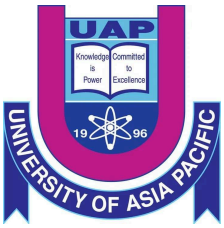


University of Asia Pacific

Department of Computer Science & Engineering

Course code: CSE 430

Course Title: Compiler Design Lab

MiniCompiler Design Project

Submitted To	**Submitted By**
Baivab Das Lecturer, CSE, UAP	Asma Sultana Id: 20101084 Section: B

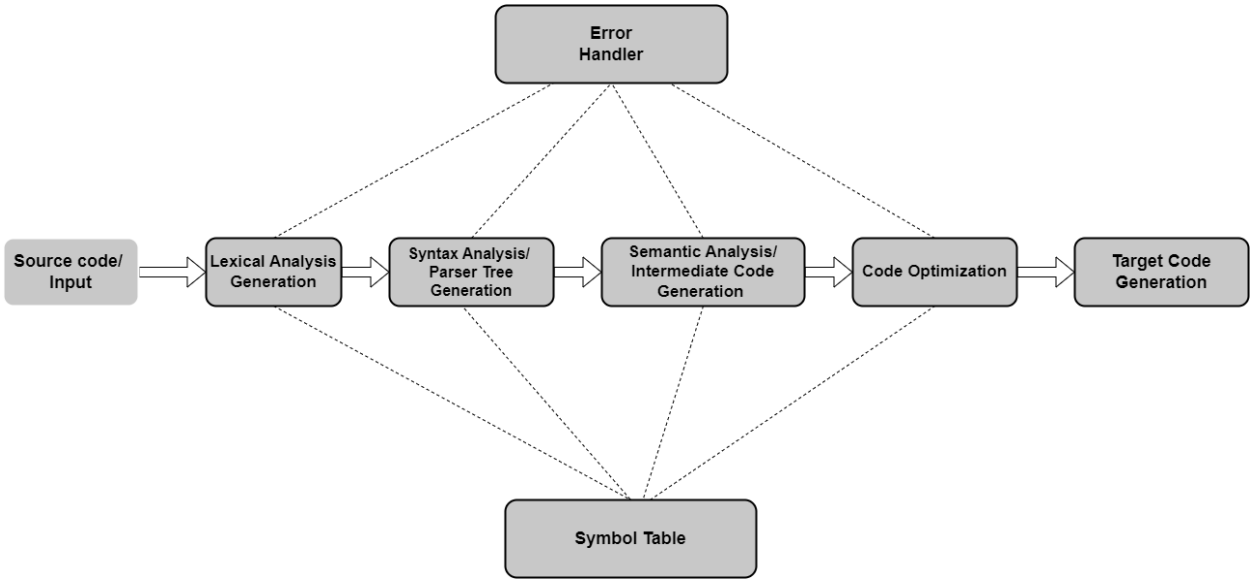
Introduction

This project is an individual effort to create a mini compiler for a simple programming language. The project aims to merge the various phases of a compiler, including lexical analysis, syntax analysis, semantic analysis, intermediate code generation, and code generation, by implementing the necessary functions and data structures.

Here, I create a mini compiler using Lex and Yacc in C++ for a small C++ and Python-like language. The compiler supports a simple programming language with features such as variables, expressions, control structures (if-else, loops), and functions. Here, outline the process and provide sample code for each phase.

Design

The design of the project involves the following components and flow:

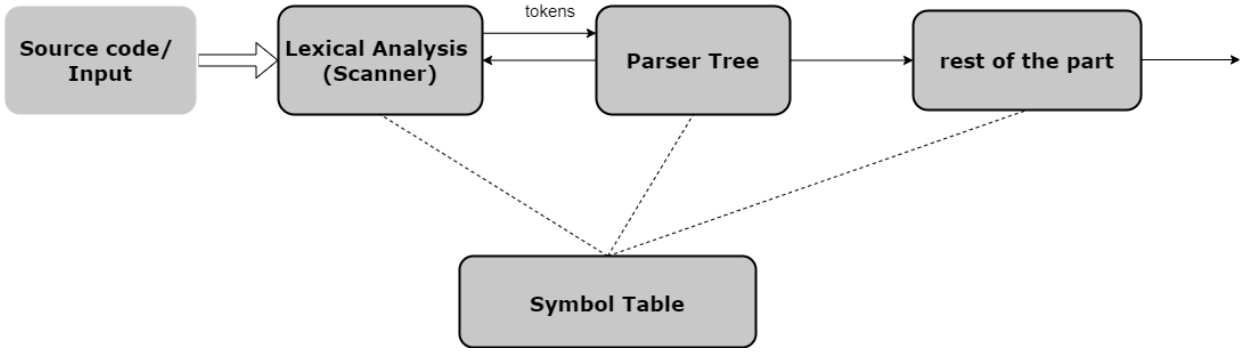


Lexical Analysis (Tokenizer)

In this part, tokenization of the source code into meaningful symbols (tokens).

Abstract Syntax Tree

I checked the tokens against the language's grammar and created a parse tree.



Intermediate Code Generation

Converts the parse tree into an intermediate representation (IR) with low-level representation- TAC.

A sequence of instructions of the form

Result = arg1 operator arg2

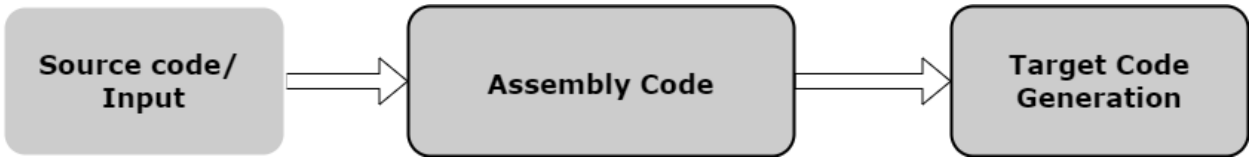
x = y * z

Code Optimization

Optimizing the TAC intermediate code to error handling is handled during the syntax validation phase of the compiler. After optimization, it eliminates the Dead Code.

Target Code Generation

Converts the intermediate representation into machine code or assembly.



Symbol Table Management

Keeps track of variable/function names and their attributes throughout the compilation process.

Implementation

Create and Run the program file:

```
touch filename
lex filename.l
yacc -d filename.y
./a.out < inputfile
```

Task1- Lexical Analysis (tokenization):

- Tools used: Lex and GCC
- Language used: C
- Data Structures: Custom struct in C that can recognize keywords, digits, identifiers, and comments
- Commands to compile and run the lexical phase:

```
lex token.l
gcc lex.yy.c
./a.out < intput0.txt
```

Task 2 - Symbol Table Creation:

- Tools used: Lex and GCC
- Language used: C
- Data Structures: Custom struct in C to hold symbol table entry details and an array of this structure is used to hold symbol table
- Commands to compile and run the lexical phase:

```
lex symbol.l
gcc lex.yy.c
./a.out < input0.cpp
```

Task 3 - Syntax Analysis:

- Tools used: Flex, Bison Yacc, and GCC
- Language used: C
- Data Structures: Handling syntax error

- Commands to compile and run the semantic phase:

```
lex lexical.l
yacc -d lexical.y
gcc lex.yy.c y.tab.c
./a.out <input0.cpp
```

Task 4- Parser/ Abstract Syntax Tree:

- Tools used: Flex, Bison Yacc, and GCC
- Language used: C
- Data Structures: Internally stored as a binary tree
- Commands to compile and run the semantic phase:

```
lex tree.l
yacc -d tree.y
gcc lex.yy.c y.tab.c
./a.out < input1.cpp
```

Task 5 - Intermediate Code Generation:

- Tools used: Flex, Bison Yacc, and GCC
- Language used: C
- Data Structures: No special data structures are used implicitly.
  - ○ Grammar rules are used to generate ICG component-wise
  - ○ Each statement of ICG is written in a text file
- Commands to compile and run the ICG generator:

```
lex tac.l
yacc -d tac.y
gcc lex.yy.c y.tab.c
./a.out < input0.cpp
```

Task 6 - Code Optimization:

- Tools used: Python
- Language used: Python
- Data Structures: Python list is used to hold a list of lines.
  - ○ ICG file is read and split into sentences and stored in the array
  - ○ After processing, these statements are written back to a text file line by line

- Commands to compile and run the ICG generator:

  ```
  python3 icg_optimize.py input.txt --print
  ```

Task 7 - Assembly Code Generation:

- Tools used: Python
- Language used: Python
- Data structures:
  - Hash map is used to store a variable list to ensure variables are not loaded from memory again and again.
  - Python list to store ARM-generated statements which are written to a .s ASM file after processing all ICG statements
- Commands to run the assembly generator:

  ```
  python assemble.py input22.txt
  ```

  Note: Assembly code will be stored in filename.s: input22.s

Results

Assembly code for the input code.

Sample input:

input22.txt

```
1 i = 2
2 t0 = i > 1
3 ifFalse t0 goto L0
4 t1 = i + 1
5 i = t1
6 goto L1
7 L0:
8 t2 = i - 1
9 i = t2
10 L1:
11 t3 = i + 3
12 i = t3
13 i = t3
14 L2:
15 t4 = i < 10
16 ifFalse t4 goto L3
17 t5 = i + 2
18 a = t5
19 t6 = i + 1
20 i = t6
21 goto L2
```

Assembly generate output:

```
asma@asma-virtualbox:~/Documents/
.text
MOV R0,=i
MOV R1,[R0]
CMP R1,#1
BLE L0
MOV R2,=i
MOV R3,[R2]
MOV R4,=t1
MOV R5,[R4]
ADD R5,#3,R1
STR R5, [R4]
MOV R6,=i
MOV R7,[R6]
MOV R8,#t1
STR R8, [R6]
B L1
L0:
MOV R9,=i
MOV R10,[R9]
MOV R11,=t2
MOV R12,[R11]
SUBS R12,#10,R1
STR R12, [R11]
MOV R0,=i
MOV R1,[R0]
MOV R2,#t2
STR R2, [R0]
L1:
MOV R3,=i
MOV R4,[R3]
MOV R5,=t3
MOV R6,[R5]
ADD R6,#4,R3
STR R6, [R5]
MOV R7,=i
MOV R8,[R7]
```

Result Snapshots

Task1- Lexical analysis (tokenization):

Input File

input0.txt

```
1 int main() {
2 a = 77
3 b = 33
4 result1 = a+ b+ 11
5 x = 60
6 y = 40
7 result2 = x * y * 2
8 if(a>=b){
9         result1= 0;
10        a= a+b;
11 }
12 else {
13        a = a + 1;
14        Result3 = x*10
15 }
16 m=9
```

Output

```
asma@asma-virtualbox:~/Documents/Compiler430-MiniCompiler_Project/Task1- Lexical$ lex token.l
asma@asma-virtualbox:~/Documents/Compiler430-MiniCompiler_Project/Task1- Lexical$ gcc lex.yy.c
asma@asma-virtualbox:~/Documents/Compiler430-MiniCompiler_Project/Task1- Lexical$ ./a.out<input0.txt
int - keyword
main - identifier
a - identifier
77 - digit
b - identifier
33 - digit
result1 - identifier
11 - digit
x - identifier
60 - digit
y - identifier
40 - digit
result2 - identifier
2 - digit
if - keyword
0 - digit
else - keyword
1 - digit
Result3 - identifier
10 - digit
m - identifier
asma@asma-virtualbox:~/Documents/Compiler430-MiniCompiler_Project/Task1- Lexical$
```

Task 2 - Symbol Table:

Input file

input0.cpp

```
1 int main() {
2 a = 77;
3 b = 33;
4 result1 = a+ b+ 11;
5 x = 60;
6 y = 40;
7 result2 = x * y * 2;
8 if(a>=b){
9         result1= 0;
10        a= a+b;
11 }
12 else {
13        a = a + 1;
14        Result3 = x*10;
15 }
16 m=9;
```

```
<bigit,9,16>

TOKEN#  DATA TYPE       TOKEN_TYPE      TOKEN_VALUE     LINE of CODE            DIMENSION       ADDRESS
1       int             IDENTIFIER      a               2 4 8 10 10 13 13                       1               9999999
2       int             IDENTIFIER      b               3 4 8 10                        1               9999999
3       int             IDENTIFIER      result1         4 9                     1               9999999
4       int             IDENTIFIER      x               5 7 14                  1               9999999
5       int             IDENTIFIER      y               6 7                     1               9999999
6       int             IDENTIFIER      result2         7                       1               9999999
7       int             IDENTIFIER      Result3         14                      2               9999999
8       int             IDENTIFIER      m               16                      1               9999999
asma@asma-virtualbox:~/Documents/Compiler430-MiniCompiler_Project/Task2- Symbol Table$ ./a.out < input0.cpp
```

Task 3 - Syntax Error generated by parser along with token and symbol table

Input file: with error

input.cpp | lexical.l | lexical.y

```
1  int main() {
2          a = 77;
3          b = 33;
4          result1 = a+ b+ 11;
5          while(count1)
6          {
7                  count--;
8          }
9          if(count=0)
10         {
11                 count = count+2;
12         }
13         else{
14                 count=0;
15         }
16 }
```

Output result

```
asma@asma-virtualbox:~/Documents/Compiler430-MiniCompiler_Project/Task3- Syntax Analysis$ ./a.out < input.cpp
decl:int
main
id:a
assignop:=
num:77
id:b
assignop:=
num:33
id:result1
assignop:=
id:a
id:b
num:11
while
id:count1
Line no: 5
 The error is: syntax error, unexpected ')', expecting comparisionop
id:count
unary:--
if
id:count
assignop:=
num:0
id:count
assignop:=
id:count
num:2
id:else
id:count
assignop:=
num:0
Line no: 16
 The error is: syntax error, unexpected '}', expecting end of file
---------------

Symbol Table
Symbol          Scope           dtype           Value
a               1               int             0
asma@asma-virtualbox:~/Documents/Compiler430-MiniCompiler_Project/Task3- Syntax asma@asma-virtualbox:~/Documents/Compiler430-MiniCom
```

Task 4- Parser/ Abstract Syntax Tree:

Input

input1.cpp | tree.y | tree.l | input0.cpp | input.cpp

```
1  int main(){
2
3          int a, b;
4          a = 77;
5          b = 33;
6          int result1 = a+ b+ 11;
7          if(a>=b)
8          {
9                  result1= 0;
10                 a= a+b;
11         }
12 }
```

Abstract Syntax Tree

```
asma@asma-virtualbox:~/Documents/Compiler430-MiniCompiler_Project/Task4- Parser Tree$ gcc lex.yy.c y.tab.c
asma@asma-virtualbox:~/Documents/Compiler430-MiniCompiler_Project/Task4- Parser Tree$ ./a.out < input1.cpp


SYMBOL TABLE

   SYMBOL              NAME      TYPE        DIMENTION    LINE OF CODE        VALUE
 identifier               a       int          1              3             110
 identifier               b       int          1              3             33
 identifier         result1       int          1              6             0



Abstract Syntax Tree

                                                                                main

                                                                          assign

                                                                    assign              if

                                                              assign              =             >=              assign


                                                         assign              =             result1             +
   a             b             =             =

                                                  Dc b             =             b             33
  +             11                               result1             0             a             +
```

Task 5 - Intermediate Code Generation:

Input file C++ Code

```
asma@asma-virtualbox:~/Documents/Compiler430-MiniCompiler_Project/Task5- Interme
diate Code Generation$ cat input0.cpp
#include<iostream>
using namespace std;
int main(){
a = 77;
b = 33;
result1 = a+b+ 11;
x = 60;
y = 40;
result2 = x * y * 2;
if(a>=b){
        result1= 0;
        a= a+b;
}
else
{
        a = a + 1;
        Result3 = x*10;
}
m=9;
}
asma@asma-virtualbox:~/Documents/Compiler430-MiniCompiler_Project/Task5- Interme
diate Code Generation$
```

Output of TAC

```
asma@asma-virtualbox:~/Documents/Compiler43
a = 77
b = 33
t0 = a + b
t1 = t0 + 11
result1 = t1
x = t1
y = t1
t2 = x * y
t3 = t2 * 2
result2 = t3
t4 = a >= b
ifFalse t4 goto L0
result1 = t4
t5 = a + b
a = t5
goto L1
L0:
t6 = a + 1
a = t6
t7 = x * 10
Result3 = t7
L1:
```

Quadruple format

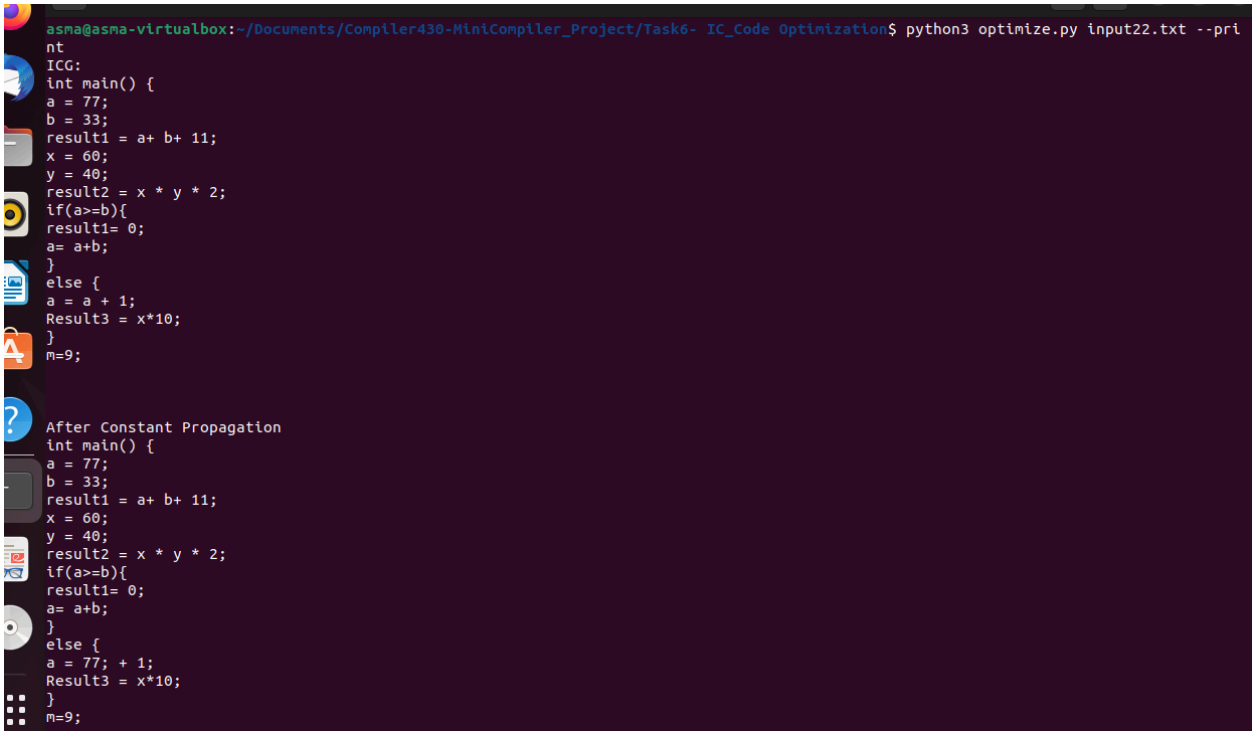
```
      Op            Arg1          Arg2          Res
--------------------------------------------------------------------
      =             77                          a
      =             33                          b
      +             a             b             t0
      +             t0            11            t1
      =             t1                          result1
      =             t1                          x
      =             t1                          y
      *             x             y             t2
      *             t2            2             t3
      =             t3                          result2
      >=            a             b             t4
      ifFalse       t4                          L0
      =             t4                          result1
      +             a             b             t5
      =             t5                          a
      goto                                      L0
      Label                                     L0
      +             a             1             t6
      =             t6                          a
      *             x             10            t7
      =             t7                          Result3
      Label                                     L1
sma@asma-virtualbox:~/Documents/Compiler430-MiniCompiler_Project/Task5- Intermediate Code Generation$
```

Task 6 - Code Optimization(Constant Propagation,Constant folding and Dead Code Elimination):

Input before optimization

```
int main() {
a = 77;
b = 33;
result1 = a+ b+ 11;
x = 60;
y = 40;
result2 = x * y * 2;
if(a>=b){
        result1= 0;
        a= a+b;
}
else {
        a = a + 1;
        Result3 = x*10;
}
m=9;
```

Output after optimization



```
asma@asma-virtualbox:~/Documents/Compiler430-MiniCompiler_Project/Task6- IC_Code Optimization$ python3 optimize.py input22.txt --pri
nt
ICG:
int main() {
a = 77;
b = 33;
result1 = a+ b+ 11;
x = 60;
y = 40;
result2 = x * y * 2;
if(a>=b){
result1= 0;
a= a+b;
}
else {
a = a + 1;
Result3 = x*10;
}
m=9;


After Constant Propagation
int main() {
a = 77;
b = 33;
result1 = a+ b+ 11;
x = 60;
y = 40;
result2 = x * y * 2;
if(a>=b){
result1= 0;
a= a+b;
}
else {
a = 77; + 1;
Result3 = x*10;
}
m=9;
```

```
After Constant Folding:
int main() {
a = 77;
b = 33;
result1 = a+ b+ 11;
x = 60;
y = 40;
result2 = x * y * 2;
if(a>=b){
result1= 0;
a= a+b;
}
else {
a = 77; + 1;
Result3 = x*10;
}
m=9;


After Dead Code Elimination:
int main() {
a = 77;
b = 33;
result1 = a+ b+ 11;
x = 60;
y = 40;
result2 = x * y * 2;
if(a>=b){
result1= 0;
a= a+b;
}
else {
a = 77; + 1;
Result3 = x*10;
}
asma@asma-virtualbox:~/Documents/Compiler430-MiniCompiler_Project/Task6- IC_Code Optimization$
```

Task 7 - Assembly Code Generation:

Input file

```
i = 2
t0 = i > 1
ifFalse t0 goto L0
t1 = i + 1
i = t1
goto L1
L0:
t2 = i - 1
i = t2
L1:
t3 = i + 3
i = t3
i = t3
L2:
t4 = i < 10
ifFalse t4 goto L3
t5 = i + 2
a = t5
t6 = i + 1
i = t6
goto L2
```

Output

```
asma@asma-virtualbox:~/Documents/
.text
MOV R0,=i
MOV R1,[R0]
CMP R1,#1
BLE L0
MOV R2,=i
MOV R3,[R2]
MOV R4,=t1
MOV R5,[R4]
ADD R5,#3,R1
STR R5, [R4]
MOV R6,=i
MOV R7,[R6]
MOV R8,#t1
STR R8, [R6]
B L1
L0:
MOV R9,=i
MOV R10,[R9]
MOV R11,=t2
MOV R12,[R11]
SUBS R12,#10,R1
STR R12, [R11]
MOV R0,=i
MOV R1,[R0]
MOV R2,#t2
STR R2, [R0]
L1:
MOV R3,=i
MOV R4,[R3]
MOV R5,=t3
MOV R6,[R5]
ADD R6,#4,R3
STR R6, [R5]
MOV R7,=i
```

```
MOV R4,[R3]
MOV R5,=t3
MOV R6,[R5]
ADD R6,#4,R3
STR R6, [R5]
MOV R7,=i
MOV R8,[R7]
MOV R9,#t3
STR R9, [R7]
MOV R10,=i
MOV R11,[R10]
MOV R12,#t3
STR R12, [R10]
L2:
MOV R0,=i
MOV R1,[R0]
CMP R1,#10
BGE L3
MOV R2,=i
MOV R3,[R2]
MOV R4,=t5
MOV R5,[R4]
ADD R5,#3,R2
STR R5, [R4]
MOV R6,=i
MOV R7,[R6]
MOV R8,=t6
MOV R9,[R8]
ADD R9,#7,R1
STR R9, [R8]
MOV R10,=i
MOV R11,[R10]
MOV R12,#t6
STR R12, [R10]
B L2
SWI 0x011
.DATA
i: .WORD 2
asma@asma-virtualbox:~/Docum
```