PES
UNIVERSITY

Report on

“Mini C++ Compiler with if, if-else, while and for constructs”

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

Compiler Design (UE18CS351)

Bachelor of Technology
in
Computer Science & Engineering

Submitted by:

Devika S Nair	**PES1201800372**
Mitravinda K M	**PES1201801872**
Prathima B	**PES1201801889**

Under the guidance of

Prof. Preet Kanwal
Associate Professor
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

1.INTRODUCTION

The project is a mini-compiler designed for the C++ language and covers:

- Lexical analysis
- Semantic analysis
- Symbol table creation
- Expression evaluation
- Intermediate code generation
- Optimization

The main tools used in the project include LEX which identifies pre-defined patterns and generates tokens for the patterns matched and YACC which parses the input for semantic meaning and generates an abstract syntax tree and intermediate code for the source code.
C++ is used to optimize the intermediate code generated by the
parser.

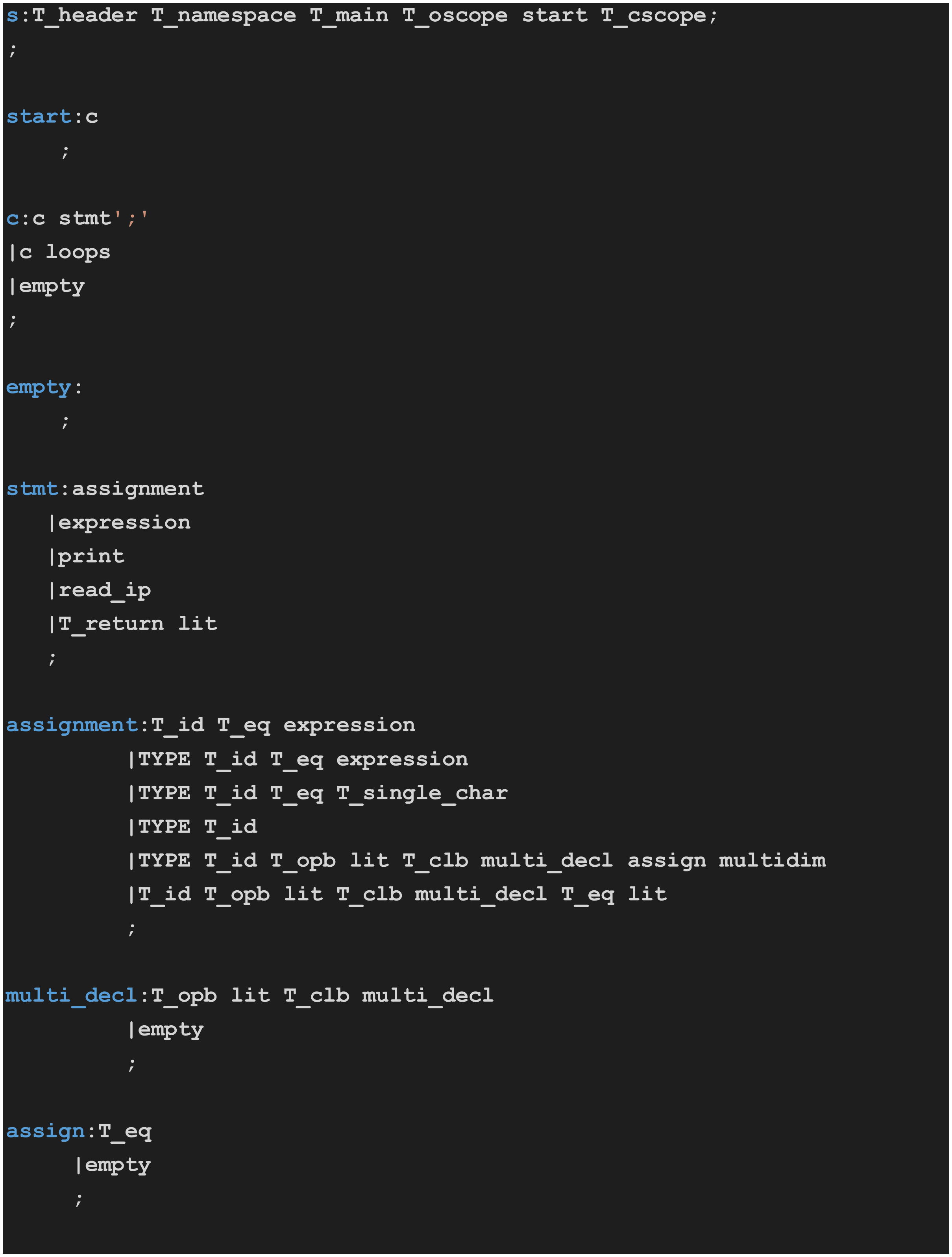
2.ARCHITECTURE OF LANGUAGE:

The major C++ constructs implemented are:

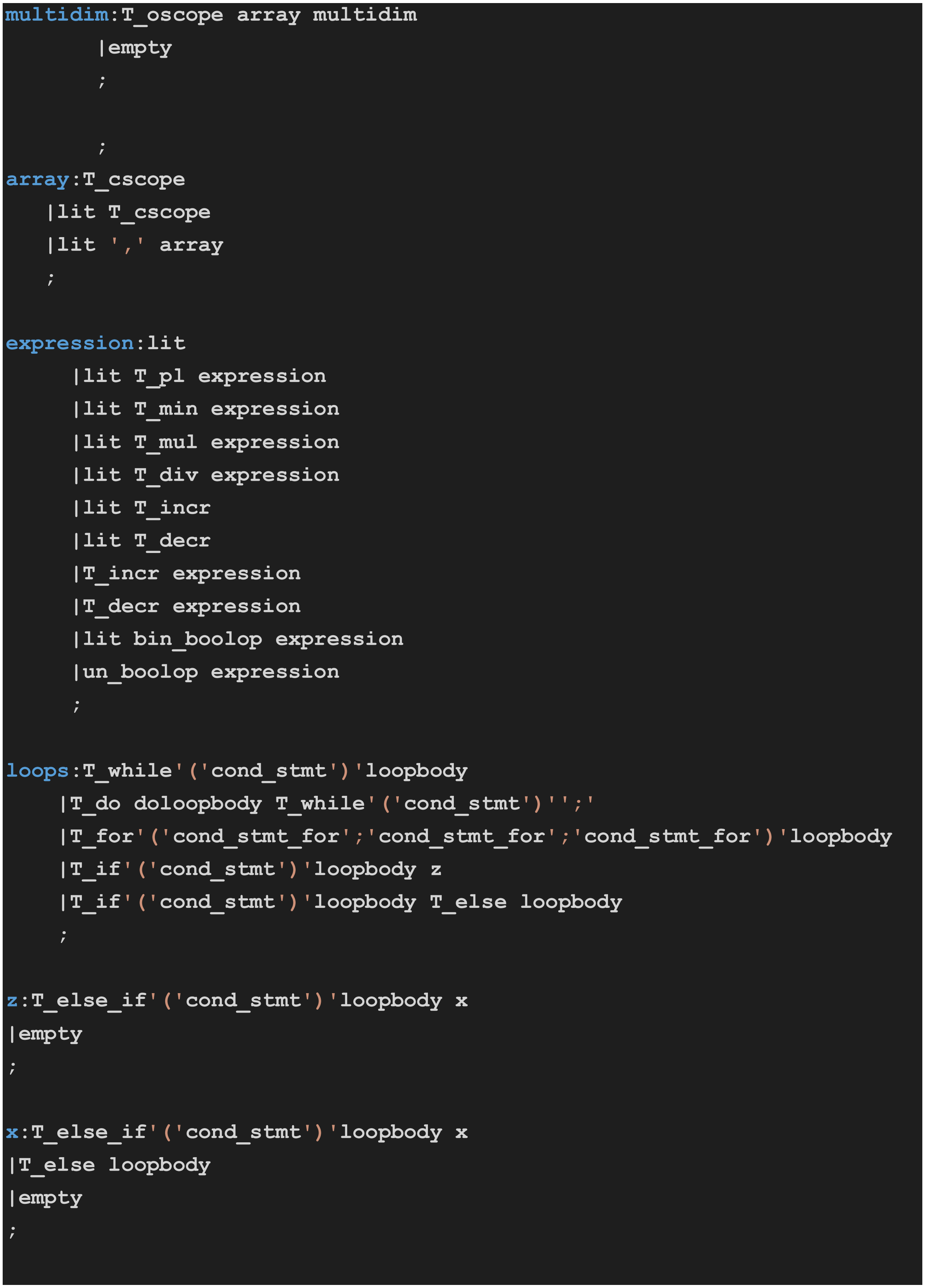
- expressions and conditions
- if clause
- if-else clause
- else-if clauses
- for loops
- while loops

This Mini-Compiler performs syntax analysis to verify if the sequence of tokens in the input forms a valid sentence based on the definition of the C++ grammar provided in the yacc file. It checks for the semantic validity of the given input. Appropriate rules are written to validate type checking, to ensure that all variables are declared before use and to catch any variable redeclarations. It also catches any bracket mismatch and correctly records the scope of each variable.

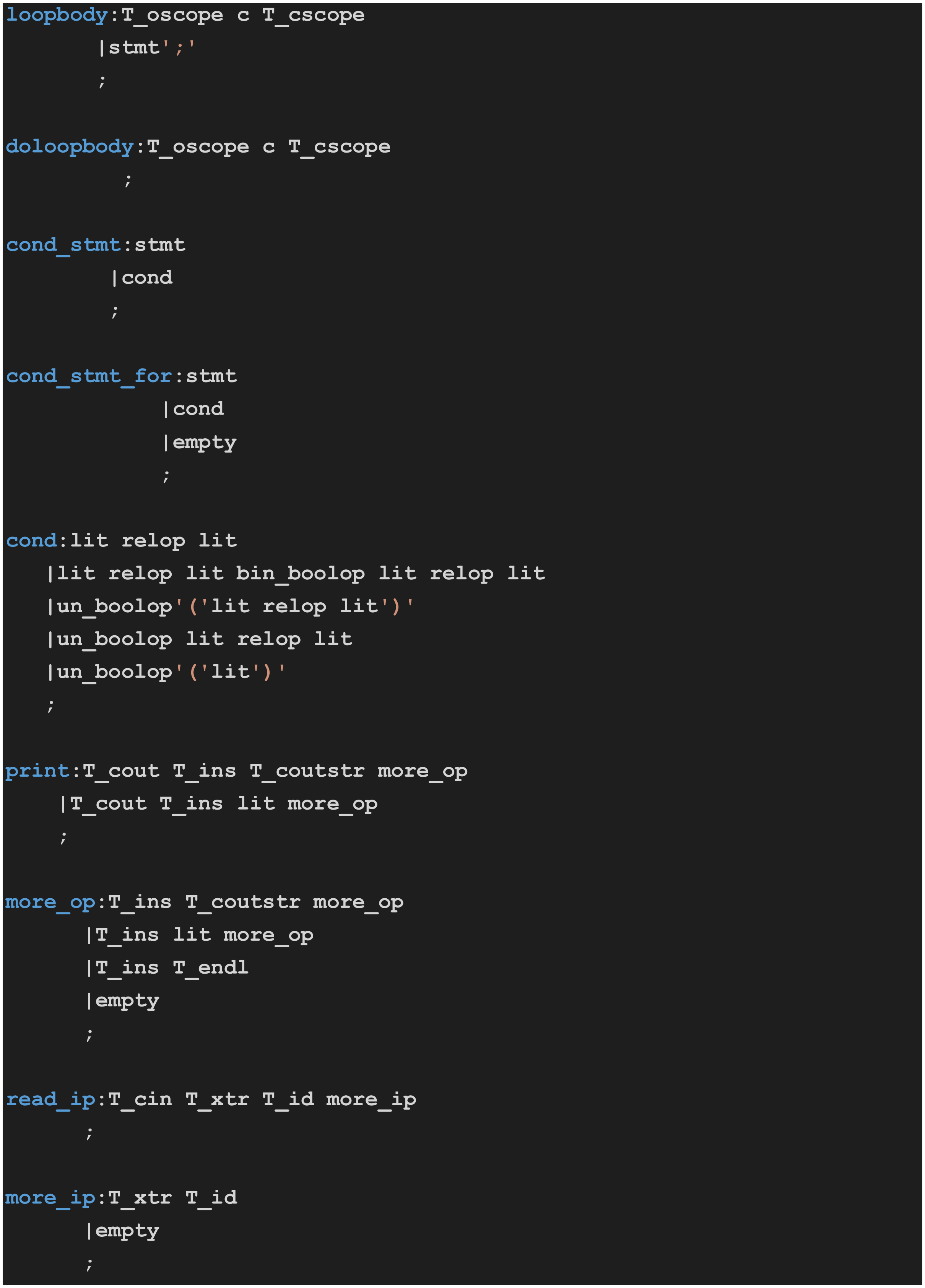
3.CONTEXT FREE GRAMMAR



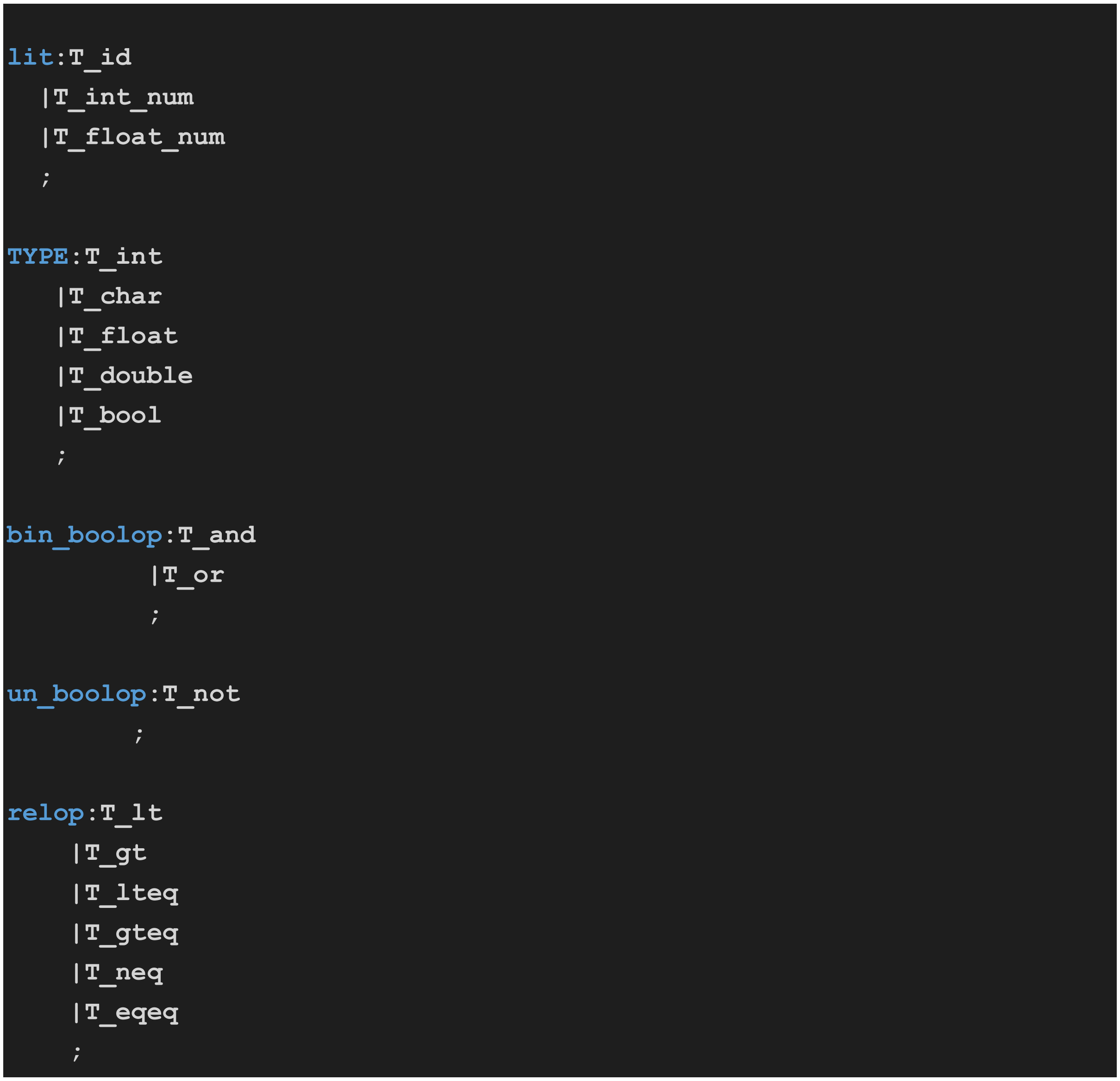
```
s:T_header T_namespace T_main T_oscope start T_cscope;
;

start:c
    ;

c:c stmt';'
|c loops
|empty
;

empty:
    ;

stmt:assignment
   |expression
   |print
   |read_ip
   |T_return lit
   ;

assignment:T_id T_eq expression
         |TYPE T_id T_eq expression
         |TYPE T_id T_eq T_single_char
         |TYPE T_id
         |TYPE T_id T_opb lit T_clb multi_decl assign multidim
         |T_id T_opb lit T_clb multi_decl T_eq lit
         ;

multi_decl:T_opb lit T_clb multi_decl
         |empty
         ;

assign:T_eq
     |empty
     ;
```



```
multidim:T_oscope array multidim
       |empty
       ;

       ;
array:T_cscope
   |lit T_cscope
   |lit ',' array
   ;

expression:lit
     |lit T_pl expression
     |lit T_min expression
     |lit T_mul expression
     |lit T_div expression
     |lit T_incr
     |lit T_decr
     |T_incr expression
     |T_decr expression
     |lit bin_boolop expression
     |un_boolop expression
     ;

loops:T_while'('cond_stmt')'loopbody
    |T_do doloopbody T_while'('cond_stmt')'';'
    |T_for'('cond_stmt_for';'cond_stmt_for';'cond_stmt_for')'loopbody
    |T_if'('cond_stmt')'loopbody z
    |T_if'('cond_stmt')'loopbody T_else loopbody
    ;

z:T_else_if'('cond_stmt')'loopbody x
|empty
;

x:T_else_if'('cond_stmt')'loopbody x
|T_else loopbody
|empty
;
```



```
loopbody:T_oscope c T_cscope
       |stmt';'
       ;

doloopbody:T_oscope c T_cscope
         ;

cond_stmt:stmt
        |cond
        ;

cond_stmt_for:stmt
            |cond
            |empty
            ;

cond:lit relop lit
   |lit relop lit bin_boolop lit relop lit
   |un_boolop'('lit relop lit')'
   |un_boolop lit relop lit
   |un_boolop'('lit')'
   ;

print:T_cout T_ins T_coutstr more_op
    |T_cout T_ins lit more_op
    ;

more_op:T_ins T_coutstr more_op
      |T_ins lit more_op
      |T_ins T_endl
      |empty
      ;

read_ip:T_cin T_xtr T_id more_ip
      ;

more_ip:T_xtr T_id
      |empty
      ;
```



```
lit:T_id
  |T_int_num
  |T_float_num
  ;

TYPE:T_int
   |T_char
   |T_float
   |T_double
   |T_bool
   ;

bin_boolop:T_and
         |T_or
         ;

un_boolop:T_not
        ;

relop:T_lt
    |T_gt
    |T_lteq
    |T_gteq
    |T_neq
    |T_eqeq
    ;
```

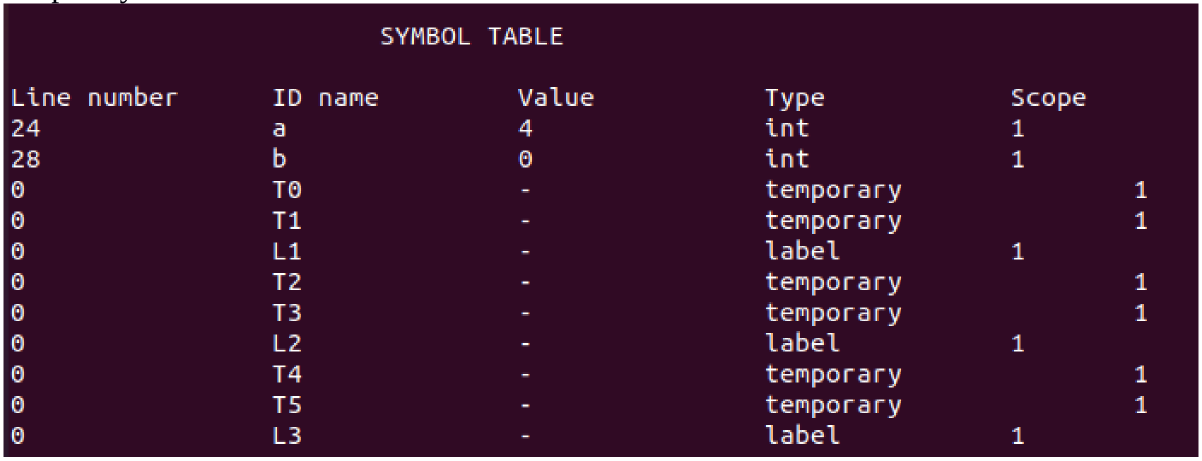
4.DESIGN STRATEGY

- **SYMBOL TABLE CREATION**

The symbol table is necessary to keep a log of the various identifiers, their values, types, scopes and their region of occurrence. The symbol table has functionalities to

- check if a node already exists
- update values of nodes
- create entries

Further basic warnings are given in case of redeclaration or undeclared variable initialization. The symbol table also becomes crucial in the Intermediate code generation phase where temporary variables and labels need to be added.



```
                        SYMBOL TABLE

Line number      ID name         Value           Type            Scope
24               a               4               int             1
28               b               0               int             1
0                T0              -               temporary                1
0                T1              -               temporary                1
0                L1              -               label           1
0                T2              -               temporary                1
0                T3              -               temporary                1
0                L2              -               label           1
0                T4              -               temporary                1
0                T5              -               temporary                1
0                L3              -               label           1
```

- **INTERMEDIATE CODE GENERATION**

The intermediate code generation phase receives the input from the semantic analyzer and develops a three address code. Statements in the three address code have no more than 3 references.

For Example, the code *a=b*c* will be converted as follows:

*T0=b*c;*
a=T0;

Where T0 represents a temporary variable. This three address code is machine independent. Quadruples are used to store and represent the three address code. Quadruples have 4 different columns

- Operation
- Argument 1
- Argument 2
- Result

For the above Three Address Code, the entries in quadruples will look like so:

op	arg1	arg2	result
*	b	c	T0
=	T0	(null)	a

The Quadruples are sent to the code optimization phase as input.

- **CODE OPTIMIZATION**

The code optimization is performed on the Three address code in quadruple format
The different code optimization techniques implemented in the project are:

- Elimination dead code/unreachable code
  - The statements below return statement are dead
  - The expressions/variable definitions are eliminated if they are not used before their next definition.
- Common subexpression elimination
  - If 2 statements are having same subexpressions i.e same arg1 and arg2 in quadruple form, then the second statement is removed and the variable that is assigned to that subexpression is replaced with that of the first statement, wherever it is used.
- Constant folding and Constant propagation
  - Replacing the variables with constants.
  - And then evaluation of expressions when both the arguments in quadruple form are constants.
- Copy propagation
  - If a variable is copied to another variable, then we can replace one with the other wherever they are used.

- **ERROR HANDLING**

While explicit error handling has not been implemented, warnings are issues during variable redeclaration or undeclared initializations.

5.IMPLEMENTATION DETAILS

- **SYMBOL TABLE**

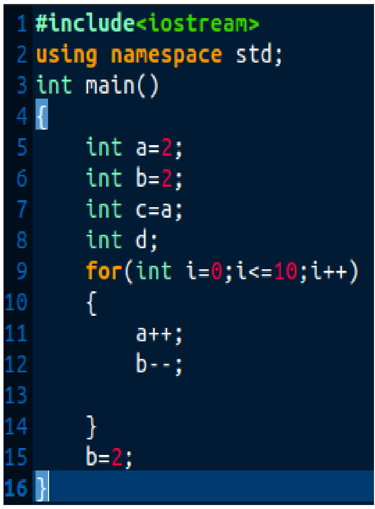
The data structure we have used to implement symbol table is an array of structures
The structure is called node and has the attributes like line number, scope of the variable, name of the variable, value of the variable and type of the variable

```
typedef struct node
{
   int line;
   int scope;
   char* name;
   char* value;
   char* type;
} node;
```

- **INTERMEDIATE CODE GENERATION**

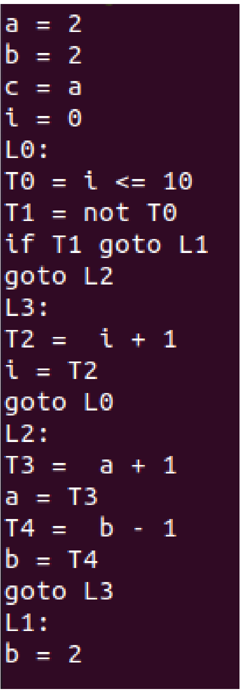
The intermediate code generation phase has two phases- TAC, Quadruples

- Conversion to Three address code
    - Different methods are used for different kinds of statements such as assignment, condition, if-else and loops to determine the following:
        - creation of temporaries
        - creation of labels
        - goto statements
    - These functions use stack data structures to keep track of label numbers and temporaries
    - This is essential to make sure that the flow of control is as expected especially for branching
    - In addition to creation of temporaries and labels care is also taken to ensure that they are updated in the symbol table.
- Examples for three address code generation
    - For loops:



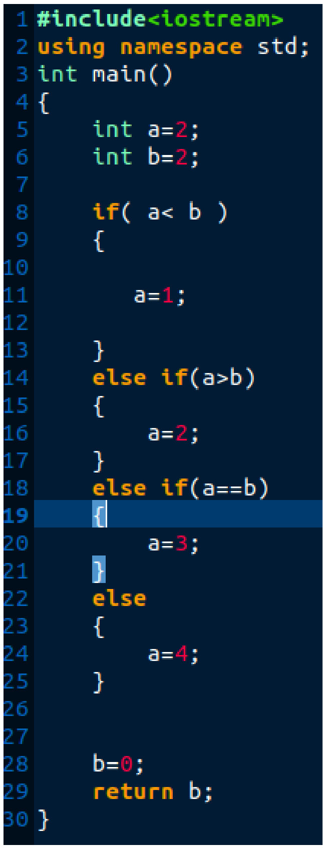
```
#include<iostream>
using namespace std;
int main()
{
    int a=2;
    int b=2;
    int c=a;
    int d;
    for(int i=0;i<=10;i++)
    {
        a++;
        b--;

    }
    b=2;
}
```

----------->



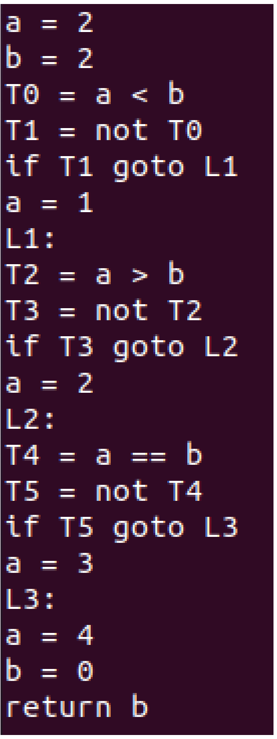
```
a = 2
b = 2
c = a
i = 0
L0:
T0 = i <= 10
T1 = not T0
if T1 goto L1
goto L2
L3:
T2 =  i + 1
i = T2
goto L0
L2:
T3 =  a + 1
a = T3
T4 =  b - 1
b = T4
goto L3
L1:
b = 2
```

- if-else:



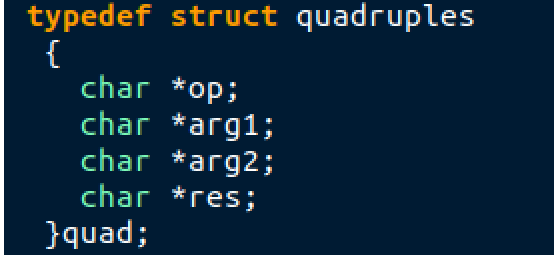
```
#include<iostream>
using namespace std;
int main()
{
    int a=2;
    int b=2;

    if( a< b )
    {

       a=1;

    }
    else if(a>b)
    {
        a=2;
    }
    else if(a==b)
    {
        a=3;
    }
    else
    {
        a=4;
    }


    b=0;
    return b;
}
```

--------->



```
a = 2
b = 2
T0 = a < b
T1 = not T0
if T1 goto L1
a = 1
L1:
T2 = a > b
T3 = not T2
if T3 goto L2
a = 2
L2:
T4 = a == b
T5 = not T4
if T5 goto L3
a = 3
L3:
a = 4
b = 0
return b
```

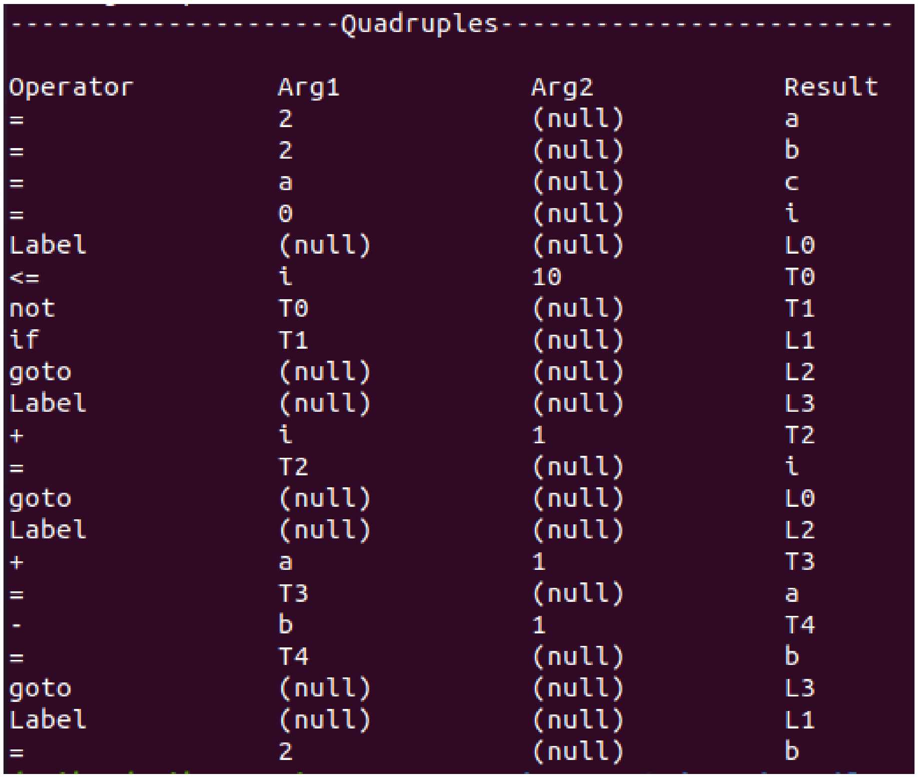
- Generating quadruples
    - Quadruples are stored using a structure having 4 fields



```
typedef struct quadruples
 {
   char *op;
   char *arg1;
   char *arg2;
   char *res;
 }quad;
```

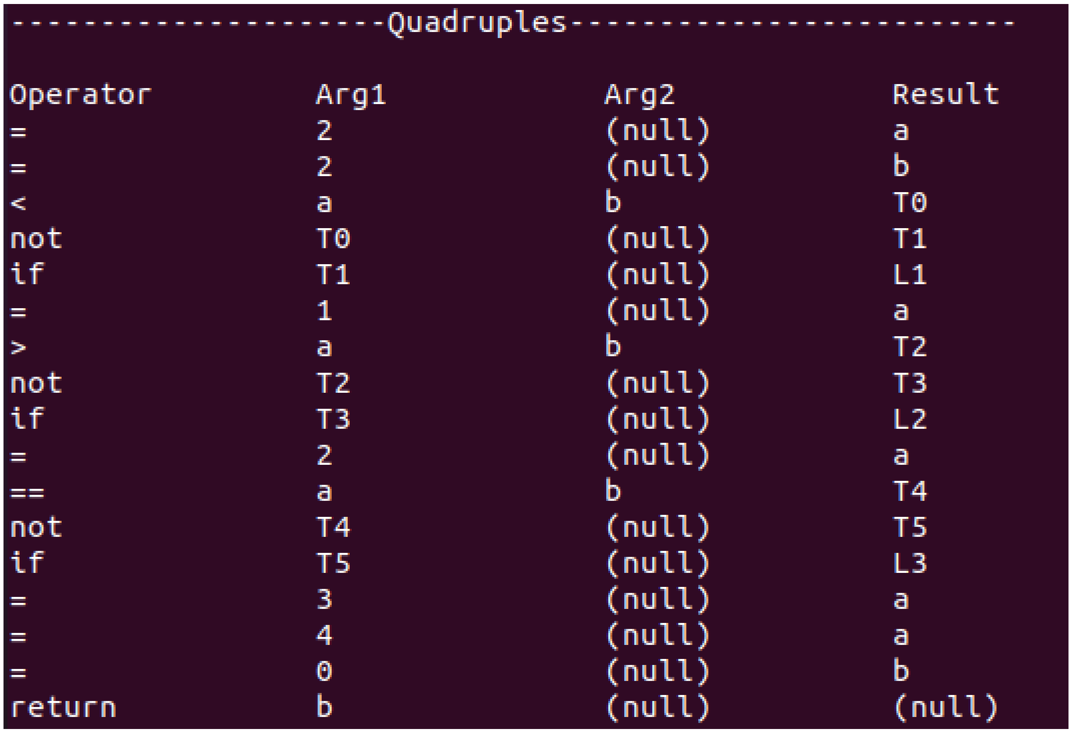
    - Using stack elements, yylval and other storage variables, the fields are updated and printed sequentially after parsing is complete

- Examples for quadruples:
    - if-else



```
--------------------Quadruples-----------------------

Operator          Arg1            Arg2            Result
=                 2               (null)          a
=                 2               (null)          b
=                 a               (null)          c
=                 0               (null)          i
Label             (null)          (null)          L0
<=                i               10              T0
not               T0              (null)          T1
if                T1              (null)          L1
goto              (null)          (null)          L2
Label             (null)          (null)          L3
+                 i               1               T2
=                 T2              (null)          i
goto              (null)          (null)          L0
Label             (null)          (null)          L2
+                 a               1               T3
=                 T3              (null)          a
-                 b               1               T4
=                 T4              (null)          b
goto              (null)          (null)          L3
Label             (null)          (null)          L1
=                 2               (null)          b
```

-
- if-else:



```
--------------------Quadruples-----------------------

Operator          Arg1            Arg2            Result
=                 2               (null)          a
=                 2               (null)          b
<                 a               b               T0
not               T0              (null)          T1
if                T1              (null)          L1
=                 1               (null)          a
>                 a               b               T2
not               T2              (null)          T3
if                T3              (null)          L2
=                 2               (null)          a
==                a               b               T4
not               T4              (null)          T5
if                T5              (null)          L3
=                 3               (null)          a
=                 4               (null)          a
=                 0               (null)          b
return            b               (null)          (null)
```

- **CODE OPTIMIZATION**

Dead code elimination

```
void dead_code_elimination(map<int,vector<string>>& stmts)
{
   //after return everything is dead
   vector<pair<int,vector<string>>> v;
   for(auto x: stmts)
   {
       v.push_back(make_pair(x.first,x.second));
   }
   for(auto it1=v.begin();it1!=v.end();it1++)
   {
       string ret=it1->second[0];
       if(ret=="return")
       {
           for(auto it2=it1+1;it2!=v.end();it2++)
           {
               auto it3=it2;
               stmts.erase(it3->first);
           }
       }
   }
   cout<<"After removing stmts below return"<<endl;
   display(stmts);

   //eliminate variables that are defined and not used before their next
definition
   vector<pair<int,vector<string>>> v2;
   for(auto x: stmts)
   {
       v2.push_back(make_pair(x.first,x.second));
   }
   for(auto it1=v2.begin();it1!=v2.end();it1++)
   {
       string res=it1->second[3];
       string op=it1->second[0];
       if(op=="if" || op=="label" || op=="return")
       {
           continue;
```

```
        }
        for(auto it2=it1+1;it2!=v2.end();it2++)
        {
            if(res==it2->second[1] || res==it2->second[2]) break;
            if(res==it2->second[3] && (it2->second[0]=="=" || it2->second[0]=="+" || it2->second[0]=="-" || it2->second[0]=="*" || it2->second[0]=="/"))
            {
                stmts.erase(it1->first);
            }
        }
    }
    cout<<"After removing variables that are redifined before their use"<<endl;
    display(stmts);
}
```

Common subexpression elimination

```
void cse(map<int,vector<string>>& stmts)
{
    vector<pair<int,vector<string>>> v;
    for(auto x: stmts)
    {
        v.push_back(make_pair(x.first,x.second));
    }
    for(auto it1=v.begin();it1!=v.end();it1++)
    {
        for(auto it2=it1+1;it2!=v.end();it2++)
        {
            if(it1->second[1]==it2->second[1] && it1->second[2]==it2->second[2] && it1->second[0]==it2->second[0])
            {
                string replace=it1->second[3];
                string res=it2->second[3];
                stmts.erase(it2->first);
                for(auto it3=it2+1;it3!=v.end();it3++)
                {
                    if(it3->second[1]==res)
                    {
                        it3->second[1]=replace;
                        stmts[it3->first][1]=replace;
                    }
                    if(it3->second[2]==res)
                    {
                        it3->second[2]=replace;
                        stmts[it3->first][2]=replace;
                    }
                }
            }
        }
    }

}
```

Constant folding and constant propagation

```
void constant_propagation_and_folding(map<int,vector<string>>& stmts)
{
    vector<pair<int,vector<string>>> v;
    for(auto x: stmts)
    {
        v.push_back(make_pair(x.first,x.second));
    }
    for(auto it1=v.begin();it1!=v.end();it1++)
    {
        string res=it1->second[3];
        string op=it1->second[0];
        string arg1=it1->second[1];
        string arg2=it1->second[2];
        if(op=="label" || op=="return" || op=="if")
        {
            continue;
        }
        if(is_number(arg1) && arg2=="NULL")
        {
            string num=arg1;
            for(auto it2=it1+1;it2!=v.end();it2++)
            {
                if(it2->second[3]==res && (it2->second[1]!=res && it2->second[1]!=res)) break;
                if(it2->second[1]==res)
                {
                    it2->second[1]=num;
                    stmts[it2->first][1]=num;
                }
                if(it2->second[2]==res)
                {
                    it2->second[2]=num;
                    stmts[it2->first][2]=num;
                }
            }
        }
        if(is_number(arg1) && is_number(arg2))
        {
            string num;
            if(op=="+")
            {
                stringstream s1(arg1);
                stringstream s2(arg2);
                int num1,num2,num3;
                s1>>num1;
                s2>>num2;
                num=to_string(num1+num2);
            }
            else if(op=="*")
            {
                stringstream s1(arg1);
                stringstream s2(arg2);
                int num1,num2,num3;
                s1>>num1;
                s2>>num2;
                num=to_string(num1*num2);
            }
            else if(op=="-")
            {
                stringstream s1(arg1);
                stringstream s2(arg2);
                int num1,num2,num3;
                s1>>num1;
                s2>>num2;
                num=to_string(num1-num2);
            }
            else if(op=="/")
            {
                stringstream s1(arg1);
                stringstream s2(arg2);
                int num1,num2,num3;
                s1>>num1;
                s2>>num2;
                num=to_string(num1/num2);
            }
            it1->second[1]=num;
            it1->second[2]="NULL";
            stmts[it1->first][1]=num;
            stmts[it1->first][2]="NULL";
            for(auto it2=it1+1;it2!=v.end();it2++)
            {
                if(it2->second[3]==res  &&  (it2->second[1]!=res  &&  it2->second[1]!=res)) break;
                if(it2->second[1]==res)
                {
                    it2->second[1]=num;
                    stmts[it2->first][1]=num;
                }
                if(it2->second[2]==res)
                {
                    it2->second[2]=num;
                    stmts[it2->first][2]=num;
                }
            }
        }
    }
}
```

Copy propagation

```cpp
void copy_propagation(map<int,vector<string>>& stmts)
{
    vector<pair<int,vector<string>>> v;
    for(auto x: stmts)
    {
        v.push_back(make_pair(x.first,x.second));
    }
    for(auto it1=v.begin();it1!=v.end();it1++)
    {
        string op=it1->second[0];
        string arg1=it1->second[1];
        string res=it1->second[3];
        if(op=="=")
        {
            for(auto it2=it1+1;it2!=v.end();it2++)
            {
                if(it2->second[3]==res) break;
                if(it2->second[1]==res)
                {
                    it2->second[1]=arg1;
                    stmts[it2->first][1]=arg1;
                }
                if(it2->second[2]==res)
                {
                    it2->second[2]=arg1;
                    stmts[it2->first][2]=arg1;
                }
            }
        }
    }
}
```

- **Instructions for running**
  - The mini-compiler has three folders that serve 3 purposes:
    - Evaluation
    - ICG
    - Optimization
  - Each folder has bash script files that take in input c++ files and produce their respective outputs on the terminals
  - The ICG files also creates text files with the three address code required as input for the Optimization code

6.RESULTS AND possible shortcomings of your Mini-Compiler

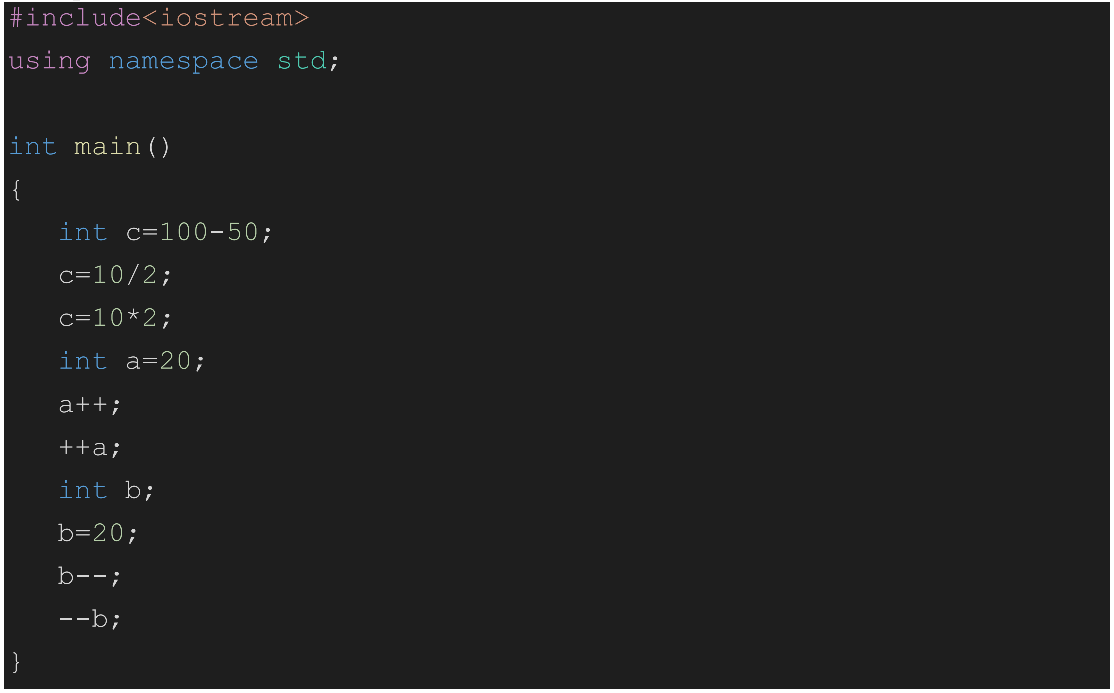
- **Results:**
  - The C++ mini-compiler is successful at all stages of compilation and takes care of numerous C++ constructs efficiently. The different stages of output obtained are
    - Symbol table
    - Three address code
    - Quadruples
    - Optimized code
- **Possible shortcomings:**
  - The symbol table has been implemented as an array of structures. Insertion into the symboltable is of O(1) time complexity and lookup is of O(n) time complexity on average.
  - Therefore, for large symbol tables, the look up can become slow.
  - All types of tokens, temporaries and labels are updated to the symbol table even though they differ in meaning and usage.
  - ICG and TAC have not been handled for arrays

7.SNAPSHOTS

- **Expression evaluation:**

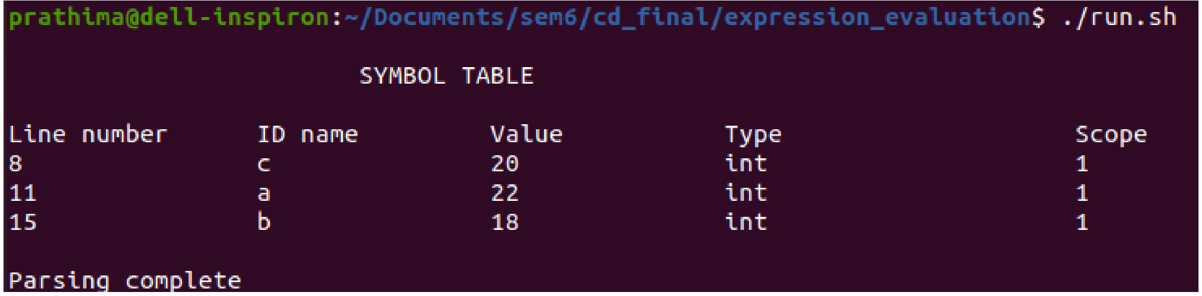
  Statements containing expressions are evaluated. The resulting values of the variables are updated accordingly in the symbol table along with the appropriate data types.

(i) Input 1:



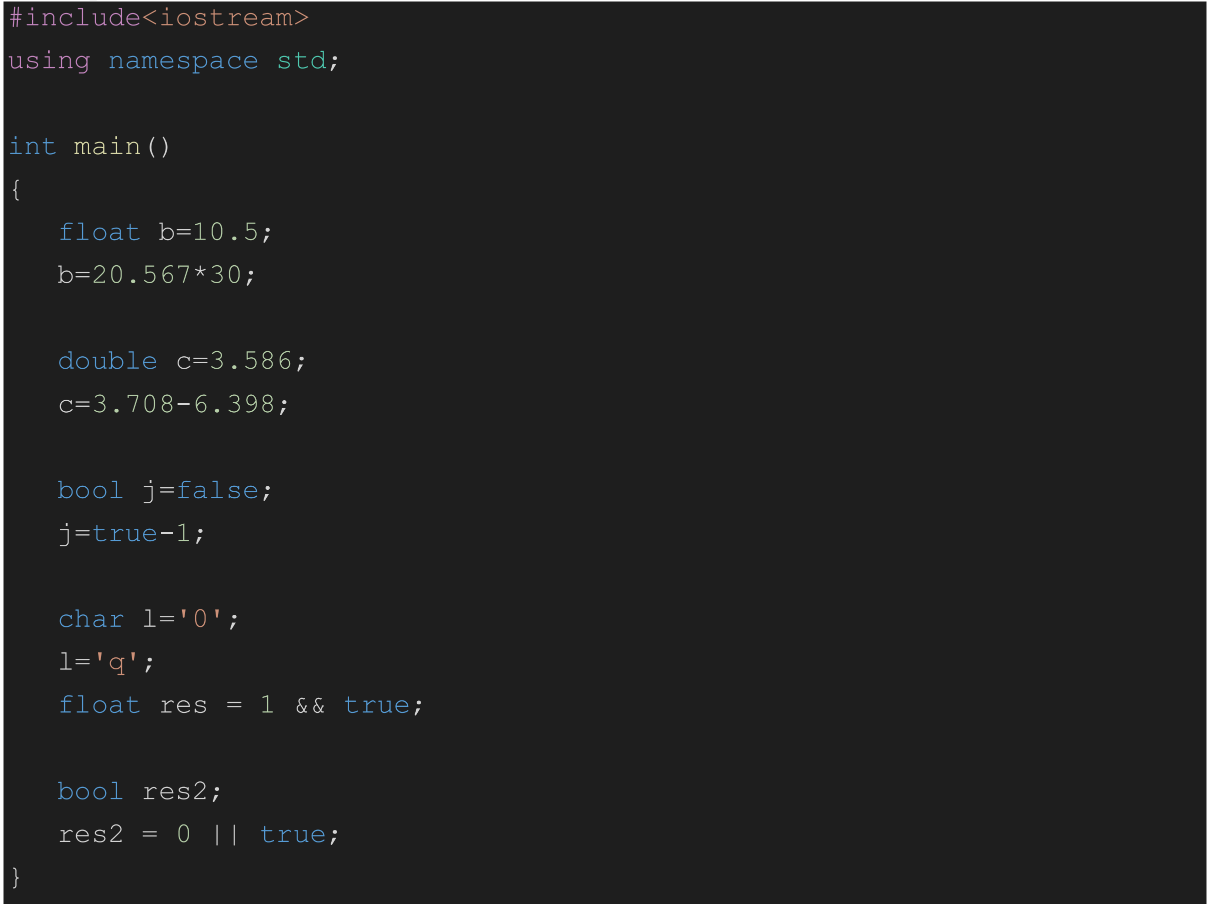
```
#include<iostream>
using namespace std;

int main()
{
   int c=100-50;
   c=10/2;
   c=10*2;
   int a=20;
   a++;
   ++a;
   int b;
   b=20;
   b--;
   --b;
}
```

(ii) Output 1:



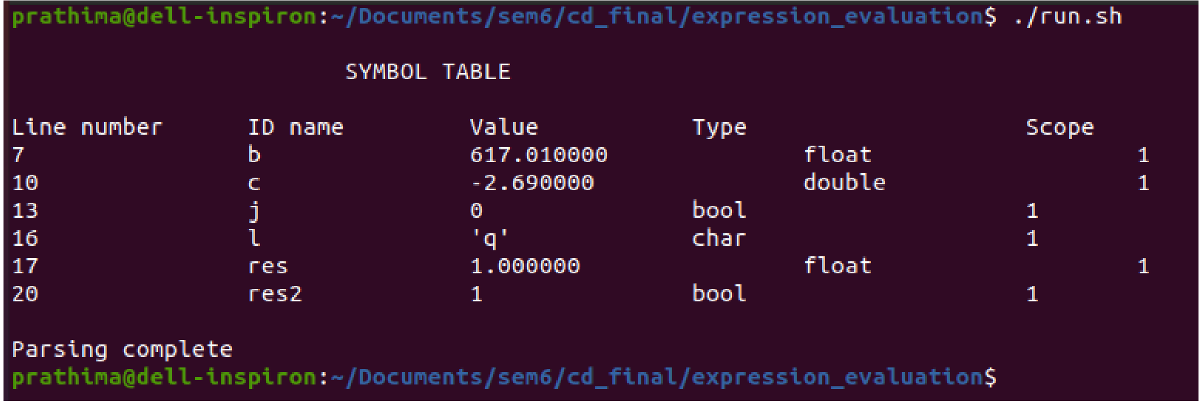
```
prathima@dell-inspiron:~/Documents/sem6/cd_final/expression_evaluation$ ./run.sh

                         SYMBOL TABLE

Line number       ID name          Value           Type                     Scope
8                 c                20              int                      1
11                a                22              int                      1
15                b                18              int                      1

Parsing complete
```

(iii) Input 2:

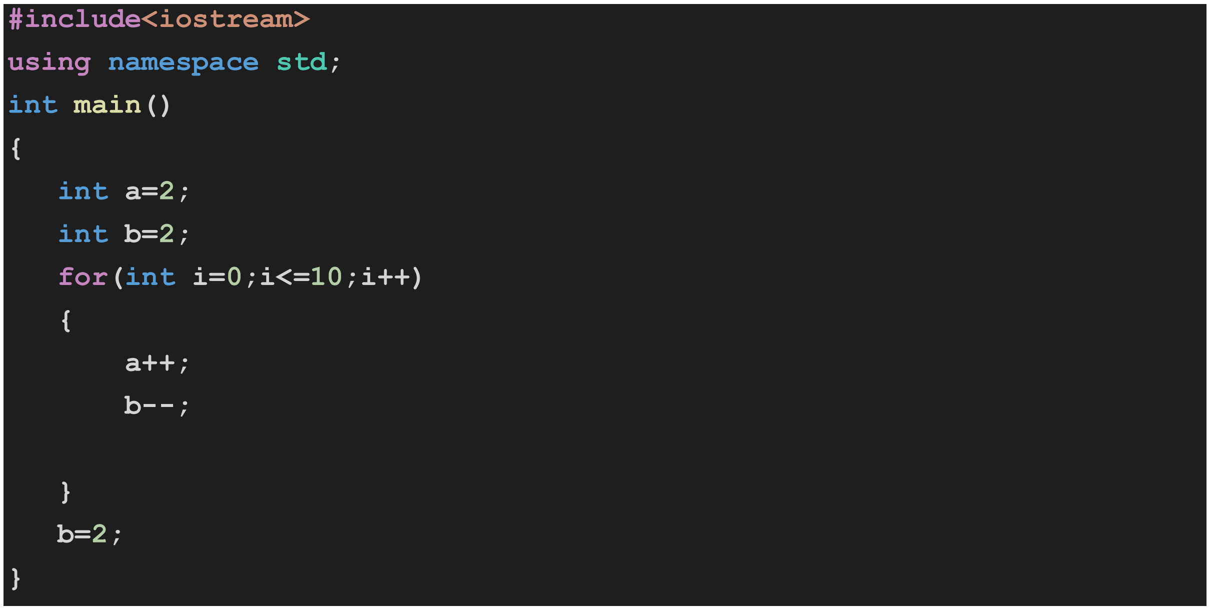


```
#include<iostream>
using namespace std;

int main()
{
    float b=10.5;
    b=20.567*30;

    double c=3.586;
    c=3.708-6.398;

    bool j=false;
    j=true-1;

    char l='0';
    l='q';
    float res = 1 && true;

    bool res2;
    res2 = 0 || true;
}
```

(iv) Output 2:

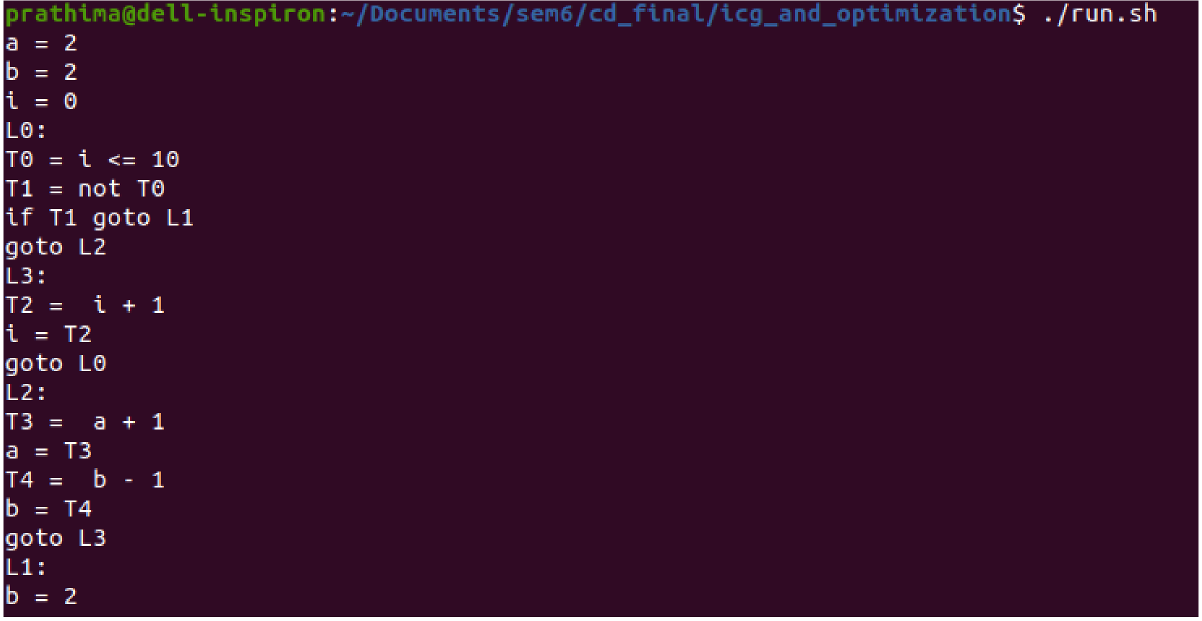


```
prathima@dell-inspiron:~/Documents/sem6/cd_final/expression_evaluation$ ./run.sh

                        SYMBOL TABLE

Line number      ID name         Value           Type                    Scope
7                b               617.010000              float                    1
10               c               -2.690000               double                   1
13               j               0               bool                    1
16               l               'q'             char                    1
17               res             1.000000                float                    1
20               res2            1               bool                    1

Parsing complete
prathima@dell-inspiron:~/Documents/sem6/cd_final/expression_evaluation$
```



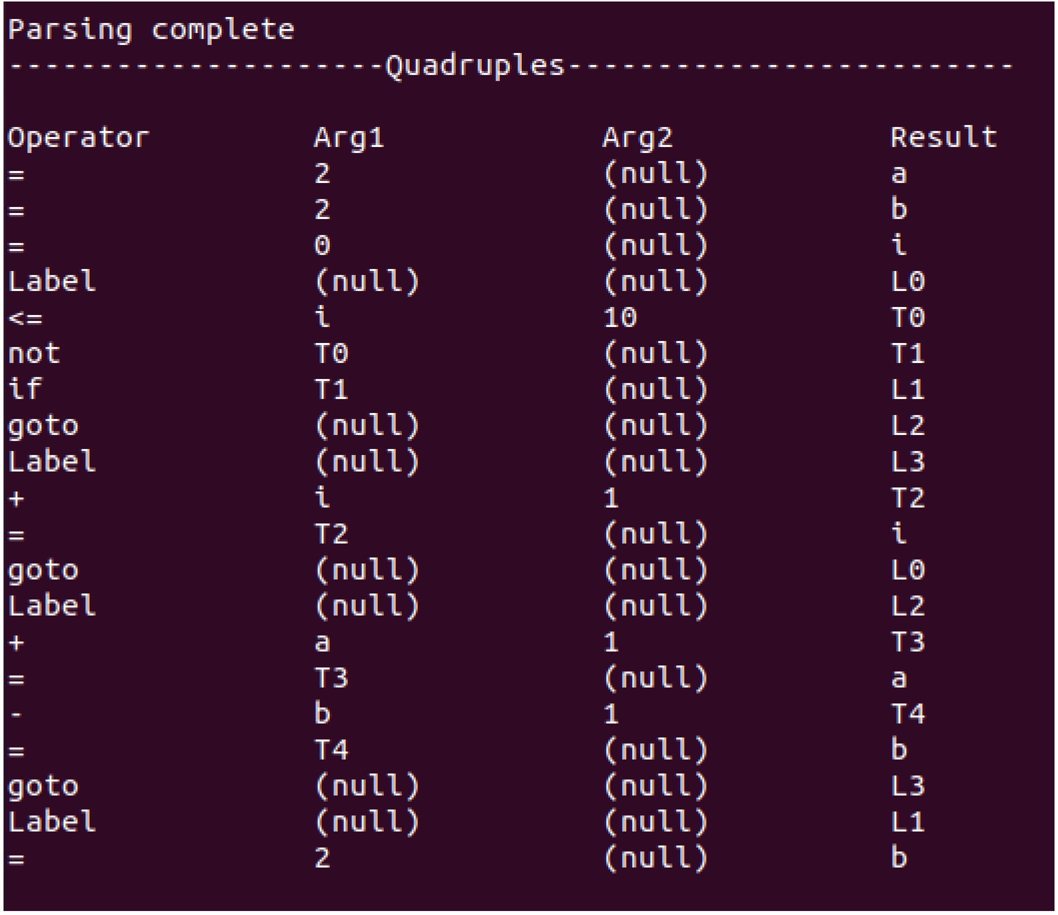
```
#include<iostream>
using namespace std;
int main()
{
   int a=2;
   int b=2;
   for(int i=0;i<=10;i++)
   {
       a++;
       b--;

   }
   b=2;
}
```

- For a basic for loop, the three address code will look like this:



```
prathima@dell-inspiron:~/Documents/sem6/cd_final/icg_and_optimization$ ./run.sh
a = 2
b = 2
i = 0
L0:
T0 = i <= 10
T1 = not T0
if T1 goto L1
goto L2
L3:
T2 =  i + 1
i = T2
goto L0
L2:
T3 =  a + 1
a = T3
T4 =  b - 1
b = T4
goto L3
L1:
b = 2
```

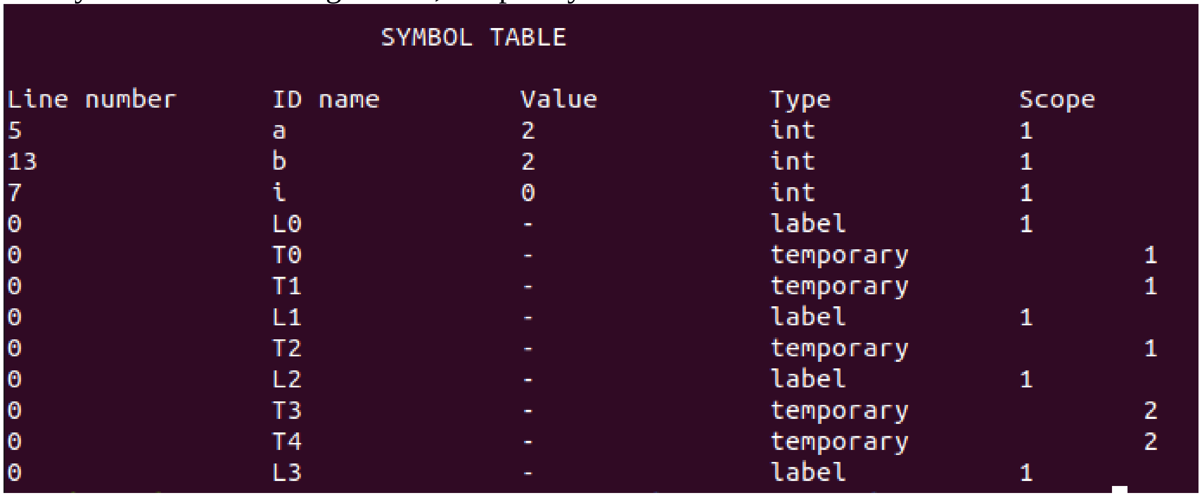
- Quadruples:



```
Parsing complete
--------------------Quadruples-----------------------

Operator         Arg1           Arg2           Result
=                2              (null)         a
=                2              (null)         b
=                0              (null)         i
Label            (null)         (null)         L0
<=               i              10             T0
not              T0             (null)         T1
if               T1             (null)         L1
goto             (null)         (null)         L2
Label            (null)         (null)         L3
+                i              1              T2
=                T2             (null)         i
goto             (null)         (null)         L0
Label            (null)         (null)         L2
+                a              1              T3
=                T3             (null)         a
-                b              1              T4
=                T4             (null)         b
goto             (null)         (null)         L3
Label            (null)         (null)         L1
=                2              (null)         b
```

-

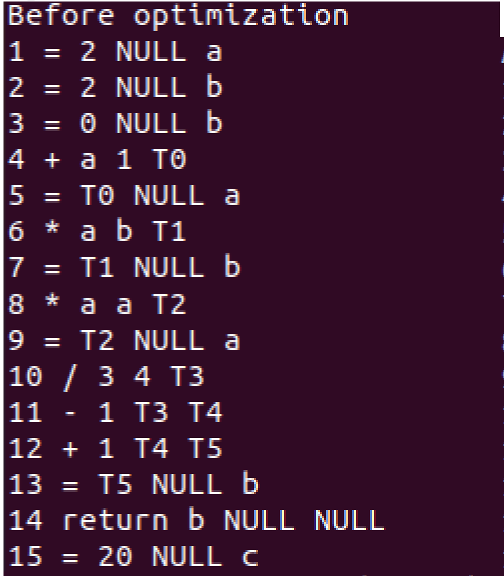
- The symbol table showing Labels, temporary



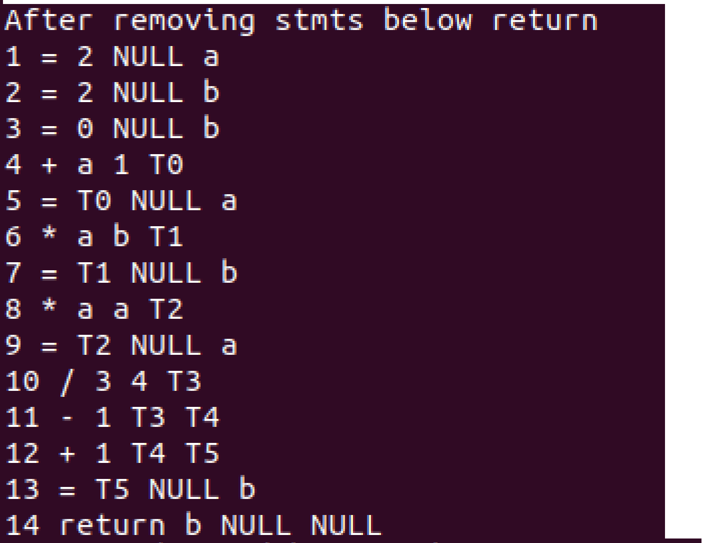
```
                         SYMBOL TABLE

Line number      ID name        Value          Type             Scope
5                a              2              int              1
13               b              2              int              1
7                i              0              int              1
0                L0             -              label            1
0                T0             -              temporary                1
0                T1             -              temporary                1
0                L1             -              label            1
0                T2             -              temporary                1
0                L2             -              label            1
0                T3             -              temporary                2
0                T4             -              temporary                2
0                L3             -              label            1
```

-

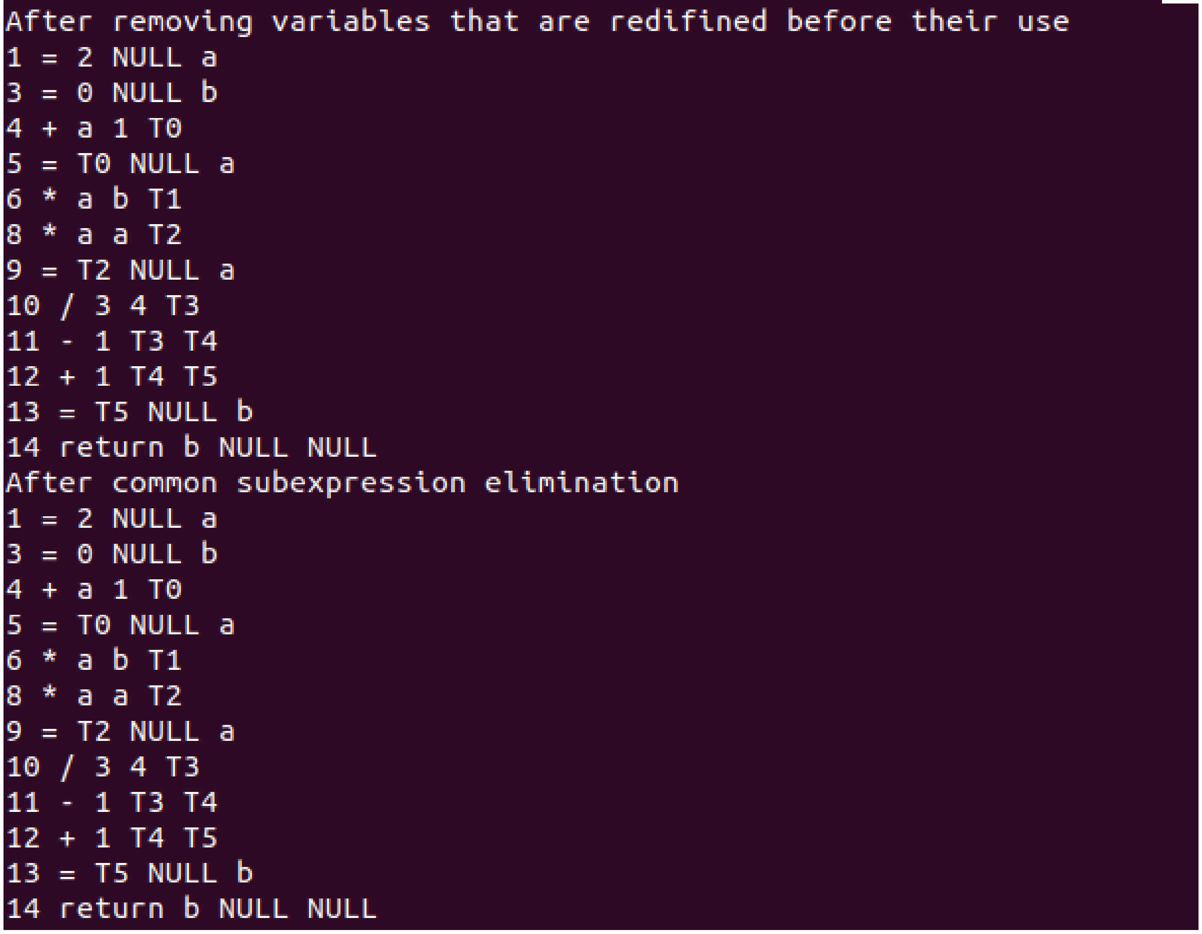
Optimizations:



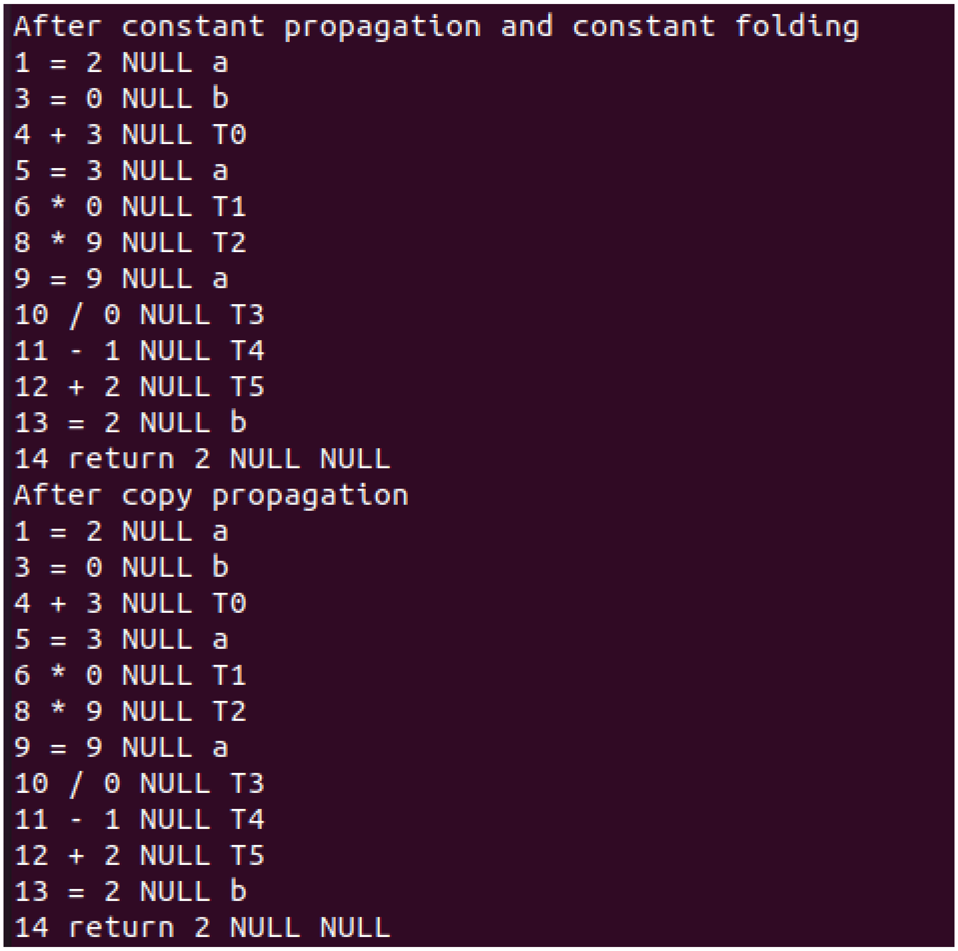
```
Before optimization
1 = 2 NULL a
2 = 2 NULL b
3 = 0 NULL b
4 + a 1 T0
5 = T0 NULL a
6 * a b T1
7 = T1 NULL b
8 * a a T2
9 = T2 NULL a
10 / 3 4 T3
11 - 1 T3 T4
12 + 1 T4 T5
13 = T5 NULL b
14 return b NULL NULL
15 = 20 NULL c
```



```
After removing stmts below return
1 = 2 NULL a
2 = 2 NULL b
3 = 0 NULL b
4 + a 1 T0
5 = T0 NULL a
6 * a b T1
7 = T1 NULL b
8 * a a T2
9 = T2 NULL a
10 / 3 4 T3
11 - 1 T3 T4
12 + 1 T4 T5
13 = T5 NULL b
14 return b NULL NULL
```



```
After removing variables that are redifined before their use
1 = 2 NULL a
3 = 0 NULL b
4 + a 1 T0
5 = T0 NULL a
6 * a b T1
8 * a a T2
9 = T2 NULL a
10 / 3 4 T3
11 - 1 T3 T4
12 + 1 T4 T5
13 = T5 NULL b
14 return b NULL NULL
After common subexpression elimination
1 = 2 NULL a
3 = 0 NULL b
4 + a 1 T0
5 = T0 NULL a
6 * a b T1
8 * a a T2
9 = T2 NULL a
10 / 3 4 T3
11 - 1 T3 T4
12 + 1 T4 T5
13 = T5 NULL b
14 return b NULL NULL
```



```
After constant propagation and constant folding
1 = 2 NULL a
3 = 0 NULL b
4 + 3 NULL T0
5 = 3 NULL a
6 * 0 NULL T1
8 * 9 NULL T2
9 = 9 NULL a
10 / 0 NULL T3
11 - 1 NULL T4
12 + 2 NULL T5
13 = 2 NULL b
14 return 2 NULL NULL
After copy propagation
1 = 2 NULL a
3 = 0 NULL b
4 + 3 NULL T0
5 = 3 NULL a
6 * 0 NULL T1
8 * 9 NULL T2
9 = 9 NULL a
10 / 0 NULL T3
11 - 1 NULL T4
12 + 2 NULL T5
13 = 2 NULL b
14 return 2 NULL NULL
```

8.CONCLUSIONS

- We have built a Mini C++ compiler with if, if-else, while and for constructs which scans the input, generates appropriate tokens, parses the input, performs semantic analysis, generates intermediate code and optimizes the code.
- It catches syntax errors and semantic errors and displays appropriate error messages.
- It also rightly handles the scope throughout the input file.
- It compiles the given input files without encountering any conflicts and giving any errors.

9.FURTHER ENHANCEMENTS

- Visibility modes handling could be built into the compiler so that it can handle various visibility modes of the variables such as private, public and protected.
- Separate structures can be used to build symbol tables for tokens, temporaries and labels. Further, a union of these structures can be defined to utilize memory appropriately.

10. BIBLIOGRAPHY

- https://www.geeksforgeeks.org/symbol-table-compiler/
- https://www.geeksforgeeks.org/yacc-program-to-implement-a-calculator-and-recognize-a-valid-arithmetic-expression/

  .