Report on

"Compiler Design Mini Project - C++"

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

Compiler Design Laboratory

Bachelor of Technology
in
Computer Science & Engineering

Submitted by:

Karan Kumar G	**PES1201801883**
Sparsh Temani	**PES1201800284**
Sumukh Aithal K	**PES1201801461**

Under the guidance of

Madhura V
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

As part of the Compiler Design Project component, we have built a mini-compiler for C++ language. The compiler aims to cover the basic syntax, grammar, intermediate code generation and some optimisation techniques.

Sample input:

A complete C++ program consisting of different constructs like if, if-else and for along with basic syntax:

```
#include<iostream>

using namespace std;

/* Note this is a comment to check the correctness of the comment removal code in
Lex
  This comment spans multiple lines.
*/
// This is a single comment line.....

int main()
{
   // Variable declaration check
   int a = 5;
   int b = 6;
   int c = a * b;
   int d = c+b;
   int e = a*b+d-b+a;
   int f = a*5 + 8-4 + 7*3 +d*e/1;
   int g = (4+5)*f+e-d/c+b;
   int k = c + 4;
   g = d+4;

   int sum = 0;
   // Simple for loop
   for(int i=0;i<10;i++)
   {
      sum = sum + 1;
```

```
}

// Double nested for loop
for(int i=0;i<10;i++)
{
   for(int j=i;j<10;j++)
   {
      sum = sum + 1;
   }
}

// Simple if condition
if(a>b)
{
   g = c + 4;
}

// Simple if condition
if(sum==55)
{
   int k = 2;
}

// Multi-Nested if-else
if(a+b==c)
{
   if(b+c>=f)
   {
      k =2;
   }
   else
   {
      if(a-b>0)
      {
         k = 5;
      }
      else
      {
         k = 4;
      }
      g = 3;
```

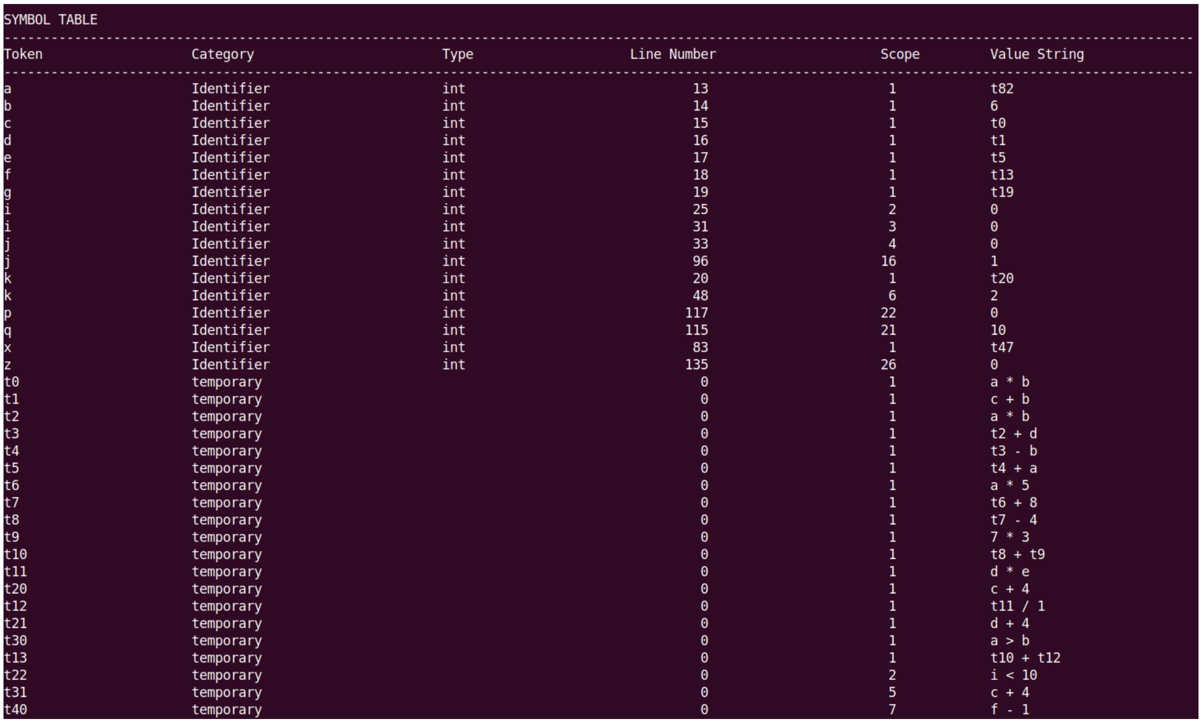
```
    }
    b = g + 1;
    e = f - 1;
}
else if(b+c==d)
{
    a = g - 1;
    b = f*2;
}
else
{
    a = g-2;
}

int x = 5;
x+=6;
x-=a;

int mx = 4+5-3;
mx = -mx;
mx = +5;

// constant expression as if condition
if(1+2)
    k = 1;

// if-else inside a for loop
for(int j=1;j<10;j++)
{
    if(a+b>c)
    {
        k = 1+2;
    }
    if(1)
        x = 1;
    else
        x = -1;
}

// Different type of if condition
if((a+b>x)&&(c+d>b))
```

```
{
   a =100;
}

// Nested for with if inside
for(int q=10;q>0;--q)
{
   for(int p=0;p<5;++p)
   {
      sum+=p+q;
      if(p+q>10)
      {
         a = a - 1;
      }
   }
}

// Simple if construct
if(a)
{
   a =20;
}

if((a+b>c))
{
   for(int z=0;z<6;z=z+5)
   {
      if(z)
      {
         a-=1;
      }
      else
         a+=1;
   }
}

// Special cases of for loops
for(;a>10;)
{
   a -=1;
}
```

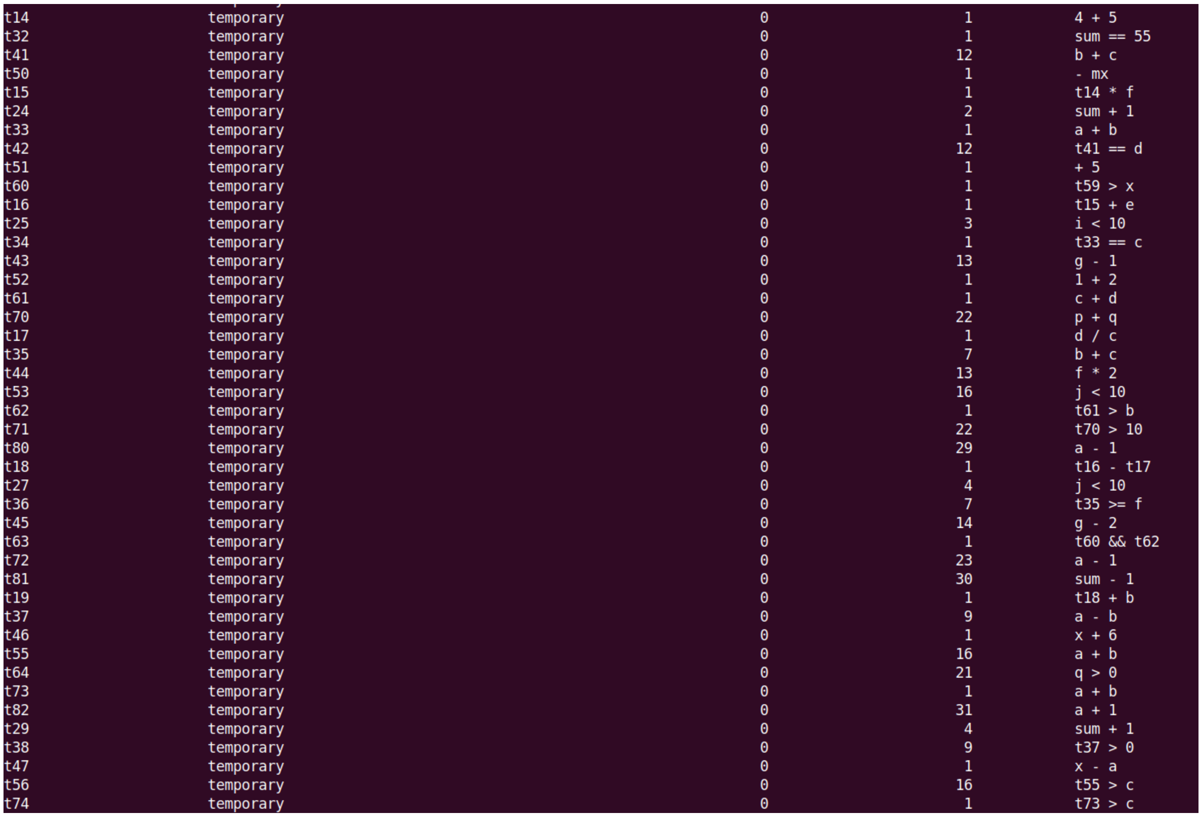
```
    for(;;)
    {
        sum-=1;
    }

    for(a=0;;a+=1)
    {
        sum+=1;
    }
}
```

Sample output:

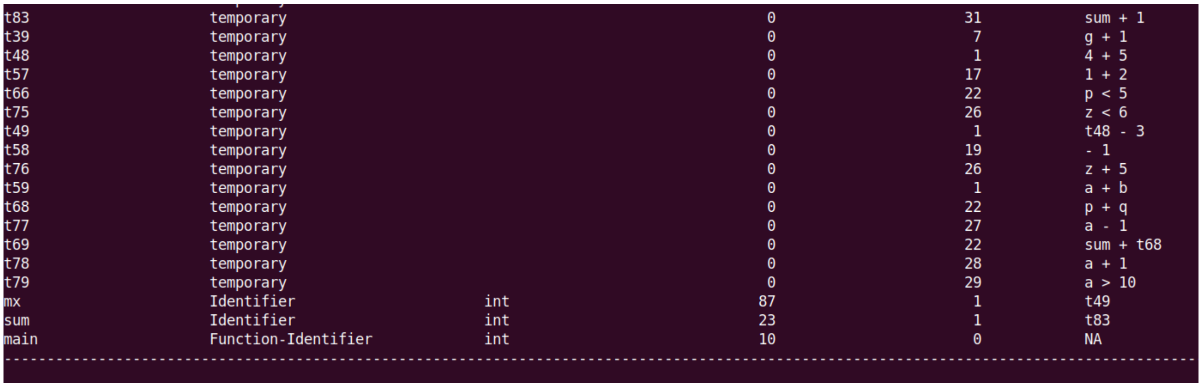
i. Symbol table with required information



```
SYMBOL TABLE
--------------------------------------------------------------------------------------------------------------------------------------------------------
Token                   Category                        Type                    Line Number                     Scope         Value String
--------------------------------------------------------------------------------------------------------------------------------------------------------
a                       Identifier                      int                             13                       1            t82
b                       Identifier                      int                             14                       1            6
c                       Identifier                      int                             15                       1            t0
d                       Identifier                      int                             16                       1            t1
e                       Identifier                      int                             17                       1            t5
f                       Identifier                      int                             18                       1            t13
g                       Identifier                      int                             19                       1            t19
i                       Identifier                      int                             25                       2            0
i                       Identifier                      int                             31                       3            0
j                       Identifier                      int                             33                       4            0
j                       Identifier                      int                             96                      16            1
k                       Identifier                      int                             20                       1            t20
k                       Identifier                      int                             48                       6            2
p                       Identifier                      int                            117                      22            0
q                       Identifier                      int                            115                      21            10
x                       Identifier                      int                             83                       1            t47
z                       Identifier                      int                            135                      26            0
t0                      temporary                                                        0                       1            a * b
t1                      temporary                                                        0                       1            c + b
t2                      temporary                                                        0                       1            a * b
t3                      temporary                                                        0                       1            t2 + d
t4                      temporary                                                        0                       1            t3 - b
t5                      temporary                                                        0                       1            t4 + a
t6                      temporary                                                        0                       1            a * 5
t7                      temporary                                                        0                       1            t6 + 8
t8                      temporary                                                        0                       1            t7 - 4
t9                      temporary                                                        0                       1            7 * 3
t10                     temporary                                                        0                       1            t8 + t9
t11                     temporary                                                        0                       1            d * e
t20                     temporary                                                        0                       1            c + 4
t12                     temporary                                                        0                       1            t11 / 1
t21                     temporary                                                        0                       1            d + 4
t30                     temporary                                                        0                       1            a > b
t13                     temporary                                                        0                       1            t10 + t12
t22                     temporary                                                        0                       2            i < 10
t31                     temporary                                                        0                       5            c + 4
t40                     temporary                                                        0                       7            f - 1
```



```
t14                     temporary                                    0                    1           4 + 5
t32                     temporary                                    0                    1           sum == 55
t41                     temporary                                    0                   12           b + c
t50                     temporary                                    0                    1           - mx
t15                     temporary                                    0                    1           t14 * f
t24                     temporary                                    0                    2           sum + 1
t33                     temporary                                    0                    1           a + b
t42                     temporary                                    0                   12           t41 == d
t51                     temporary                                    0                    1           + 5
t60                     temporary                                    0                    1           t59 > x
t16                     temporary                                    0                    1           t15 + e
t25                     temporary                                    0                    3           i < 10
t34                     temporary                                    0                    1           t33 == c
t43                     temporary                                    0                   13           g - 1
t52                     temporary                                    0                    1           1 + 2
t61                     temporary                                    0                    1           c + d
t70                     temporary                                    0                   22           p + q
t17                     temporary                                    0                    1           d / c
t35                     temporary                                    0                    7           b + c
t44                     temporary                                    0                   13           f * 2
t53                     temporary                                    0                   16           j < 10
t62                     temporary                                    0                    1           t61 > b
t71                     temporary                                    0                   22           t70 > 10
t80                     temporary                                    0                   29           a - 1
t18                     temporary                                    0                    1           t16 - t17
t27                     temporary                                    0                    4           j < 10
t36                     temporary                                    0                    7           t35 >= f
t45                     temporary                                    0                   14           g - 2
t63                     temporary                                    0                    1           t60 && t62
t72                     temporary                                    0                   23           a - 1
t81                     temporary                                    0                   30           sum - 1
t19                     temporary                                    0                    1           t18 + b
t37                     temporary                                    0                    9           a - b
t46                     temporary                                    0                    1           x + 6
t55                     temporary                                    0                   16           a + b
t64                     temporary                                    0                   21           q > 0
t73                     temporary                                    0                    1           a + b
t82                     temporary                                    0                   31           a + 1
t29                     temporary                                    0                    4           sum + 1
t38                     temporary                                    0                    9           t37 > 0
t47                     temporary                                    0                    1           x - a
t56                     temporary                                    0                   16           t55 > c
t74                     temporary                                    0                    1           t73 > c
```



```
t83                     temporary                                    0                   31           sum + 1
t39                     temporary                                    0                    7           g + 1
t48                     temporary                                    0                    1           4 + 5
t57                     temporary                                    0                   17           1 + 2
t66                     temporary                                    0                   22           p < 5
t75                     temporary                                    0                   26           z < 6
t49                     temporary                                    0                    1           t48 - 3
t58                     temporary                                    0                   19           - 1
t76                     temporary                                    0                   26           z + 5
t59                     temporary                                    0                    1           a + b
t68                     temporary                                    0                   22           p + q
t77                     temporary                                    0                   27           a - 1
t69                     temporary                                    0                   22           sum + t68
t78                     temporary                                    0                   28           a + 1
t79                     temporary                                    0                   29           a > 10
mx                      Identifier                int               87                    1           t49
sum                     Identifier                int               23                    1           t83
main                    Function-Identifier       int               10                    0           NA
-------------------------------------------------------------------------------------------------------------------
```

ii. Intermediate code (both in Three Address Code and Quadruple format)

Three Address Code:

a = 5
b = 6
t0 = a * b
c = t0
t1 = c + b

```
d = t1
t2 = a * b
t3 = t2 + d
t4 = t3 - b
t5 = t4 + a
e = t5
t6 = a * 5
t7 = t6 + 8
t8 = t7 - 4
t9 = 7 * 3
t10 = t8 + t9
t11 = d * e
t12 = t11 / 1
t13 = t10 + t12
f = t13
t14 = 4 + 5
t15 = t14 * f
t16 = t15 + e
t17 = d / c
t18 = t16 - t17
t19 = t18 + b
g = t19
t20 = c + 4
k = t20
t21 = d + 4
g = t21
sum = 0
i = 0
L0:
t22 = i < 10
if t22 goto L1
goto L2
L3:
t23 = i + 1
i = t23
goto L0
L1:
t24 = sum + 1
sum = t24
goto L3
L2:
```

```
i = 0
L4:
t25 = i < 10
if t25 goto L5
goto L6
L7:
t26 = i + 1
i = t26
goto L4
L5:
j = i
L8:
t27 = j < 10
if t27 goto L9
goto L10
L11:
t28 = j + 1
j = t28
goto L8
L9:
t29 = sum + 1
sum = t29
goto L11
L10:
goto L7
L6:
t30 = a > b
if t30 goto L12
goto L13
L12:
t31 = c + 4
g = t31
goto L14
L13:
L14:
t32 = sum == 55
if t32 goto L15
goto L16
L15:
k = 2
goto L17
```

```
L16:
L17:
t33 = a + b
t34 = t33 == c
if t34 goto L18
goto L19
L18:
t35 = b + c
t36 = t35 >= f
if t36 goto L20
goto L21
L20:
k = 2
goto L22
L21:
t37 = a - b
t38 = t37 > 0
if t38 goto L23
goto L24
L23:
k = 5
goto L25
L24:
k = 4
L25:
g = 3
L22:
t39 = g + 1
b = t39
t40 = f - 1
e = t40
goto L26
L19:
t41 = b + c
t42 = t41 == d
if t42 goto L27
goto L28
L27:
t43 = g - 1
a = t43
t44 = f * 2
```

```
b = t44
goto L29
L28:
t45 = g - 2
a = t45
L29:
L26:
x = 5
t46 = x + 6
x = t46
t47 = x - a
x = t47
t48 = 4 + 5
t49 = t48 - 3
mx = t49
t50 = - mx
mx = t50
t51 = + 5
mx = t51
t52 = 1 + 2
if t52 goto L30
goto L31
L30:
k = 1
goto L32
L31:
L32:
j = 1
L33:
t53 = j < 10
if t53 goto L34
goto L35
L36:
t54 = j + 1
j = t54
goto L33
L34:
t55 = a + b
t56 = t55 > c
if t56 goto L37
goto L38
```

```
L37:
t57 = 1 + 2
k = t57
goto L39
L38:
L39:
if 1 goto L40
goto L41
L40:
x = 1
goto L42
L41:
t58 = - 1
x = t58
L42:
goto L36
L35:
t59 = a + b
t60 = t59 > x
t61 = c + d
t62 = t61 > b
t63 = t60 && t62
if t63 goto L43
goto L44
L43:
a = 100
goto L45
L44:
L45:
q = 10
L46:
t64 = q > 0
if t64 goto L47
goto L48
L49:
t65 = q - 1
q = t65
goto L46
L47:
p = 0
L50:
```

```
t66 = p < 5
if t66 goto L51
goto L52
L53:
t67 = p + 1
p = t67
goto L50
L51:
t68 = p + q
t69 = sum + t68
sum = t69
t70 = p + q
t71 = t70 > 10
if t71 goto L54
goto L55
L54:
t72 = a - 1
a = t72
goto L56
L55:
L56:
goto L53
L52:
goto L49
L48:
if a goto L57
goto L58
L57:
a = 20
goto L59
L58:
L59:
t73 = a + b
t74 = t73 > c
if t74 goto L60
goto L61
L60:
z = 0
L62:
t75 = z < 6
if t75 goto L63
```

```
goto L64
L65:
t76 = z + 5
z = t76
goto L62
L63:
if z goto L66
goto L67
L66:
t77 = a - 1
a = t77
goto L68
L67:
t78 = a + 1
a = t78
L68:
goto L65
L64:
goto L69
L61:
L69:
L70:
t79 = a > 10
if t79 goto L71
goto L72
L73:
goto L70
L71:
t80 = a - 1
a = t80
goto L73
L72:
L74:
goto L75
L77:
goto L74
L75:
t81 = sum - 1
sum = t81
goto L77
L76:
```

```
a = 0
L78:
goto L79
L81:
t82 = a + 1
a = t82
goto L78
L79:
t83 = sum + 1
sum = t83
goto L81
L80:
```

Quadruple Format:

```
QUADRUPLES
Op    arg1  arg2  res
=     5           a
=     6           b
*     a     b     t0
=     t0          c
+     c     b     t1
=     t1          d
*     a     b     t2
+     t2    d     t3
-     t3    b     t4
+     t4    a     t5
=     t5          e
*     a     5     t6
+     t6    8     t7
-     t7    4     t8
*     7     3     t9
+     t8    t9    t10
*     d     e     t11
/     t11   1     t12
+     t10   t12   t13
=     t13         f
+     4     5     t14
*     t14   f     t15
+     t15   e     t16
/     d     c     t17
```

```
-       t16     t17     t18
+       t18     b       t19
=       t19             g
+       c       4       t20
=       t20             k
+       d       4       t21
=       t21             g
=       0               sum
=       0               i
L0                      label
<       i       10      t22
if      t22             L1
goto                    L2
L3                      label
+       i       1       t23
=       t23             i
goto                    L0
L0                      label
+       sum     1       t24
=       t24             sum
goto                    L3
L2                      label
=       0               i
L4                      label
<       i       10      t25
if      t25             L5
goto                    L6
L7                      label
+       i       1       t26
=       t26             i
goto                    L4
L4                      label
=       i               j
L8                      label
<       j       10      t27
if      t27             L9
goto                    L10
L11                     label
+       j       1       t28
=       t28             j
goto                    L8
```

```
L8                  label
+       sum     1   t29
=       t29         sum
goto                L11
L10                 label
goto                L7
L6                  label
>       a       b   t30
L12     t30         if
goto                L13
L12                 label
+       c       4   t31
=       t31         g
goto                L14
L13                 label
L14                 label
==      sum     55  t32
L15     t32         if
goto                L16
L15                 label
=       2           k
goto                L17
L16                 label
L17                 label
+       a       b   t33
==      t33     c   t34
L18     t34         if
goto                L19
L18                 label
+       b       c   t35
>=      t35     f   t36
L20     t36         if
goto                L21
L20                 label
=       2           k
goto                L22
L21                 label
-       a       b   t37
>       t37     0   t38
L23     t38         if
goto                L24
```

```
L23                     label
=       5               k
goto                    L25
L24                     label
=       4               k
L25                     label
=       3               g
L22                     label
+       g       1       t39
=       t39             b
-       f       1       t40
=       t40             e
goto                    L26
L19                     label
+       b       c       t41
==      t41     d       t42
L27     t42             if
goto                    L28
L27                     label
-       g       1       t43
=       t43             a
*       f       2       t44
=       t44             b
goto                    L29
L28                     label
-       g       2       t45
=       t45             a
L29                     label
L26                     label
=       5               x
+       x       6       t46
=       x               t46
-       x       a       t47
=       x               t47
+       4       5       t48
-       t48     3       t49
=       t49             mx
t50     mx              -
=       t50             mx
t51     5               +
=       t51             mx
```

```
+       1       2       t52
L30     t52             if
goto                    L31
L30                     label
=       1               k
goto                    L32
L31                     label
L32                     label
=       1               j
L33                     label
<       j       10      t53
if      t53             L34
goto                    L35
L36                     label
+       j       1       t54
=       t54             j
goto                    L33
L33                     label
+       a       b       t55
>       t55     c       t56
L37     t56             if
goto                    L38
L37                     label
+       1       2       t57
=       t57             k
goto                    L39
L38                     label
L39                     label
L40     1               if
goto                    L41
L40                     label
=       1               x
goto                    L42
L41                     label
t58     1               -
=       t58             x
L42                     label
goto                    L36
L35                     label
+       a       b       t59
>       t59     x       t60
```

```
+       c       d       t61
>       t61     b       t62
&&      t60     t62     t63
L43     t63             if
goto                    L44
L43                     label
=       100             a
goto                    L45
L44                     label
L45                     label
=       10              q
L46                     label
>       q       0       t64
if      t64             L47
goto                    L48
L49                     label
-       q       1       t65
=       t65             q
goto                    L46
L46                     label
=       0               p
L50                     label
<       p       5       t66
if      t66             L51
goto                    L52
L53                     label
+       p       1       t67
=       t67             p
goto                    L50
L50                     label
+       p       q       t68
+       sum     t68     t69
=       sum             t69
+       p       q       t70
>       t70     10      t71
L54     t71             if
goto                    L55
L54                     label
-       a       1       t72
=       t72             a
goto                    L56
```

```
L55                     label
L56                     label
goto                    L53
L52                     label
goto                    L49
L48                     label
L57     a               if
goto                    L58
L57                     label
=       20              a
goto                    L59
L58                     label
L59                     label
+       a       b       t73
>       t73     c       t74
L60     t74             if
goto                    L61
L60                     label
=       0               z
L62                     label
<       z       6       t75
if      t75             L63
goto                    L64
L65                     label
+       z       5       t76
=       t76             z
goto                    L62
L62                     label
L66     z               if
goto                    L67
L66                     label
-       a       1       t77
=       a               t77
goto                    L68
L67                     label
+       a       1       t78
=       a               t78
L68                     label
goto                    L65
L64                     label
goto                    L69
```

```
L61                 label
L69                 label
L70                 label
>       a     10    t79
if      t79         L71
goto                L72
L73                 label
goto                L70
L70                 label
-       a     1     t80
=       a           t80
goto                L73
L72                 label
L74                 label
goto                L75
L77                 label
goto                L74
L74                 label
-       sum   1     t81
=       sum         t81
goto                L77
L76                 label
=       0           a
L78                 label
goto                L79
L81                 label
+       a     1     t82
=       a           t82
goto                L78
L78                 label
+       sum   1     t83
=       sum         t83
goto                L81
L80                 label
```

iii. Optimized Intermediate code (in Quadruple format)

QUADRUPLES

```
op      arg1    arg2    result

=       5               a
=       6               b
=       30              t0
=       36              t1
=       66              t3
=       60              t4
=       65              t5
=       25              t6
=       33              t7
=       29              t8
=       21              t9
=       50              t10
=       2340            t11
=       2340.0          t12
=       2390.0          t13
=       9               t14
=       21510.0                 t15
=       21575.0                 t16
=       1.2             t17
=       21573.8                 t18
=       21579.8                 t19
=       34              t20
=       40              t21
L0                      label
=       True            t22
if      t22             L1
goto                    L2
L3                      label
L0                      label
L2                      label
=       0               t23
L7                      label
L4                      label
L11                     label
L8                      label
L10                     label
L6                      label
=       False           t30
L12     t30             if
```

```
L12              label
=     34         t21
L13              label
L14              label
L15   t30        if
L15              label
L16              label
L17              label
=     11         t33
L18   t30        if
L18              label
L20   t30        if
L20              label
L21              label
=     -1         t37
L23   t30        if
L23              label
L24              label
=     4          a
L25              label
=     3          t21
L22              label
=     4          b
=     2389.0     t40
L19              label
L27   t30        if
L27              label
=     4780.0     t44
L28              label
L29              label
L26              label
=     6          mx
t50   6          -
=     t50        mx
t51   5          +
=     t51        mx
L30   3          if
L30              label
L31              label
L32              label
L36              label
```

```
L33                     label
=       4781.0          t55
L37     t22             if
L37                     label
=       3               t28
L38                     label
L39                     label
L40     1               if
L40                     label
L41                     label
t58     1               -
=       t58             t28
L42                     label
L35                     label
>       4781.0t28       t60
&&      t60     t30     t63
L43     t63             if
L43                     label
=       100             t28
L44                     label
L45                     label
=       10              q
L49                     label
=       9               q
L46                     label
L53                     label
=       1               t28
L50                     label
=       12              t69
L54     t30             if
L54                     label
=       99              t72
L55                     label
L56                     label
L52                     label
L48                     label
L57     99              if
L57                     label
=       20              t72
L58                     label
L59                     label
```

```
=       4800.0      t73
L60     t22         if
L60                 label
L65                 label
=       5           t72
L62                 label
L66     5           if
L66                 label
=       19          t77
L67                 label
L68                 label
L64                 label
L61                 label
L69                 label
L73                 label
L70                 label
L72                 label
L77                 label
L74                 label
=       2           t72
L76                 label
L81                 label
L78                 label
L80                 label
```

2. ARCHITECTURE OF LANGUAGE

The designed mini compiler aims to handle basic working syntax of C++ programs along with specific constructs.

In terms of syntax, the following cases have been handled:

- Single line and multiline comments
- Incomplete multiline comments resulting in error message generation
- Recognition of multiple keywords like return, void, class, public, private, protected, int, float, double, bool, if, else, for, cin, cout, printf, scanf, break, continue, exit, string, char, true, false, etc.
- Recognition of valid identifiers (limited to maximum 32 characters)

- Conversion of exponential notation floating point numbers (like 3.14E10) to standard decimal notation floating point numbers
- Preprocessor directives
- Functions (with prototype, declaration and definition)
- Single-line if construct
- Block if constructs
- Block if else construct
- Single line for construct
- Block for construct
- All arithmetic operators (+, -, *, /, %)
- All bitwise operators (&, |, ^)
- All logical and relational operators
- Multiple cases of Assignment expressions
- Jump statements

In terms of semantics, the following cases have been handled:

- Usage of undeclared variables
- Implicit type casting between primitive data types
- Incompatible type assignments
- Incompatible type operations
- Illegal redeclarations
- Occurence of break statements at only appropriate places (inside loop bodies)

3. LITERATURE SURVEY

- Lex and Yacc Anchor material provided by PES University Compiler Design Faculty
- Regular Expressions online website (https://www.regular-expressions.info/)
- ISO C++ reference (https://isocpp.org/wiki/faq/compiler-dependencies#yaccable-grammar)
- C++ yacc-able grammar reference (http://www.computing.surrey.ac.uk/research/dsrg/fog/CxxGrammar.y)
- Lecture slides on Intermediate Code generation and Optimisation techniques provided by PES University Compiler Design Faculty

4. CONTEXT FREE GRAMMAR

Following is the C++ context free grammar written by us in the yacc file:

```
START
        : INCLUDE BODY
        | BODY
        | INCLUDE
        ;

INCLUDE
        : INCLUDE T_HEADER_INCLUDE '<' T_HEADER_FILE '>'
        | INCLUDE T_HEADER_INCLUDE T_STRING_LITERAL
        | T_HEADER_INCLUDE '<' T_HEADER_FILE '>'
        | T_HEADER_INCLUDE T_STRING_LITERAL
        ;

BODY
        : BODY_BLOCK BODY
        | BODY_BLOCK
        ;

BODY_BLOCK
        : FUNCTION
        | BLOCK
        ;

FUNCTION
        : FUNCTION_PROTOTYPE
        | FUNCTION_DEFINITION
        | FUNCTION_DECLARATION
        ;

FUNCTION_PROTOTYPE
        : FUNCTION_PREFIX TYPE_LIST ')' ';'
        | FUNCTION_PREFIX ')' ';'
        ;

TYPE_LIST
        : TYPE ',' TYPE_LIST
        | TYPE
        ;

FUNCTION_DEFINITION
        : FUNCTION_PREFIX FUNCTION_PARAMETER_LIST ')' ';' {
                scope_leave();
        }
        ;

FUNCTION_DECLARATION
        : FUNCTION_PREFIX FUNCTION_PARAMETER_LIST ')' '{'
STATEMENTS '}' {
                scope_leave();
        }
        | FUNCTION_PREFIX ')' '{' STATEMENTS '}' {
                scope_leave();
        }
        ;

FUNCTION_PARAMETER_LIST
        : TYPE T_IDENTIFIER ',' FUNCTION_PARAMETER_LIST {
                if (insert($2, "Identifier", $1, @2.last_line, NULL) == NULL) {
                        printf("[Error] at line:%d - Function Parameter \"%s\" has
already been declared\n", @2.last_line, $2);
                }
        }
        | TYPE T_IDENTIFIER '=' EXPRESSION ','
FUNCTION_PARAMETER_LIST {
                if (insert($2, "Identifier", $1, @2.last_line, NULL) == NULL) {
                        printf("[Error] at line:%d - Function Parameter \"%s\" has
already been declared\n", @2.last_line, $2);
                }
      symbol_table* element = lookup($2);
                strcpy(element->value, $4);
        }
        | TYPE T_IDENTIFIER {
                if (insert($2, "Identifier", $1, @2.last_line, NULL) == NULL) {
                        printf("[Error] at line:%d - Function Parameter \"%s\" has
already been declared\n", @2.last_line, $2);
                }
        }
```

```
    | TYPE T_IDENTIFIER '=' EXPRESSION {
        if (insert($2, "Identifier", $1, @2.last_line, NULL) == NULL) {
            printf("[Error] at line:%d - Function Parameter \"%s\" has already been declared\n", @2.last_line, $2);
        }
   symbol_table* element = lookup($2);
        strcpy(element->value, $4);
    }
    ;

FUNCTION_PREFIX
    : TYPE T_IDENTIFIER '(' {
        insert($2, "Function-Identifier", $1, @2.last_line, NULL);
        scope_enter();
    }
    ;

BLOCK
    : BLOCK_START STATEMENTS BLOCK_END
    ;

BLOCK_START
    : '{' {
   scope_enter();
  }
    ;

BLOCK_END
    : '}' {
   scope_leave();
  }
    ;

STATEMENTS
    : STATEMENT STATEMENTS
    | STATEMENT
    ;

SINGLE_LINE_IF
    : IF_PREFIX LINE_STATEMENT ';' {
        scope_leave();
    }
    | IF_PREFIX ';' {
        scope_leave();
    }
    | IF_PREFIX CONSTRUCT {
        scope_leave();
    }
    | SINGLE_LINE_IF SINGLE_LINE_ELSE
    | SINGLE_LINE_IF BLOCK_ELSE
    ;

BLOCK_IF
    : T_CONSTRUCT_IF '(' EXPRESSION ')' BLOCK
    | BLOCK_IF SINGLE_LINE_ELSE
    | BLOCK_IF BLOCK_ELSE
    ;

IF_PREFIX
    : T_CONSTRUCT_IF '(' EXPRESSION ')' {
        scope_enter();
    }
    ;

SINGLE_LINE_ELSE
    : ELSE_PREFIX LINE_STATEMENT ';'{
        scope_leave();
    }
    | ELSE_PREFIX ';'{
        scope_leave();
    }
    | ELSE_PREFIX CONSTRUCT {
        scope_leave();
    }
    ;

BLOCK_ELSE
    : T_CONSTRUCT_ELSE BLOCK
    ;

ELSE_PREFIX
    : T_CONSTRUCT_ELSE {
```

```
                scope_enter();
        }
        ;

SINGLE_LINE_FOR
        : FOR_PREFIX FOR_INIT_STATEMENT ';'
FOR_CONDITION_STATEMENT ';' FOR_ACTION_STATEMENT ')'
LINE_STATEMENT ';'{
                scope_leave();
                is_in_construct -= 1;
        }
        | FOR_PREFIX FOR_INIT_STATEMENT ';'
FOR_CONDITION_STATEMENT ';' FOR_ACTION_STATEMENT ')' ';' {
                scope_leave();
                is_in_construct -= 1;
        }
        | FOR_PREFIX FOR_INIT_STATEMENT ';'
FOR_CONDITION_STATEMENT ';' FOR_ACTION_STATEMENT ')'
CONSTRUCT {
                scope_leave();
                is_in_construct -= 1;
        }
        ;

BLOCK_FOR
        : FOR_PREFIX FOR_INIT_STATEMENT ';'
FOR_CONDITION_STATEMENT ';' FOR_ACTION_STATEMENT ')' '{'
STATEMENTS '}'{
                scope_leave();
                is_in_construct -= 1;
        }
        ;

FOR_PREFIX
        : T_CONSTRUCT_FOR '(' {
                scope_enter();
                is_in_construct += 1;
        }
        ;

FOR_INIT_STATEMENT
```

```
      :
      | LINE_STATEMENT
      ;

FOR_CONDITION_STATEMENT
      :
      | CONDITIONAL_EXPRESSION
      ;

FOR_ACTION_STATEMENT
      :
      | LINE_STATEMENT
      ;

BITWISE_OPERATOR
      : '&' {
    $$ = strdup($1);
  }
      | '|' {
    $$ = strdup($1);
  }
      | '^' {
    $$ = strdup($1);
  }
      ;

CONDITIONAL_EXPRESSION
      : EXPRESSION LOGICAL_OPERATOR EXPRESSION_GRAMMAR {
    sprintf($$, "%s %s %s", $1, $2, $3);
    $$ = strdup($$);
  }
      | EXPRESSION RELATIONAL_OPERATOR EXPRESSION_GRAMMAR {
    sprintf($$, "%s %s %s", $1, $2, $3);
    $$ = strdup($$);
  }
      | EXPRESSION BITWISE_OPERATOR EXPRESSION_GRAMMAR {
    sprintf($$, "%s %s %s", $1, $2, $3);
    $$ = strdup($$);
  }
      ;
```

```
ASSIGNMENT
        : T_IDENTIFIER ASSIGNMENT_OPERATOR EXPRESSION_GRAMMAR
{
                if (lookup($1) == NULL) {
                        printf("[Error] at line:%d - Undeclared Variable \"%s\" \n",
@1.last_line, $1);
                }
      sprintf($$, "%s %s %s", $1, $2, $3);
      $$ = strdup($$);
        }
        | T_IDENTIFIER ASSIGNMENT_OPERATOR ASSIGNMENT {
                if (lookup($1) == NULL) {
                        printf("[Error] at line:%d - Undeclared Variable \"%s\" \n",
@1.last_line, $1);
                }
      sprintf($$, "%s %s %s", $1, $2, $3);
      $$ = strdup($$);
        }
        | T_IDENTIFIER '[' EXPRESSION ']' ASSIGNMENT_OPERATOR
EXPRESSION_GRAMMAR {
                if (lookup($1) == NULL) {
                        printf("[Error] at line:%d - Undeclared Variable \"%s\" \n",
@1.last_line, $1);
                }
      sprintf($$, "%s [ %s ] %s %s", $1, $3, $5, $6);
      $$ = strdup($$);
        }
        | T_IDENTIFIER '[' EXPRESSION ']' ASSIGNMENT_OPERATOR
ASSIGNMENT {
                if (lookup($1) == NULL) {
                        printf("[Error] at line:%d - Undeclared Variable \"%s\" \n",
@1.last_line, $1);
                }
      sprintf($$, "%s [ %s ] %s %s", $1, $3, $5, $6);
      $$ = strdup($$);
        }
        ;

ASSIGNMENT_OPERATOR
        : '=' {
      $$ = strdup($1);
```

```
    }
        | T_OP_ADD_ASSIGNMENT {
      $$ = strdup($1);
    }
        | T_OP_SUBTRACT_ASSIGNMENT {
      $$ = strdup($1);
    }
        | T_OP_MULTIPLY_ASSIGNMENT {
      $$ = strdup($1);
    }
        | T_OP_DIVIDE_ASSIGNMENT {
      $$ = strdup($1);
    }
        | T_OP_MOD_ASSIGNMENT {
      $$ = strdup($1);
    }
        ;

EXPRESSION
        : ASSIGNMENT {
      $$ = strdup($1);
    }
        | CONDITIONAL_EXPRESSION {
      $$ = strdup($1);
    }
        | EXPRESSION_GRAMMAR {
      $$ = strdup($1);
    }
        ;

EXPRESSION_GRAMMAR
        : EXPRESSION_GRAMMAR '+' EXPRESSION_TERM {
      sprintf($$, "%s + %s", $1, $3);
      $$ = strdup($$);
    }
        | EXPRESSION_GRAMMAR '-' EXPRESSION_TERM {
      sprintf($$, "%s - %s", $1, $3);
      $$ = strdup($$);
    }
        | EXPRESSION_TERM {
      $$ = strdup($1);
```

```
  }
        ;

EXPRESSION_TERM
        : EXPRESSION_TERM '*' EXPRESSION_F {
    sprintf($$, "%s * %s", $1, $3);
    $$ = strdup($$);
  }
        | EXPRESSION_TERM '/' EXPRESSION_F {
    sprintf($$, "%s / %s", $1, $3);
    $$ = strdup($$);
  }
        | EXPRESSION_TERM '%' EXPRESSION_F {
    sprintf($$, "%s %% %s", $1, $3);
    $$ = strdup($$);
  }
        | EXPRESSION_F {
    $$ = strdup($1);
  }
        | '!' EXPRESSION_F {
    sprintf($$, "! %s", $2);
    $$ = strdup($$);
  }
        ;

EXPRESSION_F
        : IDENTIFIER_OR_LITERAL {
    $$ = strdup($1);
  }
        | '(' EXPRESSION ')' {
    sprintf($$, "( %s )", $2);
    $$ = strdup($$);
  }
        | '+' EXPRESSION_F {
    sprintf($$, "+ %s", $2);
    $$ = strdup($$);
  }
        | '-' EXPRESSION_F {
    sprintf($$, "- %s", $2);
    $$ = strdup($$);
  }
```

```
        ;

CONSTRUCT
        : SINGLE_LINE_CONSTRUCT
        | BLOCK_CONSTRUCT
        ;

BLOCK_CONSTRUCT
        : BLOCK_FOR
        | BLOCK_IF %prec IF_PREC
        ;

SINGLE_LINE_CONSTRUCT
        : SINGLE_LINE_FOR
        | SINGLE_LINE_IF %prec IF_PREC
        ;

STATEMENT
        : LINE_STATEMENT ';'
        | CONSTRUCT
        | BLOCK
        | ';'
        ;

JUMP_STATEMENT
        : T_JUMP_BREAK {
                if (is_in_construct == 0)
                        printf("[Error] at line:%d - \"break\" statement not within loop or
switch\n", @1.last_line);
        }
        | T_JUMP_EXIT
        | T_JUMP_CONTINUE
        ;

LINE_STATEMENT
        : VARIABLE_DECLARATION
        | EXPRESSION
        | COUT
        | CIN
        | RETURN
        | JUMP_STATEMENT
        ;

VARIABLE_DECLARATION
        : VARIABLE_DECLARATION_TYPE VARIABLE_LIST {
                strcpy(variable_declaration_type, "\0");
      sprintf($$, "%s %s", $1, $2);
      $$ = strdup($$);
        }
        ;

VARIABLE_DECLARATION_TYPE
        : TYPE {
                strcpy(variable_declaration_type, $1);
      $$ = strdup($1);
        }
        ;

VARIABLE_LIST
        : VARIABLE_DECLARATION_IDENTIFIER ',' VARIABLE_LIST {
      sprintf($$, "%s , %s", $1, $3);
      $$ = strdup($$);
   }
        | VARIABLE_DECLARATION_IDENTIFIER '=' EXPRESSION ','
VARIABLE_LIST {
                symbol_table* element = lookup($1);
                sprintf(element->value, "%s", $3);
      sprintf($$, "%s = %s , %s", $1, $3, $5);
      $$ = strdup($$);
        }
        | VARIABLE_DECLARATION_IDENTIFIER {
      $$ = strdup($1);
   }
        | VARIABLE_DECLARATION_IDENTIFIER '=' EXPRESSION {
                symbol_table* element = lookup($1);
                strcpy(element->value, $3);
      sprintf($$, "%s = %s", $1, $3);
      $$ = strdup($$);
        }

        | ARRAY_VARIABLE_DECLARATION_IDENTIFIER_WITH_SIZE ','
VARIABLE_LIST {
      sprintf($$, "%s , %s", $1, $3);
      $$ = strdup($$);
   }
        | ARRAY_VARIABLE_DECLARATION_IDENTIFIER '=' ARRAY_LIST ','
VARIABLE_LIST {
      sprintf($$, "%s = %s , %s", $1, $3, $5);
      $$ = strdup($$);
   }
        | ARRAY_VARIABLE_DECLARATION_IDENTIFIER_WITH_SIZE {
      $$ = strdup($1);
   }
        | ARRAY_VARIABLE_DECLARATION_IDENTIFIER '=' ARRAY_LIST  {
      sprintf($$, "%s = %s", $1, $3);
      $$ = strdup($$);
   }
        ;

VARIABLE_DECLARATION_IDENTIFIER
        : T_IDENTIFIER {
                if (insert($1, "Identifier", variable_declaration_type, @1.last_line,
NULL) == NULL) {
                        printf("[Error] at line:%d - \"%s\" has already been declared\n",
@1.last_line, $1);
                }
      $$ = strdup($1);
        }
        ;

ARRAY_VARIABLE_DECLARATION_IDENTIFIER
        : T_IDENTIFIER '[' ']' {
                if (insert($1, "Identifier-Array", variable_declaration_type,
@1.last_line, NULL) == NULL) {
                        printf("[Error] at line:%d - \"%s\" has already been declared\n",
@1.last_line, $1);
                }
      sprintf($$, "%s []", $1);
      $$ = strdup($$);
        }
        | T_IDENTIFIER '[' EXPRESSION ']' {
                if (insert($1, "Identifier-Array", variable_declaration_type,
@1.last_line, NULL) == NULL) {
```

```
                  printf("[Error] at line:%d - \"%s\" has already been declared\n",
@1.last_line, $1);
            }
    sprintf($$, "%s [ %s ]", $1, $3);
    $$ = strdup($$);
      }
      ;

ARRAY_VARIABLE_DECLARATION_IDENTIFIER_WITH_SIZE
      : T_IDENTIFIER '[' EXPRESSION ']' {
            if (insert($1, "Identifier-Array", variable_declaration_type,
@1.last_line, NULL) == NULL) {
                  printf("[Error] at line:%d - \"%s\" has already been declared\n",
@1.last_line, $1);
            }
    sprintf($$, "%s [ %s ]", $1, $3);
    $$ = strdup($$);
      }
      ;

ARRAY_LIST
      : '{' LITERAL_LIST '}' {
    sprintf($$, "{ %s }", $2);
    $$ = strdup($$);
  }
      | T_STRING_LITERAL {
    $$ = strdup($1);
  }
      ;

LITERAL_LIST
      : IDENTIFIER_OR_LITERAL ',' LITERAL_LIST {
    sprintf($$, "%s , %s", $1, $3);
    $$ = strdup($$);
  }
      | IDENTIFIER_OR_LITERAL {
    $$ = strdup($1);
  }
      ;

COUT
```

```
        : T_IO_COUT T_IO_INSERTION INSERTION_LIST {
      sprintf($$, "%s %s %s", $1, $2, $3);
      $$ = strdup($$);
   }
        ;

INSERTION_LIST
        : EXPRESSION T_IO_INSERTION INSERTION_LIST {
      sprintf($$, "%s %s %s", $1, $2, $3);
      $$ = strdup($$);
   }
        | EXPRESSION {
      $$ = strdup($1);
   }
        ;

CIN
        : T_IO_CIN T_IO_EXTRACTION EXTRACTION_LIST {
      sprintf($$, "%s %s %s", $1, $2, $3);
      $$ = strdup($$);
   }
        ;

EXTRACTION_LIST
        : T_IDENTIFIER T_IO_EXTRACTION EXTRACTION_LIST {
                if (lookup($1) == NULL) {
                        printf("[Error] at line:%d - Undeclared variable \"%s\" \n",
@1.last_line, $1);
                }
      sprintf($$, "%s %s %s", $1, $2, $3);
      $$ = strdup($$);
        }
        | T_IDENTIFIER {
                if (lookup($1) == NULL) {
                        printf("[Error] at line:%d - Undeclared variable \"%s\" \n",
@1.last_line, $1);
                }
                $$ = strdup($1);
        }
        ;
```

```
RETURN
        : T_RETURN EXPRESSION {
      sprintf($$, "%s %s", $1, $2);
      $$ = strdup($$);
   }
        ;

LOGICAL_OPERATOR
        : T_LOG_OP_AND {
      $$ = strdup($1);
   }
        | T_LOG_OP_OR {
      $$ = strdup($1);
   }
        ;

RELATIONAL_OPERATOR
        : T_REL_OP_EQUAL {
      $$ = strdup($1);
   }
        | '>' {
      $$ = strdup($1);
   }
        | T_REL_OP_GREATER_THAN_EQUAL {
      $$ = strdup($1);
   }
        | '<' {
      $$ = strdup($1);
   }
        | T_REL_OP_LESS_THAN_EQUAL {
      $$ = strdup($1);
   }
        | T_REL_OP_NOT_EQUAL {
      $$ = strdup($1);
   }
        ;

IDENTIFIER_OR_LITERAL
        : T_IDENTIFIER {
                if (lookup($1) == NULL) {
                        printf("[Error] at line:%d - Undeclared variable \"%s\" \n",
@1.last_line, $1);
                }
                $$ = strdup($1);
        }
        | T_IDENTIFIER '(' ')' {
                if (lookup($1) == NULL) {
                        printf("[Error] at line:%d - Function \"%s\" not defined \n",
@1.last_line, $1);
                }
                sprintf($$, "%s ()", $1);
      $$ = strdup($$);
        }
        | T_IDENTIFIER '(' LITERAL_LIST ')' {
                if (lookup($1) == NULL) {
                        printf("[Error] at line:%d - Function \"%s\" not defined \n",
@1.last_line, $1);
                }
                sprintf($$, "%s ( %s )", $1, $3);
      $$ = strdup($$);
        }
        | T_IDENTIFIER UNARY_OPERATOR {
                if (lookup($1) == NULL) {
                        printf("[Error] at line:%d - Undeclared variable \"%s\" \n",
@1.last_line, $1);
                }
                sprintf($$, "%s %s", $1, $2);
      $$ = strdup($$);
        }
        | UNARY_OPERATOR T_IDENTIFIER {
                if (lookup($2) == NULL) {
                        printf("[Error] at line:%d - Undeclared variable \"%s\" \n",
@2.last_line, $2);
                }
                sprintf($$, "%s %s", $1, $2);
      $$ = strdup($$);
        }
        | T_IDENTIFIER '[' EXPRESSION ']' {
                if (lookup($1) == NULL) {
                        printf("[Error] at line:%d - Undeclared variable \"%s\" \n",
@1.last_line, $1);
```

```
                }
                sprintf($$, "%s [ %s ]", $1, $3);
      $$ = strdup($$);
        }
        | UNARY_OPERATOR T_IDENTIFIER '[' EXPRESSION ']' {
                if (lookup($2) == NULL) {
                        printf("[Error] at line:%d - Undeclared variable \"%s\" \n",
@2.last_line, $2);
                }
                sprintf($$, "%s %s [ %s ]", $1, $2, $4);
      $$ = strdup($$);
        }
        | T_IDENTIFIER '[' EXPRESSION  ']' UNARY_OPERATOR {
                if (lookup($1) == NULL) {
                        printf("[Error] at line:%d - Undeclared variable \"%s\" \n",
@1.last_line, $1);
                }
                sprintf($$, "%s [ %s ] %s", $1, $3, $5);
      $$ = strdup($$);
        }
        | T_CHAR_LITERAL {
      $$ = strdup($1);
   }
        | T_NUMBER_LITERAL {
      $$ = strdup($1);
   }
        | T_STRING_LITERAL {
      $$ = strdup($1);
   }
        | T_BOOL_LITERAL {
      $$ = strdup($1);
   }
        ;

UNARY_OPERATOR
        : T_OP_INCREMENT {
      $$ = strdup($1);
   }
        | T_OP_DECREMENT {
      $$ = strdup($1);
   }
```

```
        ;

TYPE
        : T_TYPE_INT {
                $$ = strdup($1);
        }
        | T_TYPE_DOUBLE {
                $$ = strdup($1);
        }
        | T_TYPE_FLOAT {
                $$ = strdup($1);
        }
        | T_TYPE_CHAR {
                $$ = strdup($1);
        }
        | T_TYPE_STRING {
                $$ = strdup($1);
        }
        | T_TYPE_VOID {
                $$ = strdup($1);
        }
        | T_TYPE_BOOL {
                $$ = strdup($1);
        }
        ;
```

5. DESIGN STRATEGY

i) SYMBOL TABLE CREATION

The symbol table is implemented as a hash table for constant time (O(1)) access. Chaining is done to avoid collision. A unique hash function based on the name of the identifier is calculated and the location in the symbol table is determined based on that.

ii) INTERMEDIATE CODE GENERATION

A stack based approach is used to generate Three Address Code which is then converted to quadruple format as specified.

iii) CODE OPTIMIZATION

The quadruple data structure created by the generation of intermediate code is taken as input for the optimization phase. This phase performs the different optimizations and gives the resulting quadruple data structure as the output.

The following code optimizations have been performed:

i. Constant folding

- Expressions that can be evaluated at compile time as the arguments forming the expressions are constants, are evaluated and the resulting value is assigned to the appropriate variable.
- Algebraic identities are also constant folded by this optimization. Algebraic identities are equations that are always true regardless of the value assigned to the variables
- Example for algebraic identity constant folding:
  $a + 0 = 0 + a = a$

ii. Constant propagation

- If the value of the variable is a constant that is known at compile time, this value is propagated and substituted whenever this variable is encountered.
- This is usually followed by constant folding

iii. Common Subexpression Elimination

- An occurrence of an expression E is called a common subexpression, if E is previously computed and the values in E have not changed since the previous computation.
- All such future occurrences of the expression can be eliminated as there is no need to recompute the value of the expression
- The Variables that are assigned to these future occurrences of the expression are assigned to the temporary that holds the value of the original expression E

iv. Strength Reduction

- Here an expensive operation is replaced by a cheaper operation.
- The cost being talked about here is with respect to the evaluation of the expression by the underlying hardware
- Example:
  - a*2 => a<<1
  - a/2 => a>>1

iv) ERROR HANDLING

The following error handling strategies have been employed:

i) In Scanner

- Length of identifier greater than 32 characters (automatically truncated to first 32 characters after generating the error)
- Incomplete character literals (For eg: ‘a)
- Incomplete string literals (For eg: “string)
- Non terminating comments are flagged as erroneous
- Any invalid characters not supported by the programming language are identified to be errors

ii) In Parser

- Redeclaration of variables
- Usage of undeclared variables
- Incorrect usage of break statements (must be used only inside loop bodies)
- Usage of undefined functions

iii) Semantic Analyzer

- Invalid operations on unsupported data types (Semantic error)
- Invalid assignment of incompatible types (Semantic error)

6. IMPLEMENTATION DETAILS

SYMBOL TABLE CREATION

```
typedef struct symbol_table_ {
          int line_number;
          char name[MAX_IDENTIFIER_SIZE];
          char type[MAX_IDENTIFIER_SIZE];
          char category[MAX_IDENTIFIER_SIZE];
          char value[MAX_IDENTIFIER_SIZE];
          int size;
          int scope;
   } symbol_table;

   typedef struct node {
          symbol_table *st;
          struct node *next;
   } node_t;

   node_t* complete_symbol_table[SYMBOL_TABLE_SIZE];
```

This is the structure of the symbol table.

INTERMEDIATE CODE GENERATION

For each type of TAC which has to be generated, a separate function is defined. This function is called as part of the action when the grammar is matched.
For example, expr_code_gen function is called when a binary expression is matched. When an expression is matched, the stack now contains the top two elements as the left hand side and right hand side of the expression and thus the Three Address Code is generated. The corresponding quadruple is also generated.
Example:

```
void expr_code_gen(char *op)
{
   // create temp
   if(TAC)
   {
          char temp_var[20];
          sprintf(temp_var,"t%d",temp_id);
          temp_id+=1;
          printf("%s = %s %s %s\n",temp_var,stack[top-1],op,stack[top]);
          char value_temp[40000];
```

```
            sprintf(value_temp,"%s %s %s",stack[top-1],op,stack[top]);
            insert(temp_var,"temporary","", 0, value_temp);
            //quadruple add
            create_quad(stack[top-1],stack[top],temp_var,op);

            top-=1;
            strcpy(stack[top],temp_var);
    }

}
```

CODE OPTIMIZATION

- Input to code optimizer program is the Intermediate code generated in Quadruple format
- A python program inputs this Intermediate code and stores it in a List data structure
- Logic is implemented using general if and for loops
- Output generated is also in Quadruple format displayed accordingly

ERROR HANDLING

Provide instructions on how to build and run your program.

- yyleng used to identify the length of identifiers, to flag very long identifiers
- Invalid characters are flagged using Regular expression rules
- Incomplete character literals are identified by use of Regex
- Incomplete string literals are identified by use of Regex
- Nonterminating comments are identified by use of Regex
- All parser errors are identified by looking up the Symbol table which stores all information about variables
- Semantic errors are identified by usage of flag variables to correspondingly identify matching types and operations

7. Results and possible shortcomings of your Mini-Compiler

- All functionalities defined in the Architecture of our mini-compiler have been implemented and works as expected
- Basic constructs of if, if-else and for with different variations have been implemented and works as expected
- Intermediate code generation works for all the syntax handled by the grammar defined in the yacc file
- Optimization processes are detailed in handling cases of Constant folding, Constant propagation, Common subexpression elimination and Strength Reduction
- This only being a mini-compiler, it cannot be used as a complete alternative to the standard C++ compiler. This compiler only handles restricted syntax and constructs due to time constraints
- Compiler is designed only to handle static operations and no run time operations

8. SNAPSHOTS (of different outputs)

Example 1:

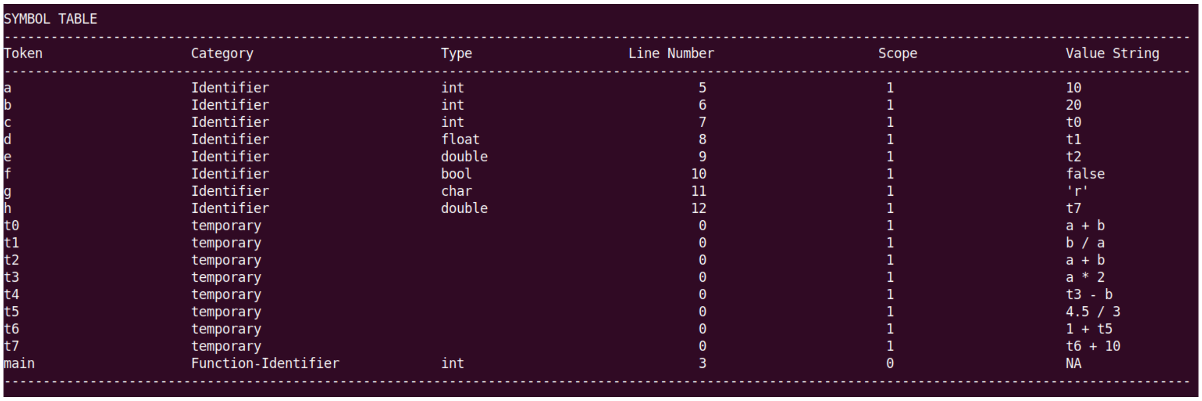
Input:

```
#include <stdio.h>

int main()
{
   int a = 10;
   int b = 20;
   int c = a + b;
   float d = b / a;
   double e = a + b;
   bool f = a * 2 - b;
   char g = 'r';
   double h = 1 + 4.5 / 3 + 10;
```

}

Output:

Symbol Table:



```
SYMBOL TABLE
--------------------------------------------------------------------------------------------------------------------------------------------------------
Token                   Category                        Type                    Line Number                     Scope                   Value String
--------------------------------------------------------------------------------------------------------------------------------------------------------
a                       Identifier                      int                               5                         1                       10
b                       Identifier                      int                               6                         1                       20
c                       Identifier                      int                               7                         1                       t0
d                       Identifier                      float                             8                         1                       t1
e                       Identifier                      double                            9                         1                       t2
f                       Identifier                      bool                             10                         1                       false
g                       Identifier                      char                             11                         1                       'r'
h                       Identifier                      double                           12                         1                       t7
t0                      temporary                                                         0                         1                       a + b
t1                      temporary                                                         0                         1                       b / a
t2                      temporary                                                         0                         1                       a + b
t3                      temporary                                                         0                         1                       a * 2
t4                      temporary                                                         0                         1                       t3 - b
t5                      temporary                                                         0                         1                       4.5 / 3
t6                      temporary                                                         0                         1                       1 + t5
t7                      temporary                                                         0                         1                       t6 + 10
main                    Function-Identifier             int                               3                         0                       NA
--------------------------------------------------------------------------------------------------------------------------------------------------------
```

Intermediate Code (in TAC):

```
a = 10
b = 20
t0 = a + b
c = t0
t1 = b / a
d = t1
t2 = a + b
e = t2
t3 = a * 2
t4 = t3 - b
f = t4
g = 'r'
t5 = 4.5 / 3
t6 = 1 + t5
t7 = t6 + 10
h = t7
```

Intermediate Code (in Quadruple format):

```
QUADRUPLES
Op      arg1    arg2    res
=       10              a
=       20              b
```

+	a	b	t0
=	t0		c
/	b	a	t1
=	t1		d
+	a	b	t2
=	t2		e
*	a	2	t3
-	t3	b	t4
=	t4		f
=	'r'		g
/	4.5	3	t5
+	1	t5	t6
+	t6	10	t7
=	t7		h

Optimised Intermediate Code:

With Common Subexpression Elimination and Strength Reduction:

op	arg1	arg2	result
=	10		a
=	20		b
+	a	b	t0
=	t0		c
/	b	a	t1
=	t1		d
<<	a	1	t3
-	t3	b	t4
=	t4		f
=	'r'		g
/	4.5	3	t5
+	1	t5	t6
+	t6	10	t7
=	t7		h

With Constant folding and Constant propagation:

op	arg1	arg2	result

```
=       10          a
=       20          b
=       30          t0
=       30          c
=       2.0         t1
=       2.0         d
=       20          t3
=       0           t4
=       0           f
=       'r'         g
=       1.5         t5
=       2.5         t6
=       12.5        t7
=       12.5        h
```

Example 2:

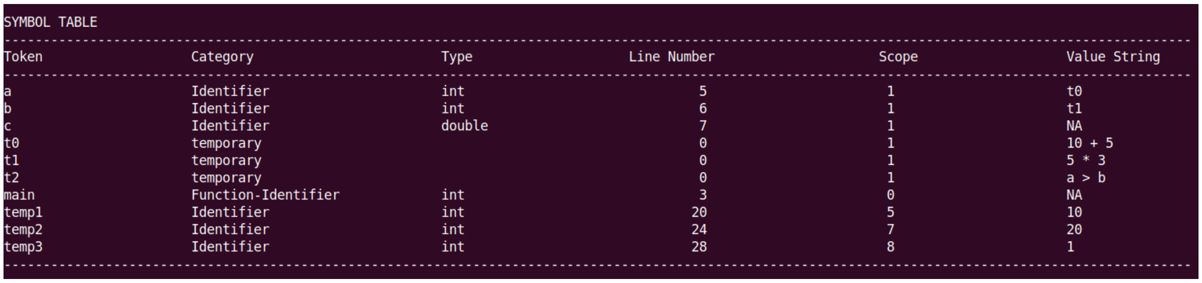
Input:

```
#include <stdio.h>

int main()
{
   int a = 10 + 5;
   int b = 5 * 3;
   double c;
   if(a > b)
   {
      c = 1.0;
   }
   else
   {
      c = 0.0;
   }
   if(a)
   {
      if(b)
      {
         int temp1 = 10;
      }
```

```
        else if(c)
        {
           int temp2 = 20;
        }
        else
        {
           int temp3 = 1;
        }
    }
}
```

Output:

Symbol Table:



```
SYMBOL TABLE
--------------------------------------------------------------------------------------------------------------------------------------------------------
Token                   Category                        Type                    Line Number                     Scope                   Value String
--------------------------------------------------------------------------------------------------------------------------------------------------------
a                       Identifier                      int                              5                       1                      t0
b                       Identifier                      int                              6                       1                      t1
c                       Identifier                      double                           7                       1                      NA
t0                      temporary                                                        0                       1                      10 + 5
t1                      temporary                                                        0                       1                      5 * 3
t2                      temporary                                                        0                       1                      a > b
main                    Function-Identifier             int                              3                       0                      NA
temp1                   Identifier                      int                             20                       5                      10
temp2                   Identifier                      int                             24                       7                      20
temp3                   Identifier                      int                             28                       8                      1
--------------------------------------------------------------------------------------------------------------------------------------------------------
```

Intermediate Code (in TAC):

```
t0 = 10 + 5
a = t0
t1 = 5 * 3
b = t1
t2 = a > b
if t2 goto L0
goto L1
L0:
c = 1
goto L2
L1:
c = 0
L2:
if a goto L3
goto L4
```

```
L3:
if b goto L5
goto L6
L5:
temp1 = 10
goto L7
L6:
if c goto L8
goto L9
L8:
temp2 = 20
goto L10
L9:
temp3 = 1
L10:
L7:
goto L11
L4:
L11:
```

Intermediate Code (in Quadruple format):

```
QUADRUPLES
Op      arg1    arg2    res
+       10      5       t0
=       t0              a
*       5       3       t1
=       t1              b
>       a       b       t2
L0      t2              if
goto                    L1
L0                      label
=       1               c
goto                    L2
L1                      label
=       0               c
L2                      label
L3      a               if
goto                    L4
L3                      label
L5      b               if
```

```
goto                    L6
L5                      label
=       10              temp1
goto                    L7
L6                      label
L8      c               if
goto                    L9
L8                      label
=       20              temp2
goto                    L10
L9                      label
=       1               temp3
L10                     label
L7                      label
goto                    L11
L4                      label
L11                     label
```

Optimised Intermediate Code:

With Common Subexpression Elimination and Strength Reduction:

```
op      arg1    arg2    result

+       10      5       t0
=       t0              a
*       5       3       t1
=       t1              b
>       a       b       t2
L0      t2              if
goto                    L1
L0                      label
=       1               c
L1                      label
=       0               c
L2                      label
L3      a               if
L3                      label
L5      b               if
L5                      label
=       10              temp1
```

L6			label
L8	c		if
L8			label
=	20		temp2
L9			label
L10			label
L7			label
L4			label
L11			label

With Constant folding and Constant propagation:

op	arg1	arg2	result
=	15		t0
=	15		a
=	15		t1
=	15		b
=	False		t2
L0	t2		if
goto			L1
L0			label
=	1		c
L1			label
=	0		c
L2			label
L3	15		if
L3			label
L5	15		if
L5			label
=	10		temp1
L6			label
L8	0		if
L8			label
=	20		temp2
L9			label
L10			label
L7			label
L4			label
L11			label

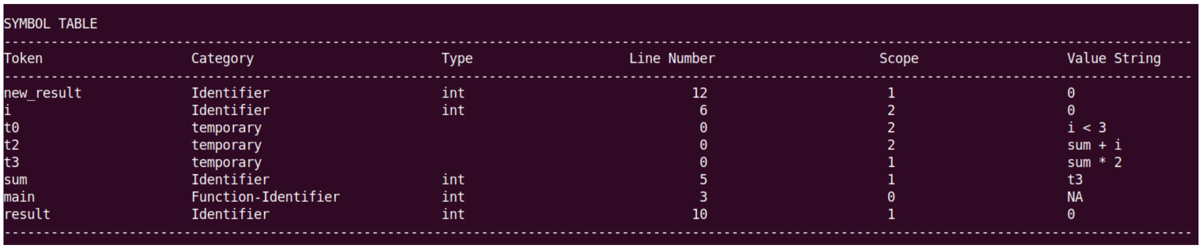
Example 3:

Input:

```
#include <stdio.h>

int main()
{
   int sum = 0;
   for(int i = 0;i < 3;i++)
   {
      sum += i;
   }
   int result = sum;
   sum *= 2;
   int new_result = sum;
}
```

Output:

Symbol Table:



```
SYMBOL TABLE
--------------------------------------------------------------------------------------------------------------------------------------------------------
Token                   Category                        Type                    Line Number                     Scope                   Value String
--------------------------------------------------------------------------------------------------------------------------------------------------------
new_result              Identifier                      int                              12                     1                       0
i                       Identifier                      int                               6                     2                       0
t0                      temporary                                                         0                     2                       i < 3
t2                      temporary                                                         0                     2                       sum + i
t3                      temporary                                                         0                     1                       sum * 2
sum                     Identifier                      int                               5                     1                       t3
main                    Function-Identifier             int                               3                     0                       NA
result                  Identifier                      int                              10                     1                       0
--------------------------------------------------------------------------------------------------------------------------------------------------------
```

Intermediate Code (in TAC):

```
sum = 0
i = 0
L0:
t0 = i < 3
if t0 goto L1
goto L2
```

```
L3:
t1 = i + 1
i = t1
goto L0
L1:
t2 = sum + i
sum = t2
goto L3
L2:
result = sum
t3 = sum * 2
sum = t3
new_result = sum
```

Intermediate Code (in Quadruple format):

```
QUADRUPLES
Op      arg1    arg2    res
=       0               sum
=       0               i
L0                      label
<       i       3       t0
if      t0              L1
goto                    L2
L3                      label
+       i       1       t1
=       t1              i
goto                    L0
L0                      label
+       sum     i       t2
=       sum             t2
goto                    L3
L2                      label
=       sum             result
*       sum     2       t3
=       sum             t3
=       sum             new_result
```

Optimised Intermediate Code:

With Common Subexpression Elimination and Strength Reduction:

op	arg1	arg2	result
=	0		sum
<	sum	3	t0
if	t0		L1
goto			L2
L3			label
+	sum	1	t1
=	t1		sum
L0			label
+	sum	sum	t2
=	sum		t2
L2			label
<<	sum	1	t3
=	sum		new_result

With Constant folding and Constant propagation:

op	arg1	arg2	result
=	0		sum
=	True		t0
if	t0		L1
goto			L2
L3			label
=	1		t1
=	1		sum
L0			label
=	2		t2
=	1		t2
L2			label
=	2		t3
=	1		new_result

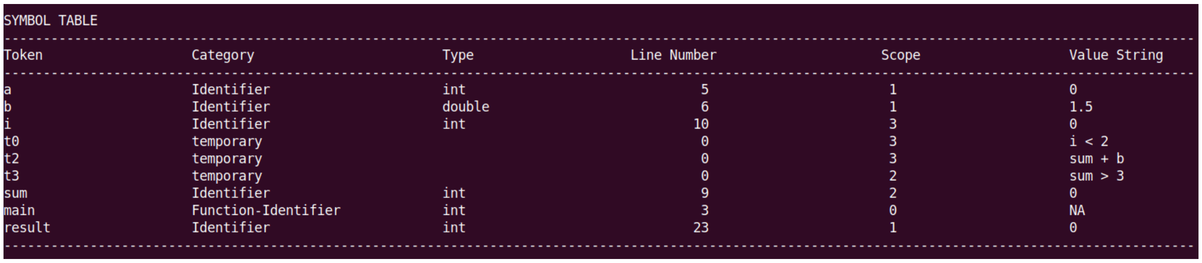
Example 4:

Input:

```
#include <stdio.h>

int main()
{
    int a = 0;
    double b = 1.5;
    int sum;
    if(b)
    {
        sum = 0;
        for(int i = 0;i<2;i++)
        {
            sum = sum + b;
        }
        if(sum > 3)
        {
            sum  = 10;
        }
        else
        {
            sum = 0;
        }
    }
    int result = sum;
}
```

Output:

Symbol Table:



```
SYMBOL TABLE
---------------------------------------------------------------------------------------------------------------------------------
Token                   Category                        Type                    Line Number             Scope                   Value String
---------------------------------------------------------------------------------------------------------------------------------
a                       Identifier                      int                               5                 1                   0
b                       Identifier                      double                            6                 1                   1.5
i                       Identifier                      int                              10                 3                   0
t0                      temporary                                                         0                 3                   i < 2
t2                      temporary                                                         0                 3                   sum + b
t3                      temporary                                                         0                 2                   sum > 3
sum                     Identifier                      int                               9                 2                   0
main                    Function-Identifier             int                               3                 0                   NA
result                  Identifier                      int                              23                 1                   0
---------------------------------------------------------------------------------------------------------------------------------
```

Intermediate Code (in TAC):

```
a = 0
b = 1.5
if b goto L0
goto L1
L0:
sum = 0
i = 0
L2:
t0 = i < 2
if t0 goto L3
goto L4
L5:
t1 = i + 1
i = t1
goto L2
L3:
t2 = sum + b
sum = t2
goto L5
L4:
t3 = sum > 3
if t3 goto L6
goto L7
L6:
sum = 10
goto L8
L7:
sum = 0
L8:
goto L9
L1:
L9:
result = sum
```

Intermediate Code (in Quadruple format):

```
QUADRUPLES
Op      arg1    arg2    res
=       0               a
=       1.5             b
```

L0	b		if
goto			L1
L0			label
=	0		sum
=	0		i
L2			label
<	i	2	t0
if	t0		L3
goto			L4
L5			label
+	i	1	t1
=	t1		i
goto			L2
L2			label
+	sum	b	t2
=	t2		sum
goto			L5
L4			label
>	sum	3	t3
L6	t3		if
goto			L7
L6			label
=	10		sum
goto			L8
L7			label
=	0		sum
L8			label
goto			L9
L1			label
L9			label
=	sum		result

Optimised Intermediate Code:

With Common Subexpression Elimination and Strength Reduction:

op	arg1	arg2	result
=	1.5		b
L0	b		if

goto			L1
L0			label
=	0		i
<	i	2	t0
if	t0		L3
L5			label
+	i	1	t1
=	t1		i
L2			label
+	a	b	t2
=	t2		a
L4			label
>	a	3	t3
L6	t3		if
L6			label
=	10		a
L7			label
L8			label
L1			label
L9			label
=	i		result

With Constant folding and Constant propagation:

op	arg1	arg2	result
=	1.5		b
L0	1.5		if
goto			L1
L0			label
=	0		i
=	True		t0
if	t0		L3
L5			label
=	1		t1
=	1		i
L2			label
+	a	1.5	t2
=	t2		a
L4			label

```
>       a       3       t3
L6      t3              if
L6                      label
=       10              a
L7                      label
L8                      label
L1                      label
L9                      label
=       1               result
```

9. CONCLUSIONS

This implementation of a C++ mini-compiler enables compilation of basic C++ programs covering basic constructs of if, if-else and for. Corresponding Intermediate code is generated by the front end of the compiler, in the form of Three Address Code, represented in Quadruple format. Multiple machine independent optimizations have been implemented which improves efficiency and execution times of the compiled programs.

This project has enabled us to experience a true hands-on approach to learning about Compiler Design. We have learnt a lot about different phases involved in the compilation process. We have learnt to write the Lexer to generate tokens, write the required context-free grammar for handling the language syntax and generate a parse tree, write modular code to generate the intermediate code and develop functions to convert the generated intermediate code into a more optimized version.

10. FURTHER ENHANCEMENTS

This being only a mini-compiler, it can be extended to cover more standard C++ functionalities like classes, STL functions, etc. This would require building upon our current context-free grammar.

We can also bring about further enhancements through optimizations. Loop level optimizations can be implemented to improve efficiency and execution times of our programs.

REFERENCES/BIBLIOGRAPHY

- Lex and Yacc Anchor material provided by PES University Compiler Design Faculty
- Regular Expressions online website (https://www.regular-expressions.info/)
- ISO C++ reference (https://isocpp.org/wiki/faq/compiler-dependencies#yaccable-grammar)
- C++ yacc-able grammar reference (http://www.computing.surrey.ac.uk/research/dsrg/fog/CxxGrammar.y)
- Lecture slides on Intermediate Code generation and Optimisation techniques provided by PES University Compiler Design Faculty