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:

Arun Srinivasan P	**PES1201800383**
Rithvik Vibhu	**PES1201801205**
DG Shivu	**PES1201801284**
N.L Akshaya	**PES1201801110**

Under the guidance of

Mahesh HB
Assistant 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 language which was chosen for the implementation of a compiler was C++. The two major constructs focussed upon in my implementation are if, if-else and do-while. The language which was used in the lexical phase was Lex. The language used for parsing was yacc/bison. The intermediate code is also generated in the parsing phase . The optimizations are performed in python taking the intermediate code in three address form as input.

2. Architecture of Language

The constructs which are handled in the language are do-while and if-else. Other basics of the language such as types,values, classes, nested control statements are also handled by the grammar. The ICG is done in three address code and the optimizations are performed using python. The optimizations which have been performed are constant folding, constant-copy propagation, dead if elimination and strength reduction.

3. Context Free Grammar

Lex File:

```
alpha [A-Za-z_]
digit [0-9]
float ({digit})+\.({digit})+
%%
\/\*(.*\n)*.*\*\/  {};
\/\/(.*) ;
[ \t\n] ;

"do" {yylval.value=strdup(yytext);return DO;}
"while" {yylval.value=strdup(yytext);return WHILE;}
"if" {yylval.value=strdup(yytext);return IF;}
"else" {yylval.value=strdup(yytext);return ELSE;}
"cout" {yylval.value=strdup(yytext);return COUT;}
"endl" {yylval.value=strdup(yytext);return ENDL;}
"break" {yylval.value=strdup(yytext);return BREAK;}
"continue" {yylval.value=strdup(yytext);return CONTINUE;}
```

```
"int" {yylval.value=strdup(yytext);return INT;}
"float" {yylval.value=strdup(yytext);return FLOAT;}
"char" {yylval.value=strdup(yytext);return CHAR;}
"void" {yylval.value=strdup(yytext);return VOID;}
"double" {yylval.value=strdup(yytext);return DOUBLE;}
"bool"   {yylval.value=strdup(yytext);return BOOL;}
"#include" {yylval.value=strdup(yytext);return INCLUDE;}
"main()" {yylval.value=strdup(yytext);return TOK_MAIN;}
"class" {yylval.value=strdup(yytext);return CLASS;}
"private" {yylval.value=strdup(yytext);return PRIVATE;}
"protected" {yylval.value=strdup(yytext);return PROTECTED;}
"public" {yylval.value=strdup(yytext);return   PUBLIC;}
"using" {yylval.value=strdup(yytext);return USING;}
"namespace" {yylval.value=strdup(yytext);return NAMESPACE;}
"std"   {yylval.value=strdup(yytext);return STD;}
"char*" {yylval.value=strdup(yytext);return STRING;}
"char *"
{yylval.value=strdup(yytext);yylval.value[strlen(yylval.value)-1]='\0';yylval.value[strlen(yyl
val.value)-1]='*';return STRING;}
("false"|"true") {yylval.value=strdup(yytext); yylval.type=2; return BOOL_VAL;}


{digit}+          {yylval.value=strdup(yytext); yylval.type=0; return NUM;}
{alpha}({alpha}|{digit})*       {yylval.value=strdup(yytext); yylval.type=1; return ID;}
{alpha}({alpha}|{digit})*"\.h"?   {yylval.value=strdup(yytext);return HEADER;}

{float} {yylval.value=strdup(yytext); yylval.type=3; return FLOAT_VAL;}
'[a-zA-Z]' {yylval.value=strdup(yytext); yylval.type=4; return CHAR_VAL;}


\".*\" {yylval.value=strdup(yytext); yylval.type=5; return STRING_VAL;}
"<"   {yylval.value=strdup(yytext);return lt;}
">"     {yylval.value=strdup(yytext);return gt;}
"="     {yylval.value=strdup(yytext);return eq;}
"<="    {yylval.value=strdup(yytext);return lteq;}
">="    {yylval.value=strdup(yytext);return gteq;}
"=="    {yylval.value=strdup(yytext);return eqeq;}
"!="    {yylval.value=strdup(yytext);return neq;}
"+"     {yylval.value=strdup(yytext);return pl;}
"-"     {yylval.value=strdup(yytext);return min;}
```

```
"*"     {yylval.value=strdup(yytext);return mul;}
"/"     {yylval.value=strdup(yytext);return division;}
"++"    {yylval.value=strdup(yytext);return incr;}
"--"    {yylval.value=strdup(yytext);return decr;}
"!"     {yylval.value=strdup(yytext);return not;}
"||"    {yylval.value=strdup(yytext);return or;}
"&&"    {yylval.value=strdup(yytext);return and;}
"%"     {yylval.value=strdup(yytext);return perce;}
.       {yylval.value=strdup(yytext);return yytext[0];}
```

Yacc File(Grammar):

```
S
        : START
        ;

START
        : INCLUDE  lt HEADER gt START
        | INCLUDE "\"" HEADER "\"" START
        | INCLUDE lt ID gt START
        | MAIN
        | ASSIGN_EXPR ';' START
        | CLASS_DEF ';' START
        | FUNCTION START
        | USING NAMESPACE STD ';' START
        ;

FUNCTION
        :FUNC_DEF
        |FUNC_DECL
        ;
FUNC_DECL
        :TYPE ID '(' PARAMETER_LIST ')' ';'
        |TYPE ID '(")' ';'
        ;

FUNC_DEF
        :TYPE ID  '(' PARAMETER_LIST ')'  MAIN_BODY
        |TYPE ID '(")' MAIN_BODY
        ;
```

```
PARAMETER_LIST
        :TYPE ID ',' PARAMETER_LIST
        |TYPE ID
        ;

CLASS_DEF
        :CLASS ID '{' CLASS_BODY '}'
        ;

CLASS_BODY
        :ACCESS_SPECIFIER ':' CLASS_BODY
        |TYPE ID ';' CLASS_BODY
        |TYPE ID ';'
        |FUNC_DECL
        ;



ACCESS_SPECIFIER
        :PRIVATE
        |PROTECTED
        |PUBLIC
        ;



MAIN
        : VOID TOK_MAIN MAIN_BODY
        | INT TOK_MAIN MAIN_BODY
        ;
MAIN_BODY
        : '{' LINES '}'
        | '{''}'
        ;

LINES
        :LINES STATEMENT ';'
        |LINES LOOP
        |STATEMENT ';'
        |LOOP
        |';'
        ;
```

```
LOOP
        :  IF_EXPR LOOPBODY
        | IF_EXPR LOOPBODY ELSE LOOPBODY
        | DO LOOPBODY WHILE  '(' ARITH_EXPR ')'';'

        ;

IF_EXPR
        : IF '(' ARITH_EXPR ')'

LOOPBODY
       : '{' LINES '}'
        |STATEMENT ';'
        ;
STATEMENT
        : ASSIGN_EXPR
        | ARITH_EXPR
        | PRINT
        ;

PRINT
        : COUT CASCADE
        ;

CASCADE
        :lt lt ID CASCADE
        |lt lt ID
        |lt lt STRING_VAL CASCADE
        |lt lt STRING_VAL
        |lt lt ENDL
        ;

ASSIGN_EXPR
        :ID eq ARITH_EXPR
        |TYPE ID eq ARITH_EXPR
        |TYPE ID
        ;
```

```
ARITH_EXPR
        : LOGICAL_EXPR
        | ID unary_arop
        | unary_arop ID
        | not LOGICAL_EXPR
        ;
LOGICAL_EXPR
        : RELLATIONAL_EXPR
        | LOGICAL_EXPR LOG RELLATIONAL_EXPR
        ;
RELLATIONAL_EXPR
        :ADDITIVE
        |RELLATIONAL_EXPR REL ADDITIVE
        ;
ADDITIVE
        :MULTIPLCIATIVE
        |ADDITIVE ADD_SUB MULTIPLCIATIVE
        ;

MULTIPLCIATIVE
        :FINAL
        |MULTIPLCIATIVE PROD_QUO FINAL
        ;

FINAL
        :'(' ARITH_EXPR ')'
        |LITERAL
        ;

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

unary_arop
```

```
          : incr
          | decr
          ;


ADD_SUB
          : pl
          | min
          ;

PROD_QUO
          : division
          | mul
          | perce
          ;

LOG
          : and
          | or
          ;

REL
          : eqeq
          | gteq
          | lteq
          | neq
          | lt
          | gt
          ;

LITERAL
          : ID
          | FLOAT_VAL
          | CHAR_VAL
          | BOOL_VAL
          | STRING_VAL
          | NUM
          | min ID
          | min NUM
          ;
```

4. Design Strategy

Symbol Table:

The structure of the symbol table is as follows:

1) Line: Tells the line number of the particular variable or temporary.
2) Scope: An integer which tells under which nesting the variable lies. If the value is 0 then the scope is global.
3) Name: A 32 character array which contains the name of the variables.
4) Type: Tells if it's a temporary or an identifier.
5) Value: Stores the value of the variable.(string)
6) Datatype: Stores the data type of the variable.(string)

Intermediate Code:

The structure of the quadruples is as follows:

1) op: Operator
2) src1: Argument 1
3) src2: Argument 2
4) des: Destination after performing the operation on the arguments.

Code Optimization:

This is a class which contains two major components:

1) Leaders: Leaders of the ICG generated.
2) Basicblocks: ICG separated into basic blocks.

The optimizations applied in order are:

1) Strength Reduction
2) Constant Folding
3) Constant Propagation
4) Eliminating Dead Control Statements

Error Handling:

We have made use of the yyerror function in lex/yacc to display the error messages.

This function would detect if there is redeclaration of a variable, addition of wrong types. There is also an error count variable which checks the number of messages to print on the screen like a compiler. A few errors would be shown in the snapshots section.

5. Implementation Details

Symbol Table:

This structure contains the variables mentioned in the above design section. Whenever a variable is being initialized or assigned there would be a change in the value of this particular table.

For example,

```
Int main()
{
        Int a=20;
}
```

‘a’ would be entered into the symbol table with
Scope- 1
Value- 20
Dataype- int
Type- identifier
Name- a

Intermediate Code:

Here the variables mentioned in the above design section would be populated when a binary operator would be performed on the variable.

For example,
int a=2+3;

The icg would be in this form

Op	Arg1	Arg2	Dest
+	2	3	T0
=	T0		a

Code Optimization:

Since python makes more simpler use of strings it has been used to perform the optimizations. It takes the ICG as input and performs the

1) Strength Reduction:
   Before Reduction:
   Int a=3*4;
   After Reduction
   Int a=3<<2;

2) Constant Folding:
   Before Folding,
   Int a=2+3;
   After Folding,
   Int a=5;

3) Constant Propagation:
   Before Propagation,
   Int a=2;
   Int c=2+a;
   After propagation.
   Int a=2;
   Int c=2+2;

4) Eliminating Dead if,
   Before elimination,
   if(0)
   {
   Int a=2;
   }
   Int c=2;
   After elimination,
   Int c=2;

These optimizations are performed in the same order and to reduce the code and also make the operations more simpler.

6. How to Run

```
lex ICG.l
yacc -d ICG.y -v
gcc  y.tab.c symbol.c quad.c -lfl -ly
./a.out < rand.cpp
```

Where rand.cpp is the input

An output called ICG.txt containing the intermediate code is created.

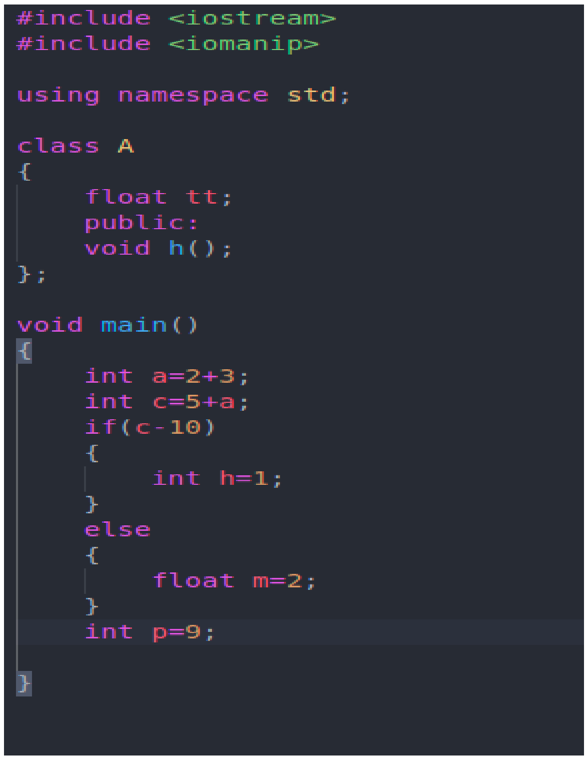
Then for optimizations call,
Python3 optimization.py

For simplicity we have provided a script,
So we could just run
./script.sh
Python3 optimization.py

7. Results and Shortcomings

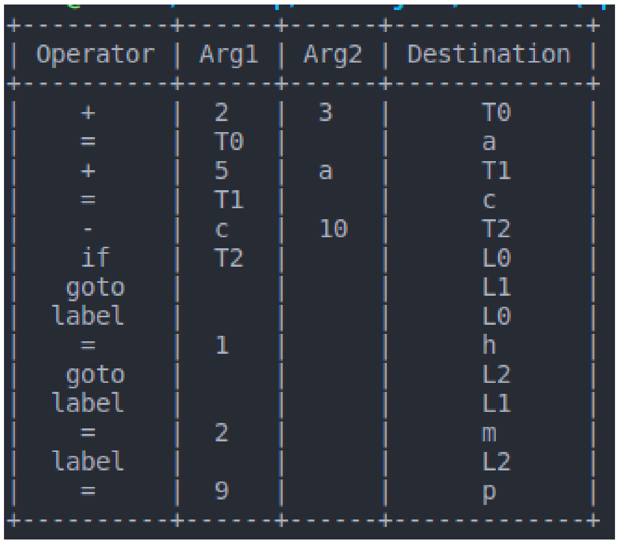
Example 1: ICG formation of if-else statement along with optimizations

Input:

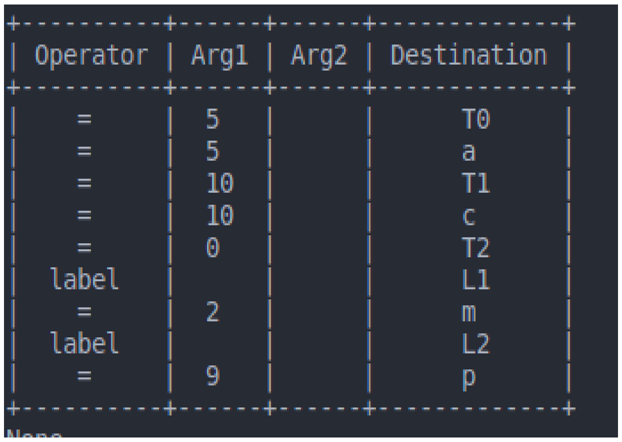


```
#include <iostream>
#include <iomanip>

using namespace std;

class A
{
    float tt;
    public:
    void h();
};

void main()
{
    int a=2+3;
    int c=5+a;
    if(c-10)
    {
        int h=1;
    }
    else
    {
        float m=2;
    }
    int p=9;

}
```

ICG before optimization and after optimizations:



```
+----------+-----+-----+------------+
| Operator | Arg1 | Arg2 | Destination |
+----------+-----+-----+------------+
|    +     |  2  |  3  |     T0     |
|    =     |  T0 |     |     a      |
|    +     |  5  |  a  |     T1     |
|    =     |  T1 |     |     c      |
|    -     |  c  |  10 |     T2     |
|    if    |  T2 |     |     L0     |
|   goto   |     |     |     L1     |
|  label   |     |     |     L0     |
|    =     |  1  |     |     h      |
|   goto   |     |     |     L2     |
|  label   |     |     |     L1     |
|    =     |  2  |     |     m      |
|  label   |     |     |     L2     |
|    =     |  9  |     |     p      |
+----------+-----+-----+------------+
```



```
+----------+-----+-----+------------+
| Operator | Arg1 | Arg2 | Destination |
+----------+-----+-----+------------+
|    =     |  5  |     |     T0     |
|    =     |  5  |     |     a      |
|    =     |  10 |     |     T1     |
|    =     |  10 |     |     c      |
|    =     |  0  |     |     T2     |
|  label   |     |     |     L1     |
|    =     |  2  |     |     m      |
|  label   |     |     |     L2     |
|    =     |  9  |     |     p      |
+----------+-----+-----+------------+
```

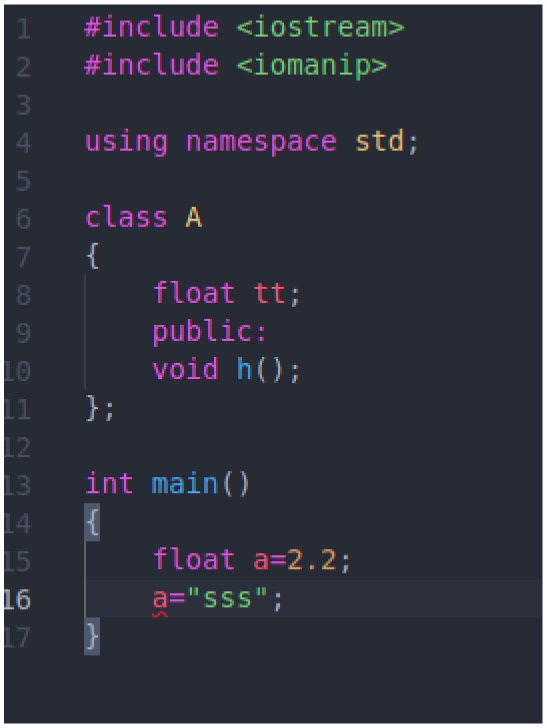
Given the above input , we first get the ICG which is not optimized in the figure to the left. The figure to the right is the optimized ICG.

ICG formation:
Here
Int a= 2+3 will store 2+3 in a temp.
Therefore T0 will have the temporary value 2+3.
And a will be assigned to this temporary.

Optimizations:
‘a’ will undergo constant folding and will attain the value 5.
This value would be propagated to c=5+a and hence would ‘c’ would have the value 5+5.
Then c would undergo constant folding and the value would be 10.
Now the if condition say if(c-10),
Since the condition would have if(0) it would be considered as a dead if.
Hence the line int h=1 would never be executed and is removed after optimization.

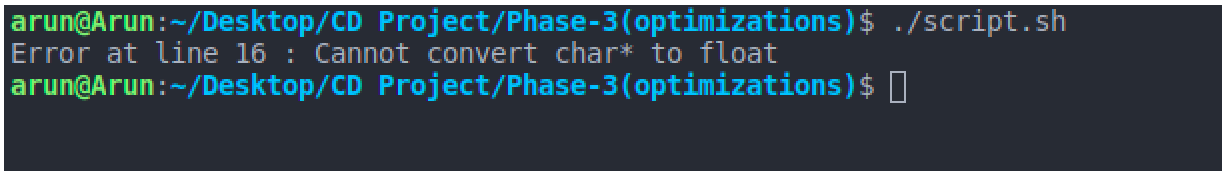
Example 2: Semantic Analysis(Type Checking)

Input:



```
#include <iostream>
#include <iomanip>

using namespace std;

class A
{
    float tt;
    public:
    void h();
};

int main()
{
    float a=2.2;
    a="sss";
}
```

Output:

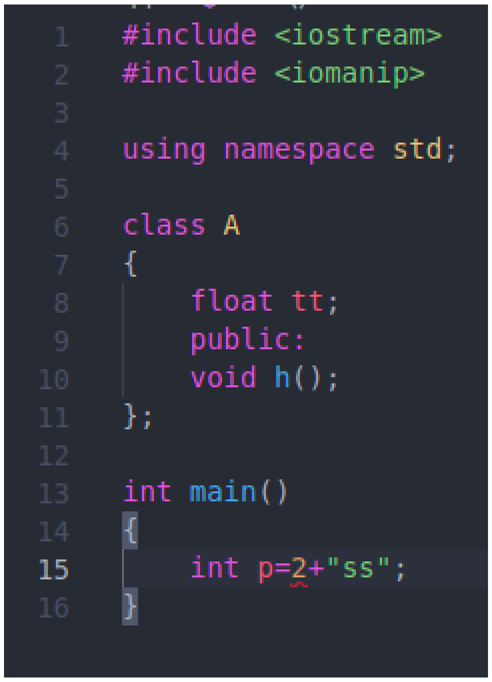


```
arun@Arun:~/Desktop/CD Project/Phase-3(optimizations)$ ./script.sh
Error at line 16 : Cannot convert char* to float
arun@Arun:~/Desktop/CD Project/Phase-3(optimizations)$
```

Here we can’t convert a char *(“sss”) to a float.

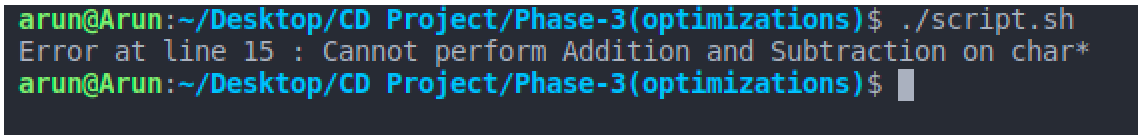
Example 3: Semantic Analysis(Type Checking) with operations

Input:



```
 1  #include <iostream>
 2  #include <iomanip>
 3
 4  using namespace std;
 5
 6  class A
 7  {
 8      float tt;
 9      public:
10      void h();
11  };
12
13  int main()
14  {
15      int p=2+"ss";
16  }
```

Output:

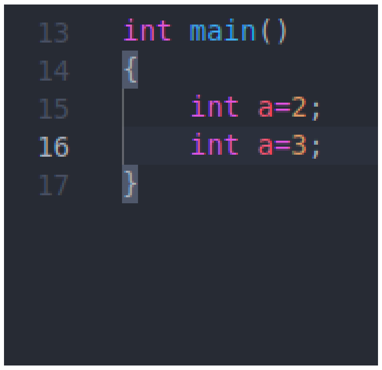


```
arun@Arun:~/Desktop/CD Project/Phase-3(optimizations)$ ./script.sh
Error at line 15 : Cannot perform Addition and Subtraction on char*
arun@Arun:~/Desktop/CD Project/Phase-3(optimizations)$
```

Here we can’t perform addition and subtraction on char*.

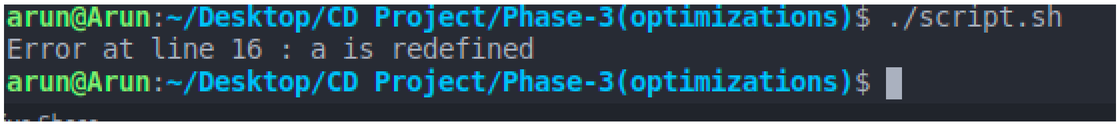
Example 4: Error message for redefining an element or accessing an element which has not been initialized.

Input1:



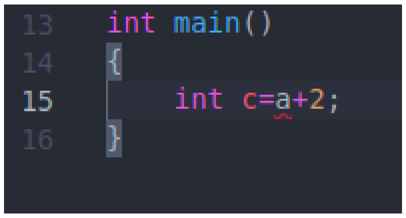
```
13  int main()
14  {
15      int a=2;
16      int a=3;
17  }
```

Output1:



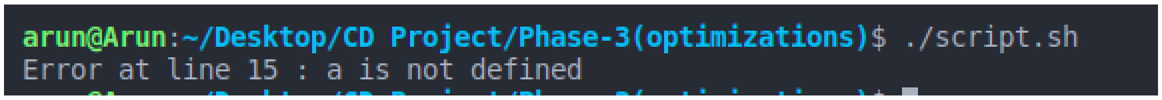
```
arun@Arun:~/Desktop/CD Project/Phase-3(optimizations)$ ./script.sh
Error at line 16 : a is redefined
arun@Arun:~/Desktop/CD Project/Phase-3(optimizations)$
```

Input2:



```
13  int main()
14  {
15      int c=a+2;
16  }
```

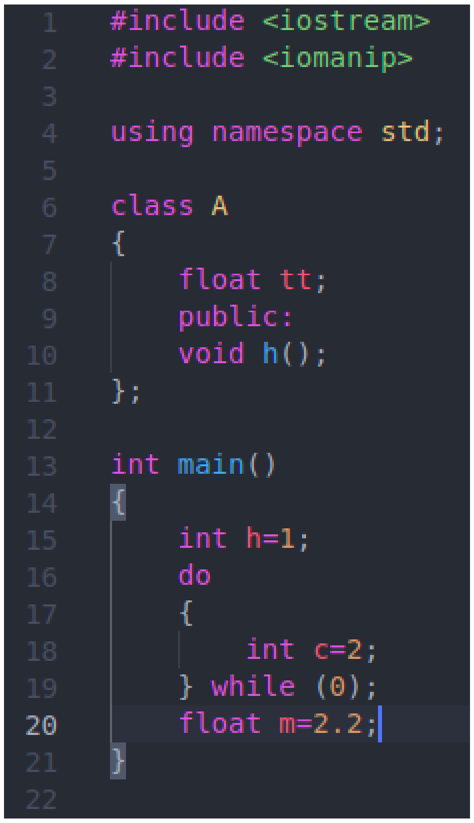
Output2:



```
arun@Arun:~/Desktop/CD Project/Phase-3(optimizations)$ ./script.sh
Error at line 15 : a is not defined
```

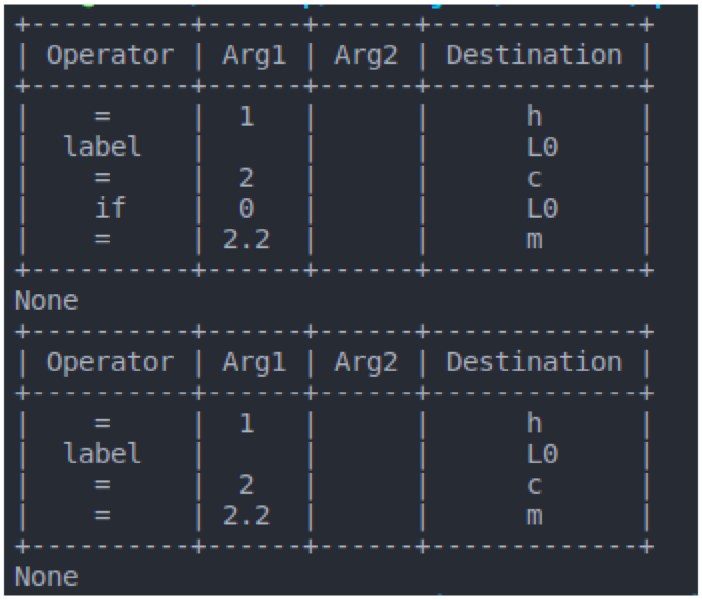
Example 5: ICG for an do-while and optimizations

Input:



```
#include <iostream>
#include <iomanip>

using namespace std;

class A
{
    float tt;
    public:
    void h();
};

int main()
{
    int h=1;
    do
    {
        int c=2;
    } while (0);
    float m=2.2;
}
```

Output:
Before and After Optimization:



```
+----------+------+------+-------------+
| Operator | Arg1 | Arg2 | Destination |
+----------+------+------+-------------+
|    =     |  1   |      |      h      |
|  label   |      |      |      L0     |
|    =     |  2   |      |      c      |
|    if    |  0   |      |      L0     |
|    =     | 2.2  |      |      m      |
+----------+------+------+-------------+
None
+----------+------+------+-------------+
| Operator | Arg1 | Arg2 | Destination |
+----------+------+------+-------------+
|    =     |  1   |      |      h      |
|  label   |      |      |      L0     |
|    =     |  2   |      |      c      |
|    =     | 2.2  |      |      m      |
+----------+------+------+-------------+
None
```

Here the only optimization performed is removing the if(0) cause it's a dead comparison.

Shortcomings:

Our shortcomings would include:

1) ICG for a variety of operations like functions and classes.

8. Conclusions

After completing this particular project, our skills in not only in understanding the compiler but also in understanding the concepts of c++ have improved . The use of lex and yacc have also increased our concepts in understanding the grammar of languages. The use of lexer has also improved our understanding in regular expressions. On a whole we also understood the phases of a compiler and the major requirements required in building a full-fledged compiler.

9. Further Enhancements

Our compiler includes many features related to if and do-while. The future improvements would include addition of other constructs such as for,ternary operator , class objects and arrays. Since C++ supports the use of the auto keyword we could use that and make the compiler determine the type of the variable.

References

1. https://cse.iitkgp.ac.in/~bivasm/notes/LexAndYaccTutorial.pdf
2. https://arcb.csc.ncsu.edu/~mueller/codeopt/codeopt00/y_man.pdf
3. https://tldp.org/HOWTO/Lex-YACC-HOWTO-5.html