
*Report on*

# "Mini Compiler for If-Else and While Constructs in Python"

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

## *Compiler Design Laboratory*

**Bachelor of Technology**
**in**
**Computer Science & Engineering**

***Submitted by:***

| | |
|---|---|
| **Vishal S Rao** | **01FB16ECS450** |
| **Vishnu V Singh** | **01FB16ECS451** |
| **Vishruth R G** | **01FB16ECS452** |

*Under the guidance of*

**Prof. C O Prakash**
Assistant Professor
PES University, Bengaluru

**January – May 2019**

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

# TABLE OF CONTENTS


| Chapter No. | Title | Page No. |
|---|---|---|
| 1. | INTRODUCTION | 01 |
| 2. | ARCHITECTURE OF LANGUAGE: | 07 |
| 3. | LITERATURE SURVEY | 07 |
| 4. | CONTEXT FREE GRAMMAR | 08 |
| 5. | DESIGN STRATEGY<br>● SYMBOL TABLE CREATION<br>● ABSTRACT SYNTAX TREE<br>● INTERMEDIATE CODE GENERATION<br>● CODE OPTIMIZATION<br>● ERROR HANDLING | 09 |
| 6. | IMPLEMENTATION DETAILS<br>(Tools and Data Structures)<br>● SYMBOL TABLE CREATION<br>● ABSTRACT SYNTAX TREE (internal representation)<br>● INTERMEDIATE CODE GENERATION<br>● CODE OPTIMIZATION<br>● ERROR HANDLING<br>● Instruction for Execution | 11 |
| 7. | RESULTS AND SHORTCOMINGS | 13 |
| 8. | SNAPSHOTS | 14 |
| 9. | CONCLUSIONS | 18 |
| 10. | FURTHER ENHANCEMENTS | 18 |
| | | |

# INTRODUCTION

This Mini Compiler is Built for Python and handles the If-Elif-Else and the While Constructs.

*Sample Input*

```
#Start Program

import helloWorld
x=10
y=x+20
z=x*10
str = "No."

if(z==x):
        print(z)
else:
        print("No.")

#End Program
```

*Sample Output*

```
---------------------------------Token Sequence-------------------------
1 T_NL
2 T_NL
3 T_IMPT T_helloWorld T_NL
4 T_x T_EQL T_10 T_NL
5 T_y T_EQL T_x T_PL T_20 T_NL
6 T_z T_EQL T_x T_ML T_10 T_NL
7 T_str T_EQL T_"No." T_NL
8 T_NL
9 T_If T_OP T_z T_EQ T_x T_CP T_Cln T_NL
10 T_ID T_Print T_OP T_z T_CP T_NL
11 T_Else T_Cln T_NL
12 T_ID T_Print T_OP T_"No." T_CP T_NL
13 T_NL
14 T_EOF
Valid Python Syntax
-----------------------------------------------------------------------------
```

```
------------------------Abstract Syntax Tree-------------------------
NewLine(2)
import(1) NewLine(2)
hWorld  =(2) NewLine(2)
      x  10  =(2) NewLine(2)
         y  +(2) =(2) NewLine(2)
          x  20  z  *(2) =(2) If(3)
             x  10  str  "No."  ==(2) BeginBlock(2) Else(1)
                      z  x  Print(1) EndBlock  BeginBlock(2)
                               z  Print(1) EndBlock
                                      "No."

-------------------------Three Address Code--------------------------
import helloWorld
T2 = 10
x = T2
T5 = x
T6 = 20
T7 = T5 + T6
y = T7
T10 = x
T11 = 10
T12 = T10 * T11
z = T12
T15 = z
T16 = - T15
k = T16
T19 = "No."
str = T19
T22 = z
T23 = x
T24 = T22 == T23
If False T24 goto L0
T25 = z
Print T25
goto L1
L0: T29 = "No."
Print T29
L1:
```

```
---------------------------------All Quads---------------------------------
0       import  helloWorld  -       -
1       =               10      -       T2
2       =               T2      -       x
3       =               x       -       T5
4       =               20      -       T6
5       +               T5      T6      T7
6       =               T7      -       y
7       =               x       -       T10
8       =               10      -       T11
9       *               T10     T11     T12
10      =               T12     -       z
11      =               z       -       T15
12      -               T15     -       T16
13      =               T16     -       k
14      =               "No."   -       T19
15      =               T19     -       str
16      =               z       -       T22
17      =               x       -       T23
18      ==              T22     T23     T24
19      If False        T24     -       L0
20      =               z       -       T25
21      Print           T26     -       -
22      goto            -       -       L1
23      Label           -       -       L0
24      =               "No."   -       T29
25      Print           T30     -       -
26      Label           -       -       L1
---------------------------------------------------------------------------
```

**After Dead Code Elimination**

```
---------------------------------All Quads---------------------------------
0       import          hWorld          -       -
1       =               10      -       T2
2       =               T2      -       x
7       =               x       -       T10
8       =               10      -       T11
9       *               T10     T11     T12
10      =               T12     -       z
16      =               z       -       T22
17      =               x       -       T23
18      ==              T22     T23     T24
19      If False        T24     -       L0
21      Print           T26     -       -
```

```
22      goto            -       -       L1
23      Label           -       -       L0
25      Print           T30     -       -
26      Label           -       -       L1
--------------------------------------------------------------------------------

----------------------------All Symbol Tables----------------------------
Scope Name          Type            Declaration   Last Used Line
(0, 1)  helloWorld  PackageName     3             3
(0, 1)  10          Constant        4             6
(0, 1)  x           Identifier      4             10
(0, 1)  20          Constant        5             5
(0, 1)  y           Identifier      5             5
(0, 1)  z           Identifier      6             11
(0, 1)  k           Identifier      7             7
(0, 1)  "No."       Constant        8             8
(0, 1)  str         Identifier      8             8
(0, 1)  T2          ICGTempVar      -1            -1
(0, 1)  T5          ICGTempVar      -1            -1
(0, 1)  T6          ICGTempVar      -1            -1
(0, 1)  T7          ICGTempVar      -1            -1
(0, 1)  T10         ICGTempVar      -1            -1
(0, 1)  T11         ICGTempVar      -1            -1
(0, 1)  T12         ICGTempVar      -1            -1
(0, 1)  T15         ICGTempVar      -1            -1
(0, 1)  T16         ICGTempVar      -1            -1
(0, 1)  T19         ICGTempVar      -1            -1
(0, 1)  T22         ICGTempVar      -1            -1
(0, 1)  T23         ICGTempVar      -1            -1
(0, 1)  T24         ICGTempVar      -1            -1
(0, 1)  T25         ICGTempVar      -1            -1
(0, 1)  T26         ICGTempVar      -1            -1
(0, 1)  L0          ICGTempLabel    -1            -1
(0, 1)  T29         ICGTempVar      -1            -1
(0, 1)  T30         ICGTempVar      -1            -1
(0, 1)  L1          ICGTempLabel    -1            -1
(1, 4)  "No."       Constant        13            13
--------------------------------------------------------------------------------
```

# ARCHITECTURE OF THE LANGUAGE

Python has a very flexible syntax and we have tried to incorporate as much as possible from our experience of using python into the grammar. But we have not taken care of semicolons, so a semicolon will result in an error while parsing. All lines of code terminate upon seeing a newline character. We have taken care of the following:

- If-Elif-Else and While constructs
- Print Statements
- pass, break and void returns
- Function definitions and Calls
- Lists
- All arithmetic operators and all boolean operators except standalone '!' ("!=" is taken care of)
- Single Line Comments (#)

Semantically we have checked the following :

- Whether any variable used on the RHS is defined and in the current scope or any Enclosing Scope of the current scope.
- Whether a variable being indexed is List
- Whether all expressions in the If and While Clauses are Boolean expressions

# LITERATURE SURVEY


1. Lex and Yacc Doc by Tom Niemann
2. Official Bison Documentation : https://www.gnu.org/software/bison/manual/
3. Stackoverflow : https://stackoverflow.com/

# THE CONTEXT FREE GRAMMAR

## Grammar to Parse While And If Statements in Python

| Grammar | Expression |
|---|---|
| StartDebugger | StartParse T_EndOfFile |
| constant | T_Number \| T_String |
| T_ID | [_a-zA-Z][_a-zA-Z0-9]* |
| term | T_ID \| constant \| list_index |
| list_index | T_ID T_OB constant T_CB |
| StartParse | T_NL StartParse \| finalStatements T_NL StartParse \| finalStatements T_NL |
| basic_smt | pass_stmt \| break_stmt \| import_stmt \| assign_stmt \| bool_exp \| arith_exp \| print_stmt \| return_stmt |
| arith_exp | term \| arith_exp T_PL arith_exp \| |
| | arith_exp T_MN arith_exp \| |
| | arith_exp T_ML arith_exp \| |
| | arith_exp T_DV arith_exp \| |
| | T_OP arith_exp T_CP |
| ROP | T_GT \| T_LT \| T_ELT \| T_EGT |
| bool_exp | bool_term \| bool_term T_Or bool_term \| arith_exp T_LT arith_exp \| bool_term T_And bool_term |
| | arith_exp T_GT arith_exp \| arith_exp T_ELT arith_exp \| arith_exp T_EGT arith_exp \| arith_exp T_In T_ID |
| bool_term | bool_factor \| arith_exp T_EQ arith_exp \| T_True \| T_False |
| bool_factor | T_Not bool_factor \| T_OP bool_exp T_CP |
| import_stmt | T_Import T_ID |
| assign_stmt | T_ID T_EQL arith_exp \| T_ID T_EQL bool_exp \| T_ID T_EQL func_call \| T_ID T_EQL T_OB T_CB \| |
| pass_stmt | T_Pass |
| break_stmt | T_Break |
| return_stmt | T_Return |
| print_stmt | T_Print T_OP constant T_CP |
| finalStatements | basic_stmt \| cmpd_stmt \| func_def \| func_call |
| cmpd_stmt | if_stmt \| while_stmt |
| if_stmt | T_If bool_exp T_Cln start_suite \| T_If bool_exp T_Cln start_suite elif_stmts |
| elif_stmts | T_Elif bool_exp T_Cln start_suite elif_stmts \| else_stmt |
| else_stmt | T_Else T_Cln start_suite |
| while_stmt | T_While bool_exp T_Cln start_suite |
| start_suite | basic_stmt \| T_NL ID finalStatements suite |
| suite | T_NL ND finalStatements suite \| T_NL end_suite |
| end_suite | DD finalStatements \| DD \| |
| args | T_ID args_list \| |
| args_list | T_Comma T_ID args_list \| |
| call_list | T_Comma term call_list \| |
| call_args | term call_list \| |
| func_def | T_Def T_ID T_OP args T_CP T_Cln start_suite |
| func_call | T_ID T_OP call_args T_CP |

| Legend |
|---|
| T_Import : "import" |
| T_Print : "print" |
| T_Pass : "pass" |
| T_If : "if" |
| T_While : "while" |
| T_Break : "break" |
| T_And : "and" |
| T_Or : "or" |
| T_Not : "not" |
| T_Elif : "elif" |
| T_Else : "else |
| T_Def : "def" |
| T_In : "in" |
| T_Cln : ":" |
| T_GT : ">" |
| T_LT : "<" |
| T_EGT : ">=" |
| T_ELT : "<=" |
| T_EQ : "==" |
| T_NEQ : "!=" |
| T_True : "True" |
| T_False : "False" |
| T_PL : "+" |
| T_MN : "-" |
| T_ML : "*" |
| T_DV : "/" |
| T_OP : "(" |
| T_CP : ")" |
| T_OB : "[" |
| T_CB : "]" |
| T_Comma : "," |
| T_EQL : "=" |
| T_NL : "\n" |
| T_String : String |
| T_Number : Number |
| ND : Nodent |
| ID : Indent |
| DD : Dedent |
| T_EOF : End Of File |

# DESIGN STRATEGY

The Design Strategy we have implemented is that firstly, every links back to the Symbol Table. This means that the required nodes of the Abstract Syntax tree and a the required Quadruples in the Intermediate code have a link to the Symbol table. All the compiler generated Temporaries are also stored in the symbol table.

The Symbol Table stores ‘Records’ having 4 columns as you can see in the sample output, ie,

- Scope : Scope of each variable contained in record
- Name : Value/Name of each variable contained in record
- Type : Type of each variable contained in record
  - PackageName
  - Func_Name
  - Identifier
  - Constant
  - ListTypeID
  - ICGTempVar
  - ICGTempLabel
- Declaration : Line of Declaration of each variable contained in record
- Last Use Line

The scope is a function of Indentation depth and to make it unique we have a tuple of the scope of the parent and the current scope calculated using the Indentation depth.

For the Abstract Syntax Tree, We have 2 Types of Nodes, Leaf nodes and Internal nodes. The nodes can have variable number of children (0-3) depending upon the construct it represents. Take the example of the If-Else Statement,

If

Condition CodeBlock Else

To display the AST, We take the AST and store it as a matrix of levels. As we can see in the sample output, we have printed each level of the AST. All Internal nodes also have a number enclosed in brackets next to them, which represents the number of children they have in the next level. Leaf nodes in the AST representing identifiers, constants, Lists, packages point to a record in the symbol table.

The Intermediate code is generated by recursively stepping through the AST. Each line is stored as a quadruple (Operation, Arg1, Arg2, Result) so that it we may easily optimize code in the following steps.

We Eliminate dead code, specifically unused variables in the whole program. For example, if we have the following lines of code:

```
a = 10
b = 10
c = a + b
```

And these 3 variables are not used on any other RHS, then these 3 lines of code are Eliminated during optimization. All the dead variables in the code are removed. We iterate through the Quads to do this.

To take care of the Indentation based code structure and scoping we have implemented a stack and we use 3 tokens. The top of the stack always points to the current indentation value, if that value on scanning the next line, doesn't change, it implies that we're in the same scope and hence, we return the token 'ND', i.e, 'No-dent'. If the value increases, it means we are entering a sub-scope and we return the token 'ID' , i.e. 'Indent' and finally, if the value decreases, it means we're returning to one of the enclosing scope and we return 'DD', i.e. 'Dedent'.

For Error Handling, Whenever the parser encounters an error, It prints the Line no and column number of the error. We also display the following errors:

- Identifier <var> Not declared in scope
- Identifier <var> Not Indexable

All Comments are removed from the code before parsing

# IMPLEMENTATION DETAILS

The Symbol table uses two Structures,

```
typedef struct record
{
        char *type;
        char *name;
        int decLineNo;
        int lastUseLine;
} record;

typedef struct STable
{
        int no;
        int noOfElems;
        int scope;
        record *Elements;
        int Parent;

} STable;
```

The “record” structure represents each record in the symbol table. Each symbol table can contain a maximum of “MAXRECST” records, MAXRECST is a macro.

The “STable” structure represents one symbol table. A new Symbol table is made for every scope. A Maximum of “MAXST” symbol tables and hence scopes can exist, MAXST is a macro.

The Abstract Syntax Tree uses one structure,

```
typedef struct ASTNode
{
        int nodeNo;
        /*Operator*/
        char *NType;
        int noOps;
        struct ASTNode** NextLevel;

         /*Identifier or Const*/
         record *id;
} node;
```

This ASTNode  structure takes care of both leaf nodes as well as Internal

“Operator” Nodes. The respective values are set depending upon the type of node. Each node can have 0-3 children. We print the AST by first storing it in a Matrix of Order “MAXLEVELS” x “MAXCHILDREN” and printing the matrix Levelwise. This Matrix, is a matrix of pointers to the AST. The “noOps” element of the Node gives the number of children of that node.

The Three-Address Code is represented and stored as Quads that are given by the Structutre,

```
typedef struct Quad
{
        char *R;
        char *A1;
        char *A2;
        char *Op;
        int I;
} Quad;
```

The last element, the integer ‘I’, is used during code optimization. All the Three-Address codes are stored as Quads in an array of Quads. There can be a maximum of “MAXQUADS” quadruples.

For the dead code elimination we continously loop through the code till no more code can be eliminated. To check if a quadruple represnts dead code we see if the result parameter/element of that quad appears as any arguments to any other subsequent quads which have not been eliminated, If not, we consider that quad to represent dead code and mark the element ‘I’ with the value “-1”.

The scope checking is handled by recursively stepping through enclosing scopes and finding the most recent definition of the variable. If no definition is found, we print the error.

Lastly, To compile the code and execute the code we have provided a makefile.

If you wish to only s
ee the AST,

```
lex grammar.l
yacc -dv grammarAST.y
gcc lex.yy.c y.tab.c -g -ll -o TestAST.out
./TestAST.out < InputFile.txt
```

If you wish to only see the Intermediate code and Optimization

```
lex grammar.l
```

```
yacc -dv grammarICG.y
gcc lex.yy.c y.tab.c -g -ll -o TestICG.out
./TestICG.out < InputFile.txt
```

The makefile provided comipiles the “grammar.y” file and prints everything.

```
./Test.out < Input.txt
```

# RESULTS AND SHORTCOMINGS

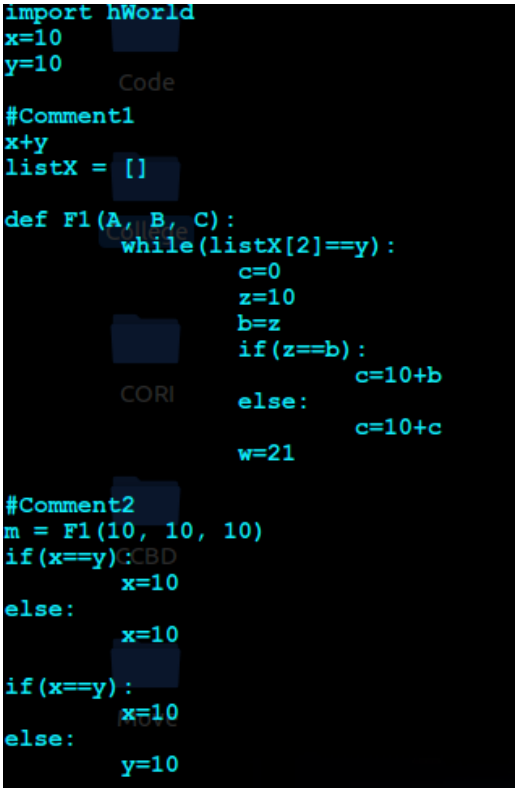
The result achived is that we have a mini compiler which parses grammar corresponding to basic python syntax and generates finally, an optimized intermidiate representation.

The areas where our mini compiler falls short are,

- Doesn’t handle semi colon.
- Only one very basic optimization is implemented that does not reduce code density by a huge margin.
- The Program has a few memory leaks, although mast of them have been taken care of.

# SNAPSHOTS

***Input File with Comments***



```
import hWorld
x=10
y=10

#Comment1
x+y
listX = []

def F1(A, B, C):
        while(listX[2]==y):
                c=0
                z=10
                b=z
                if(z==b):
                        c=10+b
                else:
                        c=10+c
                w=21

#Comment2
m = F1(10, 10, 10)
if(x==y):
        x=10
else:
        x=10

if(x==y):
        x=10
else:
        y=10
```

## *Token Sequence*

```
--------------------------------Token Sequence-------------------------
1 T_IMPT T_hWorld T_NL
2 T_x T_EQL T_10 T_NL
3 T_y T_EQL T_10 T_NL
4 T_NL
5 T_NL
6 T_x T_PL T_y T_NL
7 T_listX T_EQL T_OB T_CB T_NL
8 T_NL
9 T_Def T_F1 T_OP T_A T_Comma T_B T_Comma T_C T_CP T_Cln T_NL
10 T_ID T_While T_OP T_listX T_OB T_2 T_CB T_EQ T_y T_CP T_Cln T_NL
11 T_ID T_c T_EQL T_0 T_NL
12 T_ND T_z T_EQL T_10 T_NL
13 T_ND T_b T_EQL T_z T_NL
14 T_ND T_If T_OP T_z T_EQ T_b T_CP T_Cln T_NL
15 T_ID T_c T_EQL T_10 T_PL T_b T_NL
16 T_DD T_Else T_Cln T_NL
17 T_ID T_c T_EQL T_10 T_PL T_c T_NL
18 T_DD T_w T_EQL T_21 T_NL
19 T_DD T_NL
20 T_NL
21 T_m T_EQL T_F1 T_OP T_10 T_Comma T_10 T_Comma T_10 T_CP T_NL
22 T_If T_OP T_x T_EQ T_y T_CP T_Cln T_NL
23 T_ID T_x T_EQL T_10 T_NL
24 T_Else T_Cln T_NL
25 T_ID T_x T_EQL T_10 T_NL
26 T_NL
27 T_If T_OP T_x T_EQ T_y T_CP T_Cln T_NL
28 T_ID T_x T_EQL T_10 T_NL
29 T_Else T_Cln T_NL
30 T_ID T_y T_EQL T_10 T_NL
31 T_NL
32 T_EOF
Valid Python Syntax
```

## *Abstract Syntax Tree*

```
-----------------------Abstract Syntax Tree-------------------------
NewLine(2)
import(1) NewLine(2)
hWorld  =(2) NewLine(2)
     x  10  =(2) NewLine(2)
        y  10  +(2) NewLine(2)
           x  y  listX  NewLine(2)
                 Func_Name(3) NewLine(2)
                 F1  A, B, C  BeginBlock(2) =(2) NewLine(2)
                        While(2) EndBlock  m  Func_Call(2) If(3) If(3)
                               ==(2) BeginBlock(2) F1  10, 10, 10  ==(2) BeginBlock(2) Else(1) ==(2) BeginBlock(2) Else(1)
                                        ListIndex(2) y  =(2) Next(2) x  y  =(2) EndBlock  BeginBlock(2) x  y  =(2) EndBlock  BeginBlock(2)
                                                 listX  2  c  0  =(2) Next(2) x  10  =(2) EndBlock  x  10  =(2) EndBlock
                                                                      z  10  =(2) Next(2) x  10  y  10
                                                                           b  z  If(3) EndBlock
                                                                                 ==(2) BeginBlock(2) Else(1)
                                                                                 z  b  =(2) EndBlock  BeginBlock(2)
                                                                                       c  +(2) =(2) EndBlock(1)
                                                                                        10  b  c  +(2) =(2)
                                                                                            10  c  w  21
```

## *Intermediate Code*

```
import hWorld
T2 = 10
x = T2
T5 = 10
y = T5
T8 = x
T9 = y
T10 = T8 + T9
Begin Function F1
T15 = listX[2]
T16 = y
T17 = T15 == T16
L0: If False T17 goto L1
T18 = 0
c = T18
T21 = 10
z = T21
T24 = z
b = T24
T27 = z
T28 = b
T29 = T27 == T28
If False T29 goto L2
T30 = 10
T31 = b
T32 = T30 + T31
c = T32
goto L3
L2: T37 = 10
T38 = c
T39 = T37 + T38
c = T39
T42 = 21
w = T42
L3: goto L0
L1: End Function F1
Push Param 10
Push Param  10
Push Param  10
(T63)Call Function F1, 3
Pop Params for Function F1, 3
m = T63
T66 = x
T67 = y
T68 = T66 == T67
If False T68 goto L6
T69 = 10
x = T69
goto L7
L6: T74 = 10
x = T74
L7: T81 = x
T82 = y
T83 = T81 == T82
If False T83 goto L8
T84 = 10
x = T84
goto L9
L8: T89 = 10
y = T89
```

```
--------------------------------All Quads--------------------------------
0       import  hWorld  -       -
1       =       10      -       T2
2       =       T2      -       x
3       =       10      -       T5
4       =       T5      -       y
5       =       x       -       T8
6       =       y       -       T9
7       +       T8      T9      T10
8       BeginF  F1      -       -
9       =[]     listX   2       T15
10      =       y       -       T16
11      ==      T15     T16     T17
12      Label   -       -       L0
13      If False        T17     -       L1
14      =       0       -       T18
15      =       T18     -       c
16      =       10      -       T21
17      =       T21     -       z
18      =       z       -       T24
19      =       T24     -       b
20      =       z       -       T27
21      =       b       -       T28
22      ==      T27     T28     T29
23      If False        T29     -       L2
24      =       10      -       T30
25      =       b       -       T31
26      +       T30     T31     T32
27      =       T32     -       c
28      goto    -       -       L3
29      Label   -       -       L2
30      =       10      -       T37
31      =       c       -       T38
32      +       T37     T38     T39
33      =       T39     -       c
34      =       21      -       T42
35      =       T42     -       w
36      Label   -       -       L3
37      goto    -       -       L0
38      Label   -       -       L1
39      EndF    F1      -       -
40      Param   10      -       -
41      Param    10     -       -
42      Param    10     -       -
43      Call    F1      3       T63
44      =       T63     -       m
45      =       x       -       T66
46      =       y       -       T67
47      ==      T66     T67     T68
48      If False        T68     -       L6
49      =       10      -       T69
50      =       T69     -       x
51      goto    -       -       L7
52      Label   -       -       L6
53      =       10      -       T74
54      =       T74     -       x
55      Label   -       -       L7
56      =       x       -       T81
57      =       y       -       T82
58      ==      T81     T82     T83
59      If False        T83     -       L8
60      =       10      -       T84
61      =       T84     -       x
62      goto    -       -       L9
63      Label   -       -       L8
64      =       10      -       T89
65      =       T89     -       y
66      Label   -       -       L9
-------------------------------------------------------------------------
```

***Optimized Code (Quads Removed)***

```
-------------------------------All Quads---------------------------------
0       import  hWorld  -       -
1       =       10      -       T2
2       =       T2      -       x
3       =       10      -       T5
4       =       T5      -       y
8       BeginF  F1      -       -
9       =[]     listX   2       T15
10      =       y       -       T16
11      ==      T15     T16     T17
12      Label   -       -       L0
13      If False        T17     -       L1
14      =       0       -       T18
15      =       T18     -       c
16      =       10      -       T21
17      =       T21     -       z
18      =       z       -       T24
19      =       T24     -       b
20      =       z       -       T27
21      =       b       -       T28
22      ==      T27     T28     T29
23      If False        T29     -       L2
24      =       10      -       T30
25      =       b       -       T31
26      +       T30     T31     T32
27      =       T32     -       c
28      goto    -       -       L3
29      Label   -       -       L2
30      =       10      -       T37
31      =       c       -       T38
32      +       T37     T38     T39
33      =       T39     -       c
36      Label   -       -       L3
37      goto    -       -       L0
38      Label   -       -       L1
39      EndF    F1      -       -
40      Param   10      -       -
41      Param    10     -       -
42      Param    10     -       -
43      Call    F1      3       T63
45      =       x       -       T66
46      =       y       -       T67
47      ==      T66     T67     T68
48      If False        T68     -       L6
49      =       10      -       T69
50      =       T69     -       x
51      goto    -       -       L7
52      Label   -       -       L6
53      =       10      -       T74
54      =       T74     -       x
55      Label   -       -       L7
56      =       x       -       T81
57      =       y       -       T82
58      ==      T81     T82     T83
59      If False        T83     -       L8
60      =       10      -       T84
61      =       T84     -       x
62      goto    -       -       L9
63      Label   -       -       L8
64      =       10      -       T89
65      =       T89     -       y
66      Label   -       -       L9
-------------------------------------------------------------------------
```

## *All Symbol Tables*

```
----------------------------All Symbol Tables----------------------------
Scope    Name     Type              Declaration     Last Used Line
(0, 1)   hWorld   PackageName       1               1
(0, 1)   10       Constant          2               3
(0, 1)   x        Identifier        2               27
(0, 1)   y        Identifier        3               27
(0, 1)   listX    ListTypeID        7               10
(0, 1)   F1       Func_Name         9               9
(0, 1)   T2       ICGTempVar        -1              -1
(0, 1)   T5       ICGTempVar        -1              -1
(0, 1)   T8       ICGTempVar        -1              -1
(0, 1)   T9       ICGTempVar        -1              -1
(0, 1)   T10      ICGTempVar        -1              -1
(0, 1)   T15      ICGTempVar        -1              -1
(0, 1)   T16      ICGTempVar        -1              -1
(0, 1)   T17      ICGTempVar        -1              -1
(0, 1)   L0       ICGTempLabel      -1              -1
(0, 1)   L1       ICGTempLabel      -1              -1
(0, 1)   T18      ICGTempVar        -1              -1
(0, 1)   T21      ICGTempVar        -1              -1
(0, 1)   T24      ICGTempVar        -1              -1
(0, 1)   T27      ICGTempVar        -1              -1
(0, 1)   T28      ICGTempVar        -1              -1
(0, 1)   T29      ICGTempVar        -1              -1
(0, 1)   L2       ICGTempLabel      -1              -1
(0, 1)   T30      ICGTempVar        -1              -1
(0, 1)   T31      ICGTempVar        -1              -1
(0, 1)   T32      ICGTempVar        -1              -1
(0, 1)   L3       ICGTempLabel      -1              -1
(0, 1)   T37      ICGTempVar        -1              -1
(0, 1)   T38      ICGTempVar        -1              -1
(0, 1)   T39      ICGTempVar        -1              -1
(0, 1)   T42      ICGTempVar        -1              -1
(0, 1)   T63      ICGTempVar        -1              -1
(0, 1)   T66      ICGTempVar        -1              -1
(0, 1)   T67      ICGTempVar        -1              -1
(0, 1)   T68      ICGTempVar        -1              -1
(0, 1)   L6       ICGTempLabel      -1              -1
(0, 1)   T69      ICGTempVar        -1              -1
(0, 1)   L7       ICGTempLabel      -1              -1
(0, 1)   T74      ICGTempVar        -1              -1
(0, 1)   T81      ICGTempVar        -1              -1
(0, 1)   T82      ICGTempVar        -1              -1
(0, 1)   T83      ICGTempVar        -1              -1
(0, 1)   L8       ICGTempLabel      -1              -1
(0, 1)   T84      ICGTempVar        -1              -1
(0, 1)   L9       ICGTempLabel      -1              -1
(0, 1)   T89      ICGTempVar        -1              -1
(0, 2)   2        Constant          10              10
(0, 2)   10       Constant          21              21
(0, 2)   m        Identifier        21              21
(1, 3)   0        Constant          11              11
(1, 3)   c        Identifier        11              11
(1, 3)   10       Constant          12              12
(1, 3)   z        Identifier        12              14
(1, 3)   b        Identifier        13              15
(1, 3)   21       Constant          18              18
(1, 3)   w        Identifier        18              18
(2, 4)   10       Constant          15              15
(2, 4)   c        Identifier        15              17
(2, 16)  10       Constant          17              17
(2, 16)  c        Identifier        17              17
(1, 4)   10       Constant          23              23
(1, 4)   x        Identifier        23              23
(1, 8)   10       Constant          25              25
(1, 8)   x        Identifier        25              25
(1, 16)  10       Constant          28              28
(1, 16)  x        Identifier        28              28
(1, 32)  10       Constant          30              30
(1, 32)  y        Identifier        30              30
```

# CONCLUSION

In Conclusion, A Compiler for Python was implemented. In addition to the constructs specified, the basic python constructs were implemented and function definitions and calls were supported.

The compiler also reports the basic errors and gives the line number and column number.

The Intermediate code was represented by quads which was later optimized to remove dead code.

# FURTHER ENHANCEMENTS

The Compiler can be further enhanced by adding,

- Support for ‘For’ Loops
- Better Memory Management
- More efficient optimization techniques
- Semantic analysis for Parameter Matching
- Error Recovery