SYMBOL TABLE


scope of table 1.1.

Name    Type    Line number value
d        bool    7       "def"
(base) roshni@roshni-Latitude-7490:~/CD PROJECT$
```

3. Symbol table

Input file: Code.cpp

```
#include<iostream>
#include<stdlib.h>
using namespace std;
float a = 10;
int func()
{
    int a = 5;
    int c = a + 3 * 5 / 10;
}
class class1
{
   private: int b;
   public:
       int class_function(int d)
           {
               cout<<"In function "<<d<<endl;
           }
};
int main()
{
   class1 obj1;
   obj1.class_function(10);
       int a1[5]={1,2,3,4,5};
       int b1[]={1,2,3,4,5};
       int a = 9;
       int b = 10;

       int add = 3 + 5 + 5 + 8;
       int sub = 3 - 5;
       int op1 = 5;
```

```
        int op2 = 10;
        bool gt = op1 > op2;
        bool lteq = op1 <= op2;
        bool log_and = op1 && op2;
        bool log_or = op1 || op2;
        if(a <= b)
        {
        int day = 4;
        switch (day)
                {
                    case 1: {
                                int case1 = 4;
                                break;
                            }
                    case 2:
                    case 3: break;
                    case 4: cout << "CASE 4\n";
        }

        }
}
```

```
SYMBOL TABLE

scope of table 1.

Name     Type     Line number value
a         float    5        10
func      int_func          6        A_FUNCTION
class1    class    12       A_CLASS

scope of table 1.1.

Name     Type     Line number value
a         int      8        5
t1        int      9
t2        int      9
t3        int      9
c         int      9        t3

scope of table 1.2.

Name     Type     Line number value
b         int      14       NOT_DEFINED
class_function    int_func         16       A_FUNCTION

scope of table 1.2.1.

Name     Type     Line number value
d         int      16       NOT_DEFINED

scope of table 1.3.

Name     Type     Line number value
obj1      class1_obj        24       AN_OBJECT
a1        int_arr           26       NOT_DEFINED
b1        int_arr           27       NOT_DEFINED
a         int      28       9
b         int      29       10
t4        int      31
t5        int      31
t6        int      31
add       int      31       t6
t7        int      32
sub       int      32       t7
op1       int      33       5
op2       int      34       10
```

```
t8          int      35
gt          bool     35        t8
t9          int      36
lteq        bool     36        t9
t10         int      37
log_and              bool      37        t10
t11         int      38
log_or      bool     38        t11


scope of table 1.3.2.

Name     Type     Line number value
day       int      42        4


scope of table 1.3.2.1.1.

Name     Type     Line number value
case1     int      46        4
(base) roshni@roshni-Latitude-7490:~/CD PROJECT$
```

4. Intermediate Code Generation

Input file : Code.cpp from above section (Symbol table)

```
(base) roshni@roshni-Latitude-7490:~/CD PROJECT$ cat icg.txt
a = 10
func{
a = 5
t1 = 3 * 5
t2 = t1 / 10
t3 = a + t2
c = t3
}
class1{
class1.class_function{
PARAM "INVOKED CLASS FUNCTION WITH VALUE ="
PARAM d
PARAM endl
call ( cout , 3 )
}
main{
PARAM 10
call ( obj1.class_function , 1 )
a1[0] = 1
a1[4] = 2
a1[8] = 3
a1[12] = 4
a1[16] = 5
b1[0] = 1
b1[4] = 2
b1[8] = 3
b1[12] = 4
b1[16] = 5
a = 9
b = 10
t4 = 3 + 5
t5 = t4 + 5
t6 = t5 + 8
add = t6
t7 = 3 - 5
sub = t7
op1 = 5
op2 = 10
t8 = op1 > op2
gt = t8
t9 = op1 <= op2
lteq = t9
t10 = op1 && op2
log_and = t10
t11 = op1 || op2
log_or = t11
t12 = a <= b
if t12 goto L1
goto L2
```

```
L1 :
day = 4
t13 = day == 1
if t13 goto L3
goto L4
L3 :
case1 = 4
L4 :
t14 = day == 2
if t14 goto L5
goto L6
L5 :
L6 :
t15 = day == 3
if t15 goto L7
goto L8
L7 :
call ( break , 0 )
L8 :
t16 = day == 4
if t16 goto L9
goto L10
L9 :
PARAM "CASE 4\n"
call ( cout , 1 )
L10 :
L2 :
}
(base) roshni@roshni-Latitude-7490:~/CD PROJECT$
```

5.Code optimisation

Input code -

```
#include<iostream>
#include<stdlib.h>

using namespace std;

float a = 10;
float b = 56.0;
int s = 4;
float d = 20.0;
float v = 31;
int p = 2;

int mr = 10 * 2;
int dr = 10 / 2;
d = 18 * b;
mr = mr + 0 - dr;
int y = mr + dr;

int main()
{
if (b > s)
{
  cout<<"This variable is live ="<<d<<endl;
  cout<<"This variable is live ="<<mr<<endl;
  return ;
  cout<<"This line to be removed in dead code elimination="<<v<<endl;
}
}
```

Input file - icg [Displayed on left side of the figure]
Output file - optimised code[Displayed on right side of the figure]

Live variable analysis

```
b = 56.0
s = 4
d = 20.0
p = 2
t1 = 10 * 2
mr = t1
t2 = 10 / 2
dr = t2
t3 = 18 * b
d = t3
t4 = mr + 0
t5 = t4 - dr
mr = t5
t6 = mr + dr
y = t6
main{
t7 = b > s
if t7 goto L1
goto L2
L1 :
PARAM "This variable is live ="
PARAM d
PARAM endl
call ( cout , 3 )
PARAM "This variable is live ="
PARAM mr
PARAM endl
call ( cout , 3 )
call ( return , 0 )
PARAM "This variable is live ="
PARAM v
PARAM endl
call ( cout , 3 )
L2 :
}
```

```
b = 56.0
s = 4
d = 20.0
t1 = 10 * 2
mr = t1
t2 = 10 / 2
dr = t2
t3 = 18 * b
d = t3
t4 = mr + 0
t5 = t4 - dr
mr = t5
main{
t7 = b > s
if t7 goto L1
goto L2
L1 :
PARAM "This variable is live ="
PARAM d
PARAM endl
call ( cout , 3 )
PARAM "This variable is live ="
PARAM mr
PARAM endl
call ( cout , 3 )
call ( return , 0 )
L2 :
}
```

Dead code Elimination

```
b = 56.0
s = 4
d = 20.0
p = 2
t1 = 10 * 2
mr = t1
t2 = 10 / 2
dr = t2
t3 = 18 * b
d = t3
t4 = mr + 0
t5 = t4 - dr
mr = t5
t6 = mr + dr
y = t6
main{
t7 = b > s
if t7 goto L1
goto L2
L1 :
PARAM "This variable is live ="
PARAM d
PARAM endl
call ( cout , 3 )
PARAM "This variable is live ="
PARAM mr
PARAM endl
call ( cout , 3 )
call ( return , 0 )
PARAM "This line to be removed in dead code elimination="
PARAM v
PARAM endl
call ( cout , 3 )
L2 :
}
```

```
b = 56.0
s = 4
d = 20.0
t1 = 10 * 2
mr = t1
t2 = 10 / 2
dr = t2
t3 = 18 * b
d = t3
t4 = mr + 0
t5 = t4 - dr
mr = t5
main{
t7 = b > s
if t7 goto L1
goto L2
L1 :
PARAM "This variable is live ="
PARAM d
PARAM endl
call ( cout , 3 )
PARAM "This variable is live ="
PARAM mr
PARAM endl
call ( cout , 3 )
call ( return , 0 )
L2 :
}
```

Strength Reduction

```
p = 2
t1 = 10 * 2
mr = t1
t2 = 10 / 2
dr = t2
```

```
t1 = 10 + 10
mr = t1
t2 = 10 >> 1
dr = t2
t3 = 18 * b
```

Constant folding and Propagation

```
b = 56.0
s = 4
d = 20.0
p = 2
t1 = 10 * 2
mr = t1
t2 = 10 / 2
dr = t2
t3 = 18 * b
d = t3
t4 = mr + 0
t5 = t4 - dr
mr = t5
t6 = mr + dr
y = t6
main{
t7 = b > s
if t7 goto L1
goto L2
L1 :
PARAM "This variable is live ="
PARAM d
PARAM endl
call ( cout , 3 )
PARAM "This variable is live ="
PARAM mr
PARAM endl
call ( cout , 3 )
call ( return , 0 )
PARAM "This line to be removed in dead code elimination="
PARAM v
PARAM endl
call ( cout , 3 )
L2 :
}
```

```
b = 56.0
s = 4
d = 20.0
t1 = 20
mr = 20
t2 = 5
dr = 5
t3 = 1008.0
d = 1008.0
t4 = 20
t5 = 15
mr = 15
main{
t7 = True
if t7 goto L1
goto L2
L1 :
PARAM "This variable is live ="
PARAM d
PARAM endl
call ( cout , 3 )
PARAM "This variable is live ="
PARAM mr
PARAM endl
call ( cout , 3 )
call ( return , 0 )
L2 :
}
```

9. CONCLUSIONS

We can conclude that a satisfactorily accurate compiler can be built using Lex and Yacc for various programming languages. Each phase of the compiler needs to be built in a step by step thorough manner, by following all regulations and using the tools mentioned, in order to reach good accuracy in the compilation process.We have learned that grammar of a language is the fundamental core that defines how good the compiler will be. A thorough non ambiguous grammar removes any conflicts that occur. The symbol table for a compiler can be built using any data structure and it must include scope. Semantic analysis and error

handling/reporting is essential for the effectiveness of the compilation process. TAC in Quadruple format is an effective intermediate code.

We conclude that we can use Lex and Yacc to build a standard compiler for almost any language.

10. FURTHER ENHANCEMENTS

1. Different structures for the different types of tokens and a union of these structures - this would significantly reduce complexity of semantic analysis and expression evaluation as all variables will be read in the format of their respective initialised types and there is no need for any casting.
2. This way, memory will be properly utilized.
3. The compiler is a mini-compiler and doesn't entirely mimic or compile all C++ code.
4. Implementation of all Object Oriented concepts such as polymorphism and namespaces are yet to be done.
5. The generated code is optimised to a great extent, it follows only the 4 mentioned optimisations are performed.

REFERENCES/BIBLIOGRAPHY

1. https://steemit.com/programming/@drifter1/writing-a-simple-compiler-on-my-own-lexical-analysis-using-flex
2. https://visualstudiomagazine.com/articles/2014/05/01/how-to-write-your-own-compiler-part-1.aspx
3. https://www.wisdomjobs.com/e-university/compiler-design-tutorial-1144/compiler-design-symbol-table-25307.html
4. https://isocpp.org/wiki/faq/compiler-dependencies#free-cpp-compiler
5. http://dinosaur.compilertools.net/
6. https://developers.redhat.com/blog/2018/03/21/compiler-and-linker-flags-gcc/
7. https://www.geeksforgeeks.org/intermediate-code-generation-in-compiler-design/
8. https://www.geeksforgeeks.org/code-optimization-in-compiler-design/

Link to github repo : https://github.com/kavyakpk25/Mini-Compiler