C++ Mini Compiler

Innovative Assignment

Compiler Construction [2CS701]

B.Tech in
Computer Science & Engineering

Submitted by

18BCE191	**Rajat Patel**
18BCE195	**Dhruv Ravaliya**

Under the guidance of
Prof. Monika Shah

1. Introduction

The project aims to develop a mini compiler for the C++ programming language. It focuses on the lexical, syntactical, semantical analysis along with the generation of intermediate code. The generation of code is done in the following steps :

- Generate symbol table after performing expression evaluation (Phase-1)
- Generate Abstract Syntax Tree for the code (Phase-2)
- Generate the intermediate code quadruple form (Phase-2)

Major tools used are **Flex** and **Bison**. Code is tested on the ubuntu 20.04 platform operating system.

2. Structure of language

C++ consists of control statements, loop statements, and different objects and methods such as

- If
- If-else along with else if
- Ternary operator
- While loop
- For-loop
- Class (used only in phase-1)
- Functions
- Arithmetic expressions +, -, *, /, ++, --
- Boolean expressions with >,<,>=,<=,==
- Error handling reports undeclared variables
- Error handling reports for redeclared variables
- Error handling also reports syntax errors with line numbers

3. Design phase and Implementation

Here, we lay out different phases which will at the end result into an intermediate code.

- symbol table generation (Phase-1)
    - lexical analysis
    - syntax analysis
    - semantic analysis

- abstract syntax tree generation (Phase-2)
- Intermediate code generation (Phase-2)

a. Phase-1 (Output is symbol table)

Lexical analysis

In this phase sequence of characters is converted into a stream of tokens.

In this part, we use FLEX to scan the whole C++ program and transform the source file into tokens which will be used in later steps. These tokens consist of operators, symbols, keywords, identifiers, and white space.

Syntax analysis

In this phase, the written code is verified syntactically. In this phase, the input file is tested against the grammar defined in the '.y' file for its correctness. If the text in the input file can be derived from the grammar then this step is successfully completed.

Semantic Analysis

In this phase, we generate the symbol table. Here the scope of the variables is checked along with their datatype and definition. If the variable is not declared or it is redeclared in the same scope then the error is prompted to the user and parsing fails. Here symbol table is maintained as a list, where each entry is the node of the list. The attributes of a row are scope, value, name, datatype along with the line number where it is declared.

b. Phase-2 (Output is AST and intermediate code)

Abstract Syntax Tree generation

In this part of the project, we generate AST(abstract syntax tree) is generated. An AST is a tree structure representing the linguistic flow of code. Its Nodes structure consists of a container for its data values and its children nodes pointers. It outputs the tree in the preorder traversal. It is an n-ary tree that consists of multiple children depending on the statement type. It's difficult to visualize it therefore we just printed its preorder traversal.

Intermediate code generation

In the final step, we generate intermediate code based on the AST received from its predecessor process. The intermediate code is represented by a Three-address code which is a statement involving at most 2 operators. It's described by the quadruple data structure which describes the statement as an operator, operand1, operand2, and the result.

4. Github Link for the Code

Code Link

5. ScreenShots of the Demo

a. Sample Test case 1

Sample Input

```
#include<iostream>

int main()
{
   int x = 5;
   while(x>0){
       for(int j=0;j<10;j++){
           cout << j << (x++);
       }
   }
   return 0;
}
```

Sample Output

```
1       #include        PREPROCESSOR DIRECTIVE
1       <       RELATIONAL OPERATORS
1       stdio.h PREPROCESSOR
1       >       RELATIONAL OPERATORS
3       int     TYPE
3       main()  KEYW
4       {       OPEN BRACES
5       int     TYPE
5       x       ID
5       =       ASSIGNMENT
5       5       INTEGER
5       ;       TERMINATOR
6       while   KEYW
6       (       OPEN BRACKETS
6       x       ID
6       >       RELATIONAL OPERATORS
6       0       INTEGER
6       )       CLOSE BRACKETS
6       {       OPEN BRACES
7       for     KEYW
7       (       OPEN BRACKETS
7       int     TYPE
7       j       ID
7       =       ASSIGNMENT
7       0       INTEGER
7       ;       TERMINATOR
entry: x7       j       ID
```

```
Successful parsing.
Parsing successful


======================================== Symbol table ========================================
Token           Data type       Scope           Value           Line number
x               int             2               5               5
j               int             4               0               7
==============================================================================================
```

b. Sample Test case 2

Sample Input (nested if and if-else statement)

```
#include<iostream.h>

int main()
{
   int a = 5;
   cout<<"The value of a is "<<a;
   cin>>a;
   float b;
   char c;
   cin>>b>>c;
   int x;
```

```
    int y = 7; int z = 8;
    x = y+z; int final;
    if(x > y)
    {
        if(x > z)
        {
            cout<<"x is the greatest";
            final = x;
        }
    }
    else
    {
        if(y > z)
        {
            cout<<"y is the greatest";
            final = y;
        }
    }
    return 0;
}
```

Sample Output

```
Parsing successful


===================================== Symbol table =======================================
Token           Data type       Scope           Value           Line number
a               int             2               5               5
b               float           2               0               8
c               char            2               0               9
x               int             2               15              11
y               int             2               7               12
z               int             2               8               12
final           int             2               7               13
==========================================================================================
```

c. Sample Test case 3 with nested loops

Input

```
#include<iostream.h>

int main()
{
    int n = 10;
    for(int i=0;i<n;i++){
        for(int j=i+1;j<n;j++){
            if(i<j){
                continue;
            }
```

```
            else if(i==j){
                float testVar = 1.2;
                cout << "i and j are equal";
            }
            else{
                break;
            }
        }
    }
    return 0;
}
```

Sample Output (Phase-1)

```
Parsing successful


===================================== Symbol table =======================================
Token           Data type       Scope           Value           Line number
n               int             2               10              5
i               int             3               0               6
j               int             5               1               7
testVar         float           7               0               12
==========================================================================================
```

d. Sample Test Case 4 with a ternary expression

Input

```
#include<stdio.h>

int main()
{
   int a = 10;
   int b=20;
   int max = (a > b ) ? a: b ;
}
```

Output

```
Parsing successful


===================================== Symbol table =======================================
Token           Data type       Scope           Value           Line number
a               int             2               10              5
b               int             2               20              6
max             int             2               -314238912              7
==========================================================================================
```

e. Sample Test Case 5 with 2 variables of the same name but different scope

Input

```c
#include<stdio.h>

int main(){
   int n = 10;
   for(int i=0;i<n;i++){
       n--;
   }
   n=20;
   int a = 20;
   int b = 20;
   while(n>0){
       a=n*n;
       n--;
       int b = 10;
   }
   return 0;
}
```

Output

```
Parsing successful


======================================== Symbol table ========================================
Token           Data type       Scope           Value           Line number
n               int             2               20              4
i               int             3               0               5
a               int             3               400             9
b               int             3               20              10
b               int             4               10              14
==============================================================================================
```

f. Function and class demo for phase-1

Input

```c
#include<stdio.h>

class test{
  int b = 20;

};
int sum(int a,int b){
   return a+b;
```

```
}

int main(){

   int a = 10;

   return 0;
}
```

Output

```
Parsing successful


======================================== Symbol table ========================================
Token           Data type       Scope           Value           Line number
b               int             2               20              4
b               int             1               0               7
a               int             1               0               7
a               int             2               10              13
==============================================================================================
```

g. Variable redeclared error

Input

```
#include<stdio.h>

int main(){
   int n = 10;
   for(int i=0;i<n;i++){
       n--;
   }
   n=20;
   int a = 20;
   int b = 20;
   while(n>0){
       a=n*n;
       n--;
       int b = 10;
   }
   int b = 30;
   return 0;
}
```

Output

```
ERROR: line number: 16 - error: Variable redeclared


17      return  KEYW
17      0       INTEGER
17      ;       TERMINATOR
18      }       CLOSE BRACES
Successful parsing.
Parsing unsuccessful


===================================== Symbol table =======================================
Token           Data type       Scope           Value           Line number
n               int             2               20              4
i               int             3               0               5
a               int             3               400             9
b               int             3               30              10
b               int             4               10              14
==========================================================================================
(base) rajat@rajat-VivoBook-S14-X430UA:/Rajat1/Books/Compiler Construction/Innovative Assignment/CC-Innovative/Phase-1$
```

h. Variable not declared error

Input

```c
#include<stdio.h>
int main(){
   int n = 10;
   int sq = n*n;
   int cb = n*n*n;
   int res=0;

   if(sq == cb && cb == n){
       res = 0;
   }
   else if(sq > cb && n > sq){
       res= 2;
   }
   else{
       res=1;
   }
   x=res;
   return 0;
}
```

Output

```
ERROR: line number: 18 - error: variable not declared


19      return  KEYW
19      0       INTEGER
19      ;       TERMINATOR
20      }       CLOSE BRACES
Successful parsing.
Parsing unsuccessful


===================================== Symbol table =======================================
Token           Data type       Scope           Value           Line number
n               int             2               10              4
sq              int             2               100             5
cb              int             2               1000            6
res             int             2               1               7
==========================================================================================
```

i. AST demo for simple code

Input

```
#include<iostream>


int main()
{
   int a = 10;
   int b = 20;
   int c = (a+b);


}
```

Output

```
14
15 Abstract Syntax Tree
16
17                                                  main
18
19
20                                              stmt
21
22
23                                          stmt            =
24
25
26                                      =           =           c           +
27
28
29                                  a           10          b           20                  a           b
30
31
32
33
34
35
36
37
38
39
40 Preorder Traversal
41
42 main      ( stmt      ( stmt      ( =   a   10   )      ( =   b   20   )   )      ( =   c      ( +   a   b   )   )   )
```

```
a = 10
b = 20
T0 = a + b
c = T0
============================== Intermediate Code Generation (Quadruple Form) ==============================

-----------------------------------------------------------------------------------------------
=               10              (null)          a
=               20              (null)          b
+               a               b               T0
=               T0              (null)          c

-----------------------------------------------------------------------------------------------
```

j. AST and intermediate code for loops

Input

```c
int main()
{
   int a = 4;
   for(int i = 1; i < 10; i = i + 1)
   {
        a=a+1;
        for(int j = 2; j < 10; j = j + 2)
        {
            int c = a*a;
        }
   }
}
```

Output

```
14
15
16 Abstract Syntax Tree
17
18                                              main
19
20
21                                          stmt
22
23
24                                  =       for
25
26
27                              a       +       =       stmt
28
29
30                             a       1       j       2       <       for
31
32
33
34
35 Preorder Traversal
36
37 main      ( stmt      ( =   a      ( +   a   1   )   )      ( for      ( =   j   2   )      ( stmt      ( <   j   10   )      ( for      ( =   j      ( +   j
   2   )   )      ( =   c      ( *   a   a   )   )   )   )   )   )
```

```
============================ Intermediate Code Generation (Quadruple Form) ============================

-----------------------------------------------------------------------------------------------
=               4               (null)          a
=               1               (null)          i
Label           (null)          (null)          L0
<               i               10              T0
not             T0              (null)          T1
if              T1              (null)          L1
goto            (null)          (null)          L2
Label           (null)          (null)          L3
+               i               1               T2
=               T2              (null)          i
goto            (null)          (null)          L0
Label           (null)          (null)          L2
+               a               1               T3
=               T3              (null)          a
=               2               (null)          j
Label           (null)          (null)          L4
<               j               10              T4
not             T4              (null)          T5
if              T5              (null)          L5
goto            (null)          (null)          L6
Label           (null)          (null)          L7
+               j               2               T6
=               T6              (null)          j
goto            (null)          (null)          L4
Label           (null)          (null)          L6
*               a               a               T7
=               T7              (null)          c
goto            (null)          (null)          L7
Label           (null)          (null)          L5
goto            (null)          (null)          L3
Label           (null)          (null)          L1

-----------------------------------------------------------------------------------------------
```

k. AST and intermediate code for if-else statements

Input

```c
int main()
{
   int a = 4;
   int c = 5,b,d;
   if(a < 5)
   {
        if(a < 3)
        {
            int e = 5;
        }
        b = 6;
   }
   else
   {
        d = 4;
   }
}
```

Output

```
Abstract Syntax Tree

                                        main


                                    stmt


                                =       else


                            c       5       stmt       =


                               stmt       if      d       4



Preorder Traversal

main     ( stmt     ( =   c   5   )     ( else     ( stmt     ( stmt   Dc b   Dc d   )     ( if     ( <   a   5   )     ( stmt     ( if     ( <   a
  3   )     ( =   e   5   )   )     ( =   b   6   )   )   )   )     ( =   d   4   )   )   )
```

```
============================== Intermediate Code Generation (Quadruple Form) ==============================

-------------------------------------------------------------------------------------------------------
=               4               (null)          a
=               5               (null)          c
=               0               (null)          b
=               0               (null)          d
<               a               5               T0
not             T0              (null)          T1
if              T1              (null)          L0
<               a               3               T2
not             T2              (null)          T3
if              T3              (null)          L1
=               5               (null)          e
Label           (null)          (null)          L1
=               6               (null)          b
Label           (null)          (null)          L0
=               4               (null)          d

-------------------------------------------------------------------------------------------------------
```

I. Error detection if some token is missing where it is expected

Input

```
int main(){
   int n = 10;
   for(int i=0 ; i<n;i=i+1){
        n=n-1;
   }
   n=20;
   int a = 20;
   int b = 20;
   while(n>0){
        a=n*n;
        n=n-1
        int b = 10;
```

```
    }
    return 0;
}
```

output

```
ERROR: line number: 12 - error: syntax error, unexpected INT, expecting TERMINATOR


12      b       ID
12      =       ASSIGNMENT
12      10      INTEGER
12      ;       TERMINATOR
13      }       CLOSE BRACES
14      return  KEYW
14      0       INTEGER
14      ;       TERMINATOR
15      }       CLOSE BRACES
Successful parsing.
Parsing unsuccessful


======================================== Symbol table ========================================
Token           Data type       Scope           Value           Line number
n               int             2               19              2
i               int             3               1               3
a               int             3               400             7
b               int             3               20              8
==============================================================================================
```

6. Steps to run the code

Command: For each module just run a shell script using the command **‘sh start.sh’**.

Every module has run.sh script which includes all the commands to compile and run the flex and bison files.

Phase-1
From Phase-1 directory run command
sh start.sh <path to input-file.cpp>

Example: sh start.sh Inputs/simple-code.cpp

Phase-2 (Abstract syntax tree)
From Phase-2/”Abstract Syntax Tree” directory run command
sh start.sh <path to input-file.cpp>

Example: sh start.sh. Inputs/loops.cpp

Phase-2 (Quadruple form)
From Phase-2/”Intermediate Code Generation” directory run command

sh start.sh <path to input-file.cpp>

Example: sh start.sh. Inputs/if-else.cpp

7. Limitations

- Doesn’t have all cpp features like namespace, stl library, access modifiers, and many more.
- Phase-2 is quite difficult to implement, therefore only limited language constructs are supported by phase-2.
- Error handling mechanism only detects certain errors like redeclaration, variables not declared or expected some another token. But it doesn’t detect some errors related to type checking.

8. Conclusion

Although we are able to develop a very basic compiler with only 2 phases. It is a great learning experience. We realized the amount of effort and logic we need to apply while developing a compiler. We understood the importance of every phase of the compiler in the code compilation. We went through lexical analysis, syntax analysis, semantic analysis, abstract syntax tree generation, and finally generating the intermediate code.

It is quite challenging to design the mini compiler. Still, there are certain issues with the compiler which we can be rectified and improve in the future.