Report on

C++ Mini-Compiler

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

Compiler Design Laboratory

Bachelor of Technology
in
Computer Science & Engineering

Submitted by:

Bhargavi G	**PES1201800211**
Nisha Suresh	**PES1201801777**
H M Thrupthi	**PES1201801987**

Under the guidance of

Prof Preet Kanwal
Assistant Professor
PES University

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

Introduction

To implement a Mini Compiler for C++ using Lex, Yacc and C++. The input is any valid C++ program containing for loops and switch case constructs. The output is a symbol table for phase 1. For phase 2, output contains three address code in quadruple format, a symbol table containing variables and temporaries and optimized code.

Phase 1 :

- Lex and yacc file
- Grammar encoded in yacc file w.r.t your chosen language
- Syntax Validation
- Symbol table generation - make entry in the symbol table for each identifier,its datatype, value, scope and line number.

Phase 2 :

- Evaluation of Expressions and update in the Symbol table.
- Intermediate Code Generation - Three Address Code (Quadruple format)
- Eliminate Dead code/ unreachable code.
- Implement Common subexpression elimination.
- Implement Constant folding and Constant propagation.
- Move loop invariant code outside the loop.

Input :

```
#include<iostream>
using namespace std;
int main()
{
        int a=3;
        int b=4;
        int i;
        int g = 8;
        int n = 5;
        int k = 2;
        k = a+1;
        int x = a+1;
        for(i=n;i>0;i=i-1)
        {
                k = 2*i;
                g = k+1;
        }
        int c = a;
        return c;
}
```

Output :

```
---------------Three address code-
a = 3
b = 4
g = 8
n = 5
k = 2
t1 = a + 1
k = t1
t2 = a + 1
x = t2
i = n
L1:
t3 = i > 0
if  t3  goto L2
goto L3
L4:
t4 = i - 1
i = t4
goto L1
L2:
t5 = 2 * i
k = t5
t6 = k + 1
g = t6
goto L4
L3:
c = a

Parsing complete
```

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

Operator        Arg1            Arg2            Result
=               3               NULL            a
=               4               NULL            b
=               8               NULL            g
=               5               NULL            n
=               2               NULL            k
+               a               1               t1
=               t1              NULL            k
+               a               1               t2
=               t2              NULL            x
=               n               NULL            i
label           NULL            NULL            L1
>               i               0               t3
if              n               NULL            L2
goto            NULL            NULL            L3
label           NULL            NULL            L4
-               i               1               t4
=               t4              NULL            i
goto            NULL            NULL            L1
label           NULL            NULL            L2
*               2               i               t5
=               t5              NULL            k
+               k               1               t6
=               t6              NULL            g
goto            NULL            NULL            L4
label           NULL            NULL            L3
=               a               NULL            c
```

```
----------------------------Symbol table----------------------------

Slno.               Name              Value           Datatype            lineno              scope

   0                   c                  a               temp                18                  1
   1              return          undefined            keyword                19                  1
   2                   b                  4                int                 6                  1
   3                   i          undefined                int                 7                  1
   4                   g                  8                int                 8                  1
   5                   n                  5                int                 9                  1
   6                   k                  4                int                11                  1
   7                  t1                a+1               temp                11                  1
   9                   k                 t1               temp                11                  1
  10                  t2                a+1               temp                12                  1
  11                   x                  4                int                12                  1
  12                   i                  1                int                13                  2
  14                  t3                i>0               temp                13                  2
  15                  t5                2*i               temp                15                  2
  16                   k                  2                int                15                  2
  17                  t6                k+1               temp                16                  2
  18                   g                  5                int                16                  2
  19                   c                  3                int                18                  1
```

Code Optimization:

```
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$ python3 co.py icg_for.txt
Eliminated 4 lines of code
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$
```

```
 1 a = 3
 2 g = 8
 3 n = 5
 4 k = 2
 5 t1 = 4
 6 k = t1
 7 i = 5
 8 L1:
 9 t3 = i > 0
10 if  t3  goto L2
11 goto L3
12 L4:
13 t4 = i - 1
14 i = t4
15 goto L1
16 L2:
17 t5 = 2 * i
18 k = t5
19 t6 = k + 1
20 g = t6
21 goto L4
22 L3:
```

Architecture of Language

The syntax is almost the same as C++ for the constructs(for and switch case) given to us. An error is thrown if a bracket is missing, if a semicolon is missing.

The main tools used in the project include LEX which identifies predefined patterns and generates tokens for the patterns matched and YACC which parses the input for semantic meaning and prints out whether parsing is complete or not. An error is thrown if a bracket is missing, if a semicolon is missing. Intermediate code for the source code. PYTHON is used to optimize the intermediate code generated by the parser.
The semantics of the language implemented is close to the semantics of C++. We have made sure to throw meaningful error messages when a variable is undefined and if it is undeclared.

C++ constructs implemented:
1. For - loop
2. Switch case

Arithmetic expressions with +, -, *, /, ++, -- are handled
Boolean expressions with >,<,>=,<=,== are handled
Error handling reports undeclared variables
Error handling also reports syntax errors with line number

Literature Survey

https://www.lysator.liu.se/c/ANSI-C-grammar-y.html
http://cse.iitkgp.ac.in/~bivasm/notes/LexAndYaccTutorial.pdf

Context Free Grammar

The following is the CFG used -

```
Prog  : Prog ext_Decl
      | ext_Decl
      ;
ext_Decl  : func_defn
          | Decl
          ;

Decl      : TYPE assign';'
          | assign ';'
          | TYPE T_ID number ';'
          | T_ID number ';'
          ;

number    :  number T_OP T_NUM T_CP
             | T_OP T_NUM T_CP
             | number T_OP T_ID T_CP
             | T_OP T_ID T_CP
             ;


assign    :    T_ID  T_BOP1 evaluate
             | T_ID  T_BOP  assign
```

```
                | T_ID T_UOP
                | T_UOP T_ID
                | T_ID ',' assign
                | T_NUM ',' assign
                | T_ID number T_BOP1 assign
                | T_ID number T_BOP assign
                | T_NUM
                | T_ID
                |  '{' assign '}'
                ;

evaluate  : E
                E : E T_ADD E
                  | E T_SUB E
                  | E T_MULT E
                  | E T_DIV E
                  | '(' E ')'
                  | T_NUM
                  | T_ID
                  |'{' assign '}'
                ;


func_defn :  TYPE T_MAIN '(' ')'  codeblock
               ;

codeblock : '{' stmt_list '}'
               | '{' stmt_list defaultstm '}'
               ;

TYPE        : T_INT
               | T_FLOAT
               | T_CHAR
               ;

stmt_list:    stmt_list stmt
                | stmt
                ;
```

```
stmt:     loop
        | switch
        | caselist
        | Decl
        | printf
        | ret
        |';'
        ;


ret : T_RET T_NUM ';'
     |  T_RET T_ID ';'
      ;

loop : T_FOR '('  exp ‘;’ exp ‘;’ exp ‘;’  ')'  codeblock
   ;

printf : T_COUT T_PR  exp ';'

switch : T_SWITCH '(' T_ID ')'   codeblock
       ;

caselist    : casestm
            ;

casestm    : T_CASE T_NUM  ':' stmt_list T_BREAK ';'
            ;

defaultstm  :    T_DEFAULT  ':' stmt1 T_BREAK ';'
              ;
```

Design Strategy

Phase 1: (a) **Lexical Analysis**

- LEX tool was used to create a scanner for C++ language

- The scanner transforms the source file from a stream of bits and bytes into a series of meaningful tokens containing information that will be used by the later stages of the compiler.
- The scanner also scans for the comments (single-line and multiline comments) and writes the source file without comments onto an output file which is used in the further stages.
- All tokens included are of the form T_.Eg: T_ID for 'identifier', T_SUB for '-', T_ADD for '+' etc.
- A global variable 'yylval' is used to record the value of each lexeme scanned. 'yytext' is the lex variable that stores the matched string.
- Skipping over white spaces and recognizing all keywords, operators, variables and constants are handled in this phase.
- Scanning error is reported when the input string does not match any rule in the lex file.
- The rules are regular expressions which have corresponding actions that execute on a match with the source input.

Phase 1: (b)**Syntax Analysis**

- Syntax analysis is only responsible for verifying that the sequence of tokens forms a valid sentence given the definition of Programming Language grammar.
- The design implementation supports
    1. Variable declarations and initializations
    2. Variables of type int,float and char
    3. Arithmetic and boolean expressions
    4. Constructs - for loop and switch case
- Yacc tool is used for parsing. It reports shift-reduce and reduce-reduce conflicts on parsing an ambiguous grammar.

Phase 2:

1. **Symbol table with expression evaluation**
    - A structure is maintained to keep track of the variables, constants and the keywords in the input. The parameters of the structure are

the name of the token, the line number of occurence, the data type of the token (int, char, etc..),the value that it holds, scope of the variable.

- As each line is parsed, the actions associated with the grammar rules are executed.
- $1 is used to refer to the first token in the given production and $$ is used to refer to the resultant of the given production.
- Expressions are evaluated and the values of the used variables are updated accordingly.
- At the end of the parsing, the updated symbol table is displayed.

2. Intermediate Code Generation (ICG)

Intermediate code generator receives input from its predecessor phase, semantic analyzer after successful parsing. That syntax tree then can be converted into a linear representation. Intermediate code tends to be machine independent code.

Three-Address Code –
A statement involving no more than three references (two for operands and one for result) is known as a three address statement. A sequence of three address statements is known as three address code. Three address statements are of the form x = y op z, here x, y, z will have an address (memory location).

Example – The three address code for the expression a + b * c + d :
T 1 = b * c
T 2 = a + T 1
T 3 = T 2 + d
T1, T2, T3 are temporary variables. The data structure used to represent Three address Code is the Quadruples. It is shown with 4 columns- operator, operand1, operand2, and result.

3. Code Optimization:

The python file which has a separate functionalities to implement code optimization methods. This file is used to perform **constant propagation and constant folding** in sequential blocks followed by **dead code elimination**. Then performing **elimination of subexpression** and **move loop invariant code outside.**

4. Error handling:

- Detection of an undeclared variable, showing an appropriate error.
- Detection of invalid syntax at line particular line, showing an appropriate error message with line number.
- Detection of an undefined variable, showing an appropriate error.
- Detection of a redeclaration variable, showing an appropriate error.

Implementation details

Symbol Table Generation:

We are using a hash table to implement our symbol table. Every time any variable or temporaries are identified it gets updated to the hash table by calling insert and insert_temp functions respectively.
Structure is used to store the content of a single node and update it to the symbol table.

Symbol table contains:

- name
- value
- datatype
- line
- scope

All the variables and the temporaries used in the program and created during three address code generation are updated to the symbol table.

Intermediate Code Generation:

Intermediate Code Generation is implemented using various functions and data structures. Stacks are used to keep track of the temporaries and their values and labels that are required for three address code generation. A

structure (quadruples) is used to store the quadruples generated from the three address code. As and when a three address code line is generated, an object of the structure quadruple is used to store this line in Quadruple format by calling the necessary function.

Code Optimization:

Python has been used to optimize the three address code generated. A number of functions, lists and dictionaries have been used for this process. The three address code is first checked for unreachable code and dead code, if found they are removed. Unused variables are removed in the next step. Temporaries that are never assigned to any variable nor used in any expression are deleted. In the next step common subexpressions are removed after which constant propagation and folding functions are implemented. Finally, loop invariant codes are moved outside the loop.

Error Handling:

- Detection of an undeclared variable, showing an appropriate error.
- Detection of invalid syntax at line particular line, showing an appropriate error message with line number.
- Detection of an undefined variable, showing an appropriate error.
- Detection of a redeclaration variable, showing an appropriate error.

Steps to run the file :

Phase 1 :

```
lex lex.l
bison -d yacc.y
g++ yacc.tab.c lex.yy.c -ll -ly -w
./a.out
```

File names : lex file - lex.l, yacc file - yacc.y

Phase 2 :

lex lex.l
bison -d yacc.y
g++ yacc.tab.c lex.yy.c -ll -ly -w
./a.out

File names : lex file - lex.l, yacc file - yacc.y
Name of the input file is in yacc file

Intermediate code generated is copied to icg.txt

Step to run code optimization file :
python3 co.py icg.txt
(python3 co.py “icg file name.txt”)

Results

Our Mini Compiler has been able to compile and generate code for the sample input files. It can detect a number of errors and throw appropriate error messages. It can satisfactorily compile and produce optimal code for and switch constructs and some expressions.

Possible Shortcoming

The compiler we built is a mini-compiler and it doesn’t entirely mimic or compile all C++ code. We haven’t implemented all the constructs(functions included) present in C++. The code that we have generated has been optimized specifically for the grammar implemented by us.

The symbol table structure is the same across all types of tokens (constants, identifiers and operators). This leads to some fields being empty for some of the tokens. This can be optimized by using a better representation.

Snapshots

Program without errors:

Input Program:

```
1 #include<iostream>
2 using namespace std;
3 int main()
4 {
5
6         int button=2;
7         int a=3;
8         a = 2 + a;
9         int b = (2 + a) *7;
10        int c = a;
11        int i = 6;
12
13        switch(button)
14        {
15                case 1:
16                        b=2;
17                        for(int j =0;j<a;j=j+1)
18                        {
19                                int sum= i * 2 + j;
20                                c = j;
21                        }
22                        break;
23                case 2:
24                        int d=1;
25                        break;
26                case 3:
27                        a=1;
28                        break;
29                case 4:
30                        int k = 1;
31                        break;
32                default:
33                        int y = 7;
34                        break;
35        }
36        return 0;
37 }
```

Output:

Three address code

```
--------------Three address code--------------
button = 2
a = 3
t1 = 2 + a
a = t1
t2 = 2 + a
t3 = t2 * 7
b = t3
c = a
i = 6
t4 = button
goto test
L1:
b = 2
j = 0
L2:
t5 = j < a
if  t5  goto L3
goto L4
L5:
t6 = j + 1
j = t6
goto L2
L3:
t7 = i * 2
t8 = t7 + j
sum = t8
c = j
goto L5
L4:
goto last
L6:
d = 1
goto last
L7:
a = 1
goto last
L8:
k = 1
goto last
L9:
y = 7
goto last
test:
if (button==1) goto L1
if (button==2) goto L6
if (button==3) goto L7
if (button==4) goto L8
goto L9
last: end

Parsing complete
```

Quadruple Format

----------Quadruples----------

Operator	Arg1	Arg2	Result
=	2	NULL	button
=	3	NULL	a
+	2	a	t1
=	t1	NULL	a
+	2	a	t2
*	t2	7	t3
=	t3	NULL	b
=	a	NULL	c
=	6	NULL	i
=	button	NULL	t4
goto	NULL	NULL	test
label	NULL	NULL	L1
=	2	NULL	b
=	0	NULL	j
label	NULL	NULL	L2
<	j	a	t5
if	0	NULL	L3
goto	NULL	NULL	L4
label	NULL	NULL	L5
+	j	1	t6
=	t6	NULL	j
goto	NULL	NULL	L2
label	NULL	NULL	L3
*	i	2	t7
+	t7	j	t8
=	t8	NULL	sum
=	j	NULL	c
goto	NULL	NULL	L5
label	NULL	NULL	L4
goto	NULL	NULL	last
label	NULL	NULL	L6
=	1	NULL	d
goto	NULL	NULL	last
label	NULL	NULL	L7
=	1	NULL	a
goto	NULL	NULL	last
label	NULL	NULL	L8
=	1	NULL	k
goto	NULL	NULL	last
label	NULL	NULL	L9
=	7	NULL	y
goto	NULL	NULL	last
label	NULL	NULL	test
if	button==1	NULL	L1
if	button==2	NULL	L6
if	button==3	NULL	L7
if	button==4	NULL	L8
goto	NULL	NULL	L5
label	NULL	NULL	last

Symbol Table

----------Symbol table----------

Slno.	Name	Value	Datatype	lineno	scope
0	a	1	temp	27	2
1	k	1	int	30	2
2	y	7	int	33	2
3	return	undefined	keyword	36	1
5	a	t1	temp	8	1
6	t2	2+a	temp	9	1
7	b	49	int	9	1
8	c	5	int	10	1
9	i	6	int	11	1
10	t4	button	temp	13	1
11	b	2	int	16	2
12	j	4	int	17	3
14	t5	j<a	temp	17	3
15	t7	i*2	temp	19	3
16	sum	16	int	19	3
17	c	4	int	20	3
18	d	1	int	24	2
19	a	1	int	27	2

Optimized code for the above program:

```
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$ python3 co.py icg2.txt
Eliminated 7 lines of code
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$
```

On subexpression removal and before constant propagation and folding

```
 1 button = 2
 2 a = 3
 3 t1 = 2 + a
 4 a = t1
 5 t2 = t1
 6 t3 = t2 * 7
 7 b = t3
 8 c = a
 9 i = 6
10 goto test
11 L1:
12 b = 2
13 j = 0
14 L2:
15 t5 = j < a
16 if  t5  goto L3
17 goto L4
18 L5:
19 t6 = j + 1
20 j = t6
21 goto L2
22 L3:
23 c = j
24 goto L5
25 L4:
26 goto last
27 L6:
28 goto last
29 L7:
30 a = 1
31 goto last
32 L8:
33 goto last
34 L9:
35 goto last
36 test:
37 if (button==1) goto L1
38 if (button==2) goto L6
39 if (button==3) goto L7
40 if (button==4) goto L8
41 goto L9
42 last: end
```

On complete optimization

```
 1 button = 2
 2 a = 3
 3 t1 = 5
 4 a = t1
 5 t2 = t1
 6 t3 = t2 * 7
 7 b = t3
 8 c = 3
 9 i = 6
10 goto test
11 L1:
12 b = 2
13 j = 0
14 L2:
15 t5 = j < 3
16 if  t5  goto L3
17 goto L4
18 L5:
19 t6 = j + 1
20 j = t6
21 goto L2
22 L3:
23 c = j
24 goto L5
25 L4:
26 goto last
27 L6:
28 goto last
29 L7:
30 a = 1
31 goto last
32 L8:
33 goto last
34 L9:
35 goto last
36 test:
37 if (button==1) goto L1
38 if (button==2) goto L6
39 if (button==3) goto L7
40 if (button==4) goto L8
41 goto L9
42 last: end
```

Program with undefined variable error:

Input file:

```
1 #include<iostream>
2 using namespace std;
3 int main()
4 {
5
6         int button=2;
7         int a=3;
8         a = a + 9;
9         int b = (2 + a) *7;
10        c = a;
11        int a =4;
12        switch(button)
13        {
14                case 1:
15                        a = 0;
16                        break;
17                        int s = 0;
18                case 2:
19                        a=1;
20                        break;
21                default:
22                        a = 7;
23                        break;
24        }
25        return 0;
26 }
```

Output:

```
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$ lex lex.l
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$ bison -d yacc.y
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$ g++ yacc.tab.c lex.yy.c -ll -ly -w
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$ ./a.out
--------------Three address code---------------
button = 2
a = 3
t1 = a + 9
a = t1
t2 = 2 + a
t3 = t2 * 7
b = t3
c = a
Error parsing failed!

Variable undefined at line number : 10
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$
```

Program with redeclaration error:

Input File:

```
#include<iostream>
using namespace std;
int main()
{

        int button=2;
        int button=4;
        int a=3;
        a = a + 9;
        int b = (2 + a) *7;
        int c = a;
        int a =4;
        switch(button)
        {
                case 1:
                        a = 0;
                        break;
                        int s = 0;
                case 2:
                        a=1;
                        break;
                default:
                        a = 7;
                        break;
        }
        return 0;
}
```

Output:

```
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$ lex lex.l
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$ bison -d yacc.y
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$ g++ yacc.tab.c lex.yy.c -ll -ly -w
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$ ./a.out
--------------Three address code--------------
button = 2
button = 4
Error parsing failed!

Redeclaration error at line number : 7
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$
```

Program with syntax error:

Input file

```
#include<iostream>
using namespace std;
int main()
{

        int button=2;
        int a=3
        a = a + 9;
        int b = (2 + a) *7;
        int c = a;
        int a =4;
        switch(button)
        {
                case 1:
                        a = 0;
                        break;
                        int s = 0;
                case 2:
                        a=1;
                        break;
                default:
                        a = 7;
                        break;
        }
        return 0;
}
```

Output

```
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$ lex lex.l
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$ bison -d yacc.y
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$ g++ yacc.tab.c lex.yy.c -ll -ly -w
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$ ./a.out
--------------Three address code---------------
button = 2
a = 3
syntax error at line no : 7

Parsing failed

nisha@nisha-VirtualBox:~/Desktop/cd phase 2$
```

Program for invariant code removal:

Input file: (Here in for loop, g = x + 1 needs to be moved outside the loop)

```
#include<iostream>
using namespace std;
int main()
{
        int a=3;
        int b=4;
        int i;
        int g = 8;
        int n = 5;
        int k = 2;
        k = a+1;
        int x = a+1;
        for(i=n;i>0;i=i-1)
        {
                k = i+1;
                g = x+1;
        }
        int c = a;
        return c;
}
```

Output:

```
--------------Three address code---------------
a = 3
b = 4
g = 8
n = 5
k = 2
t1 = a + 1
k = t1
t2 = a + 1
x = t2
i = n
L1:
t3 = i > 0
if  t3  goto L2
goto L3
L4:
t4 = i - 1
i = t4
goto L1
L2:
t5 = i + 1
k = t5
t6 = x + 1
g = t6
goto L4
L3:
c = a

Parsing complete
```

On optimisation:

```
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$ python3 co.py icg.txt
Eliminated 2 lines of code
nisha@nisha-VirtualBox:~/Desktop/cd phase 2$
```

```
 1 a = 3
 2 g = 8
 3 n = 5
 4 k = 2
 5 t1 = a + 1
 6 k = t1
 7 t2 = t1
 8 x = t2
 9 t6 = x + 1
10 g = t6
11 i = n
12 L1:
13 t3 = i > 0
14 if  t3  goto L2
15 goto L3
16 L4:
17 t4 = i - 1
18 i = t4
19 goto L1
20 L2:
21 t5 = i + 1
22 k = t5
23 goto L4
24 L3:
```

Here we can see that lines
t6 = x + 1
g = t6
corresponding to g = x + 1 is moved outside the loop

Conclusion

This project is based on the implementation of the different phases involved while executing a C++ program. The constructs given to us were

for and **switch case** and our language was C++. The deliverables from this project include

- the lexical analyser
- syntax validation
- generation and updation of symbol table
- evaluation of expressions
- Intermediate Code Generation
- quadruple format
- 4 Code optimizations

Further Enhancements

We have included only two constructs as already mentioned. Further enhancements could be the addition of more constructs including functions. We could modify the grammar file to mimic the C++ compiler and also throw meaningful errors whenever an error is encountered (under every erroneous situation).

References

1. Lex and Yacc: A Brisk Tutorial Saumya K. Debray, Department of Computer Science, The University of Arizona, Tucson, AZ 85721
2. LEX & YACC TUTORIAL by Tom Niemann
3. Here in for loop, g = x + 1 needs to be moved outside the loopMastering Regular Expression, Third Edition, Jeffrey E. F. Friedl