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:

Arpit Singh	**01FB16ECS073**
Ashish Sanu	**01FB16ECS075**
Bilal Shakil	**01FB16ECS091**

Under the guidance of

Madhura V
Course Lecturer
PES University, Bengaluru

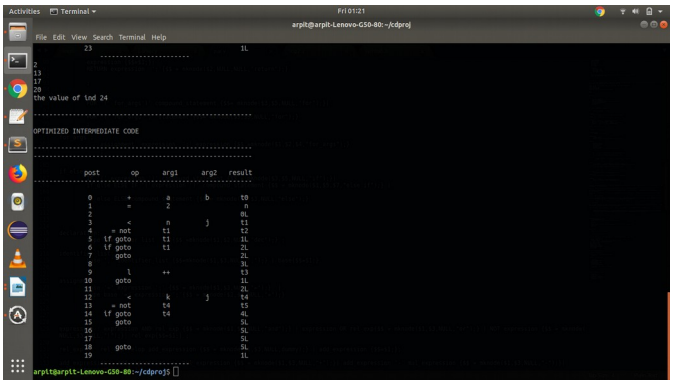
January – May 2019

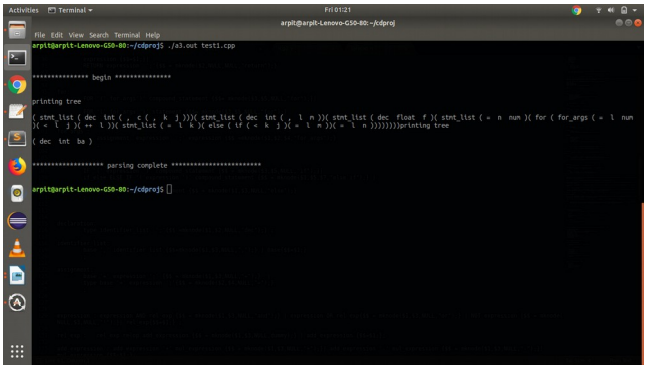
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

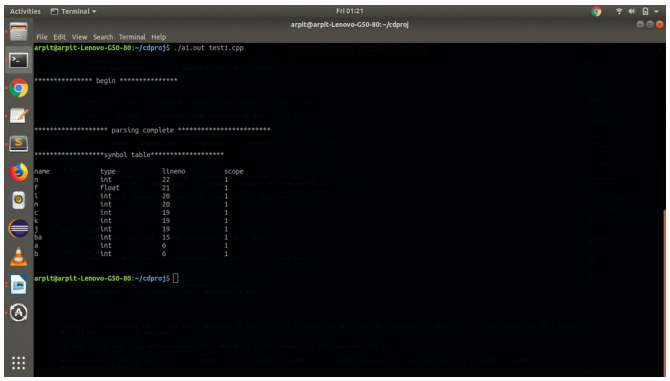
TABLE OF CONTENTS

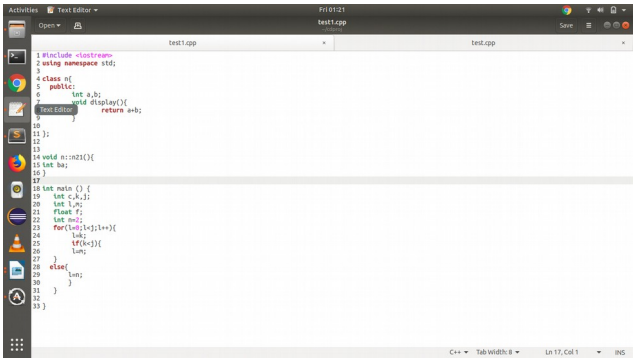
INTRODUCTION :

- We have implemented a simple c++ compiler using LEX and YACC.
- The compiler performs Lexical Analysis, Syntax Analysis , Intermediate Code Generation (Three Address Code),Code Optimization and generates Abstract Syntax tree.
- The identifier information is stored in a symbol table.
- The tools used are Flex and Bison.
- **About C++ :**
  - C++ is a middle-level programming language developed by Bjarne Stroustrup starting in 1979 at Bell Labs.
  - C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming.
  - C++ is a superset of C, and that virtually any legal C program is a legal C++ program.









ARCHITECTURE OF THE LANGUAGE :

- C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming.

FEATURES OF THE COMPILER :

1. Syntax includes detecting class , compound statements , functions , header files , namespace , for loop, if else conditional statements , variable declaration , assignment , array declaration, function call. If any of the above mentioned syntax rules is not satisfied the compiler throws a syntax error with the line number.

2. Semantics include variable type checking , undeclared identifiers , type mismatch.

3. Compiler prints the abstract Syntax tree for the input code. Syntax tree for if else statements, for loop and expressions will be printed .

4. Intermediate code Generation for expressions , if else statements, for loops.

5. Copy Propogation and Constant propogation Optimization.

LITERATURE SURVEY :

references for the project :

- *class slide*
- *geeks for geeks(for concepts)*
- *tutorialspoint (lex and yacc)*
- lex and yacc , 2nd Edition by Tony Mason, Doug Brown, John Levine
- Lab Conduction

Grammar :

```
start:
    INCLUDE start | function start |  class start | nam start | declaration start |
    ;

nam : USING NAMESPACE obj ';' ;

class :
                CLASS ID '{' classbdy '}'  classobj ';'
                 ;

classbdy :
                ACCESS ':' classbdy | declaration classbdy | function classbdy | ;

classobj : ID ',' classobj | ID | ;

function:
          type ID '(' arg_list ')'  compound_statement  |
          type ID '(' ')'  compound_statement |
          type ID '(' type_list ')' ';' |
          type ID ':'':' ID '(' arg_list ')'  compound_statement |
          type ID ':'':' ID '(' ')'  compound_statement
          ;

arg_list : arg ',' arg_list | arg ;

type_list : type ',' type_list | type ;

arg:
        type ID |
        ;

compound_statement:
          '{' statement_list '}' |
          '{' '}'
        ;
```

```
statement_list:
                statement |
                statement statement_list
                ;
statement:
                declaration |
                assignment |
                array |
                for |
                if_else|
                function_call  |
                RETURN expression ';'

                ;
declaration:
                type identifier_list ';' ;

identifier_list:
                ID ',' identifier_list | ID
                ;
assignment:
                ID '=' expression ';' |
                type ID '=' expression ';'
                ;

for:
                FOR '(' assignment  expression  ';' expression ')' compound_statement |
                FOR '(' assignment  expression  ';' expression ')' statement |
                FOR '(' ';' expression ';' ')' compound_statement |
                FOR '(' ';' ';' ')' compound_statement |
                FOR '(' assignment ';' ')' compound_statement |
                FOR '(' assignment expression ';' ')' compound_statement |
                FOR '(' ';' expression ';' expression ')' compound_statement |
                FOR '(' assignment ';'expression ')' compound_statement
                ;

if_else:
                IF '(' expression ')' compound_statement |
                if_else ELSE IF '('expression ')' compound_statement |
                if_else ELSE compound_statement
                ;
```

```
expression : expression AND rel_exp | expression OR rel_exp | NOT rel_exp |
rel_exp ;

rel_exp : rel_exp relop add_expression | add_expression ;

add_expression : add_expression '+' mul_expression | add_expression '-'
mul_expression |                                    mul_expression ;

mul_expression : mul_expression '*' cast_exp |  mul_expression '/' cast_exp | cast_exp
;

cast_exp : unary_exp | '(' type ')' cast_exp ;

unary_exp : exp | INCR exp | DECR exp | exp INCR | exp DECR | unary_op exp ;

unary_op : '-' | '+' | '&' | '!' ;

exp : base | exp '(' ')' | exp '(' identifier_list ')' ;

base : ID | NUM | FNUM | STRING | '(' expression')' ;

relop : LE | GE | GT | LT | EE  | NE ;

array:
                type ID '[' NUM ']' ';'   |
                type ID '[' NUM ']' '=' STRING ';'   |
                ID '[' NUM ']' '=' STRING ';' |
                type ID '[' NUM ']' '=' NUM ';'  |
                ID '[' NUM ']' '=' NUM ';'
                ;
function_call:
                ID '(' identifier_list ')' ';'
                ID '(' ')' ';'
                ;
obj :    STD;

type:
                INT | VOID | CHAR | FLOAT  | DOUBLE | BOOL
                ;
```

DESIGN STRATEGY :

- **SYMBOL TABLE CREATION -**
  - *Determining the structure of the symbol table.*
  - *What information needs to be stored.*
  - *Decide the data structure to be used.*
- **ABSTRACT SYNTAX TREE -**
  - *Syntax tree for expressions, if else conditional statements , for loops .*
  - *SDD for AST.*
  - *Decide the data structure to be used.*
- **INTERMEDIATE CODE GENERATION -**
  - *Generate Three address code for expressions , if else statements , for loop.*
  - *Implement functions.*
  - *Use records to store Intermediate Code.*
- **CODE OPTIMIZATION -**
  - *Select any optimization technique.*
  - *Copy propogation and Constant Propogation.*
  - *Implement functions.*

- **ERROR HANDLING -**
  - *Detect any unmatched tokens.*
  - *Throw syntax error if the grammar is not satisfied.*
  - *Check for type mismatch, undeclared variables , Scope.*

IMPLENTATION DETAILS :

- **SYMBOL TABLE CREATION -**
  - *Symbol table fields include Identifier name, type, line no, scope.*
  - *Data Structure used to implement Symbol table - “LINKED LIST”*
- **ABSTRACT SYNTAX TREE -**
  - *Data Structure used to implement AST - “Tree”*
  - *SDD for AST.*
  - *Mknode function used to create nodes for each rule.*
  - *Print the tree in post order.*
- **INTERMEDIATE CODE GENERATION -**
  - *functions used to generate code , create labels.*
  - *Quadruple records used to store ICG.*
  - *Structure contains result ,arg1 ,arg2 ,op*
- **CODE OPTIMIZATION -**
  - *Implements a function to apply copy propogation and contsant propogation optimization technique.*

- **ERROR HANDLING -**
  - *specify the regex to detect tokens.*
  - *Use symbol table to store identifier information and use it for detecting type mismatches and undeclared variables.*
  - *BNF grammar to detect Syntax errors and prints the line number.*

RESULTS AND CONCLUSIONS:

- A Simple Mini c++ compiler that features Symbol Table, Intermediate Code Genration, Abstract Syntax tree and Code Optimization.
- Detects Syntax errors and prints the lines.
- Stores identifier information in Symbol Table.
- Intermediate Code Generation for expressions, if else statements, for loops.
- Copy Propogation and constant propogation Optimization.

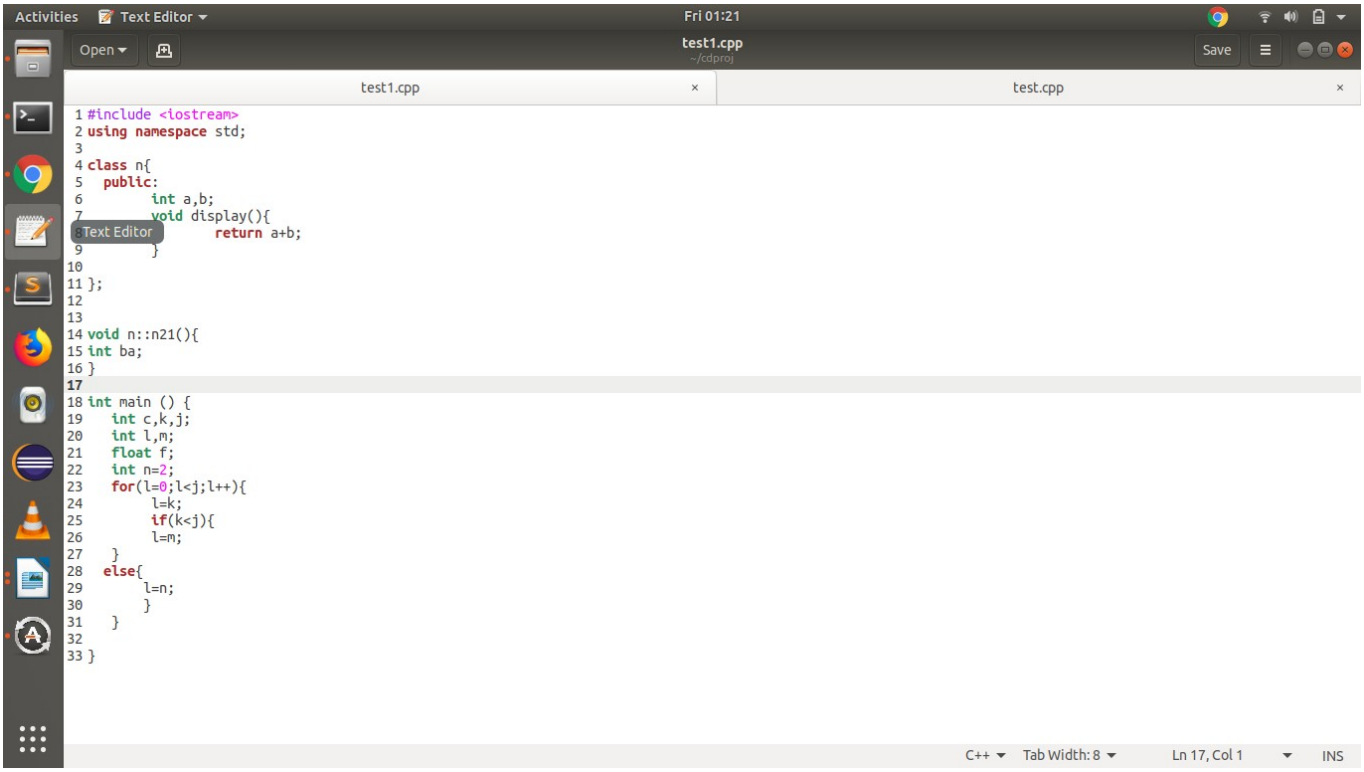
Shotcomings -

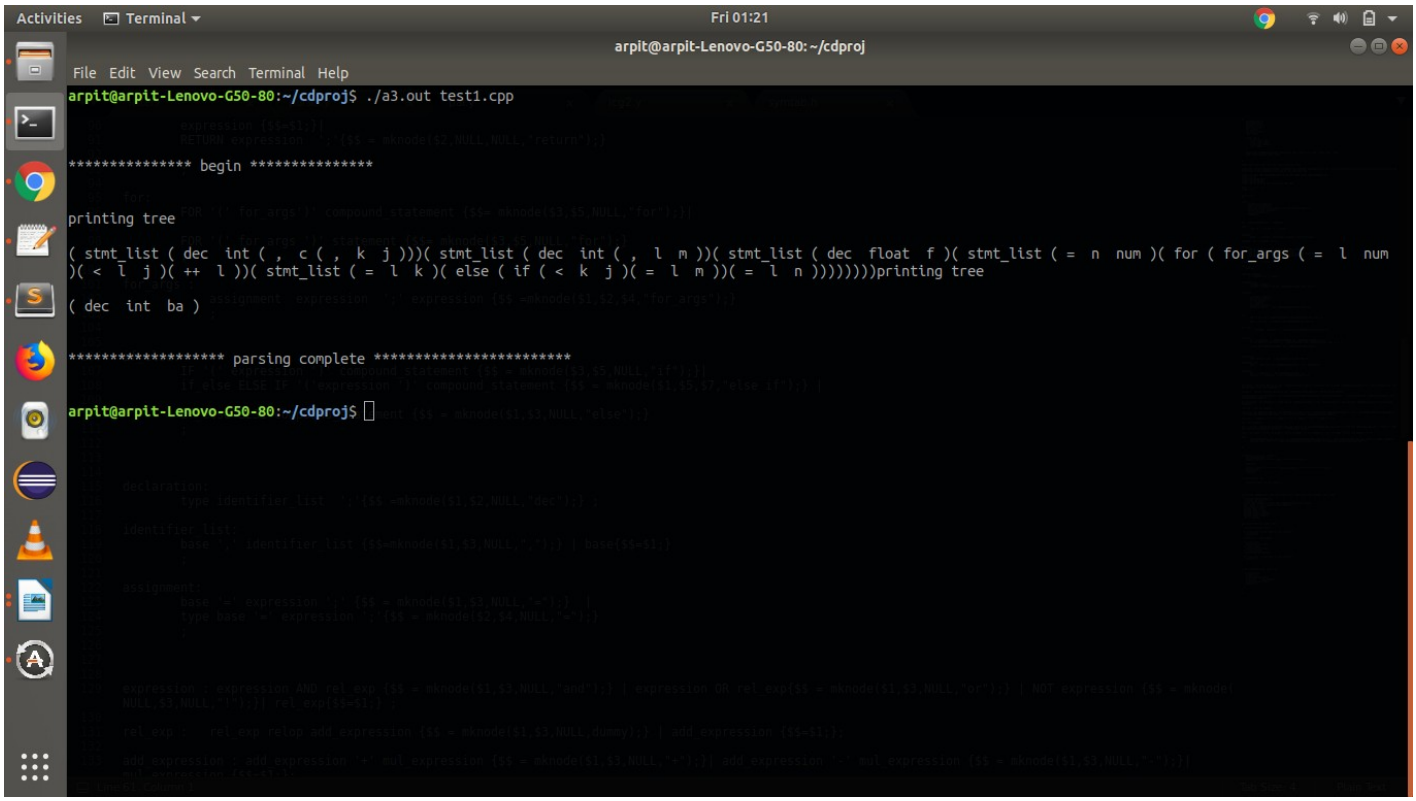
1. *Grammar , AST , ICG and optimization implemented only for expressions , if else statements and for loops.*
2. *Only copy propogation and constant propogation optimization techniques applied.*

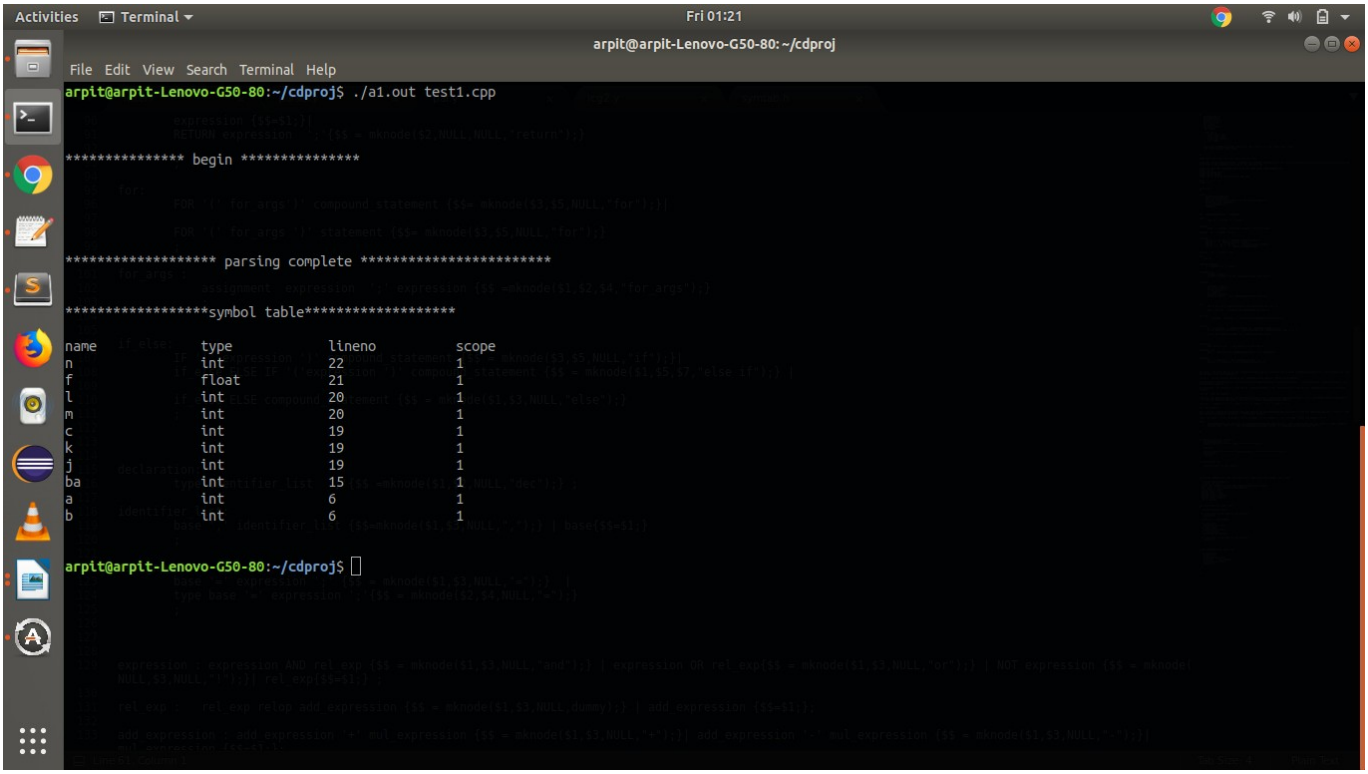
FUTURE ENHANCEMENTS -

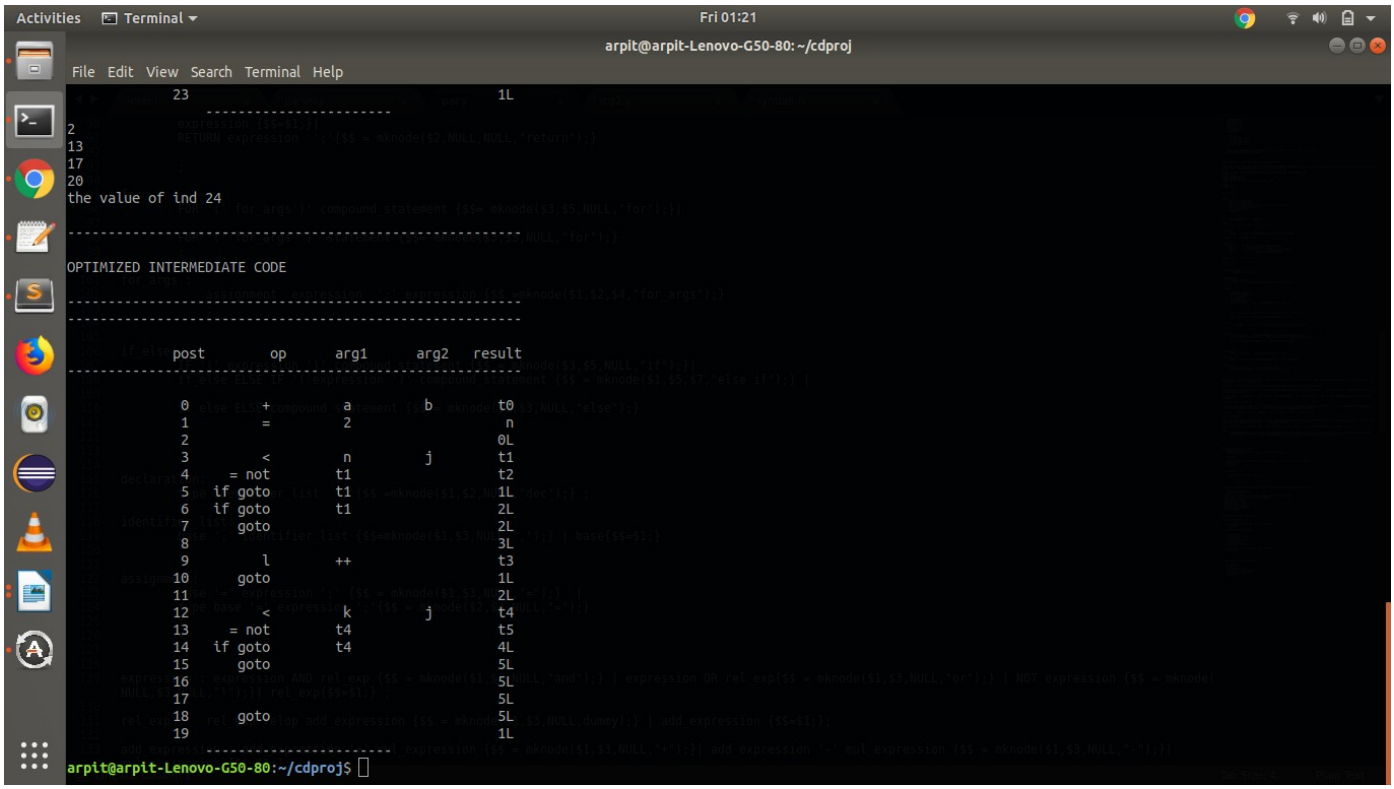
1. *Grammar , AST , ICG and optimization implementation for while loops, switch statements , control structures etc.*
2. *Add more optimizations to the code.*

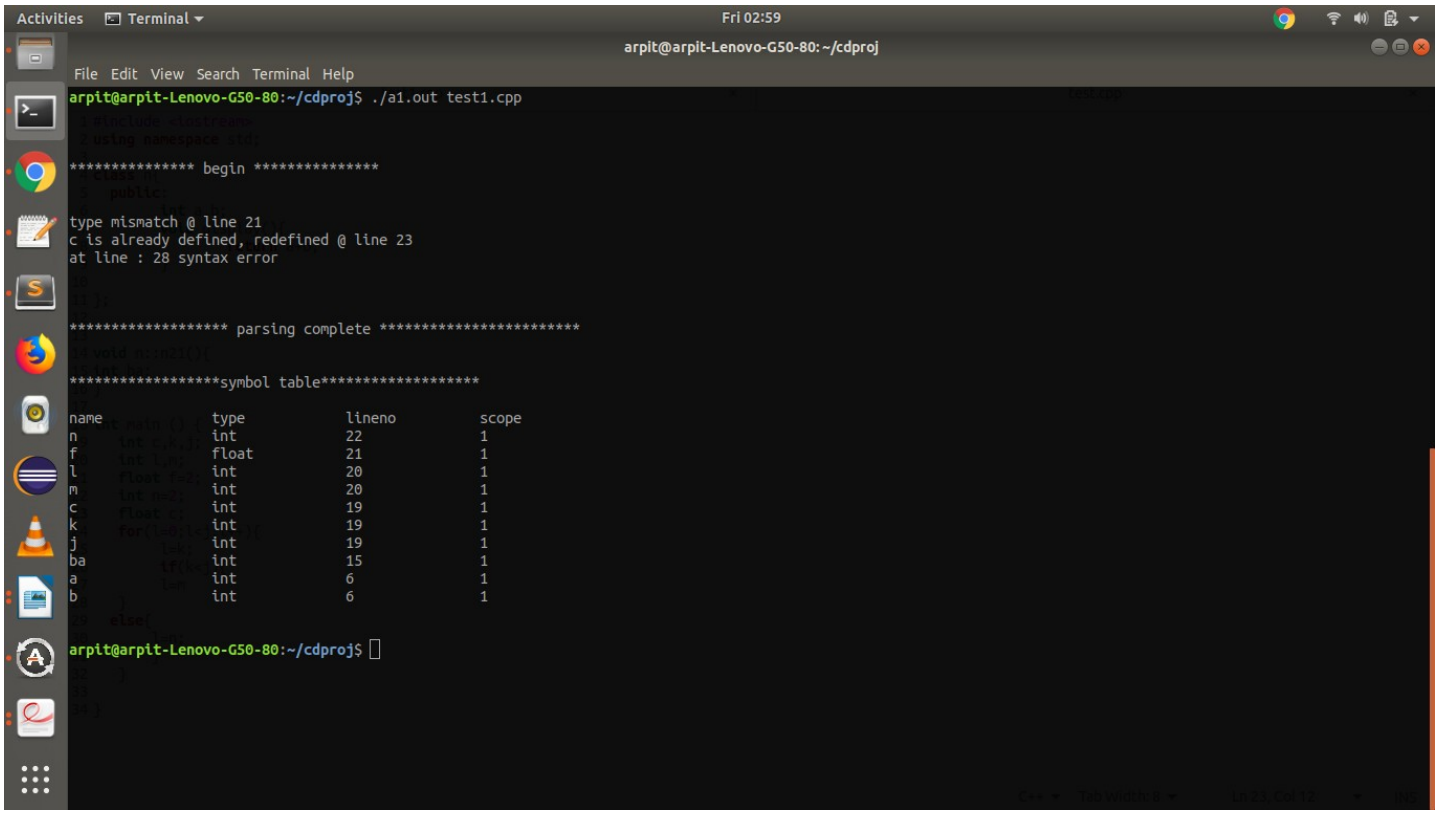
SNAPSHOTS :


Activities
Text Editor
Fri 01:21
Open
test1.cpp
~/cdproj
Save
test1.cpp
test.cpp
#include <iostream>
using namespace std;
class n{
public:
int a,b;
void display(){
return a+b;
}
};
void n::n21(){
int ba;
}
int main () {
int c,k,j;
int l,m;
float f;
int n=2;
for(l=0;l<j;l++){
l=k;
if(k<j){
l=m;
}
else{
l=n;
}
}
}
C++
Tab Width: 8
Ln 17, Col 1
INS


Activities
Terminal
Fri 01:21
arpit@arpit-Lenovo-G50-80: ~/cdproj
File Edit View Search Terminal Help
arpit@arpit-Lenovo-G50-80:~/cdproj$ ./a3.out test1.cpp
********** begin **********
printing tree
(stmt_list (dec int (, c (, k j))(stmt_list (dec int (, l m))(stmt_list (dec float f)(stmt_list (= n num)(for (for_args (= l num)(< l j)(++ l))(stmt_list (= l k)(else (if (< k j)(= l m))(= l n)))))))printing tree
(dec int ba)
********** parsing complete **********
arpit@arpit-Lenovo-G50-80:~/cdproj$


Activities
Terminal
Fri 01:21
arpit@arpit-Lenovo-G50-80: ~/cdproj
File Edit View Search Terminal Help
arpit@arpit-Lenovo-G50-80:~/cdproj$ ./a1.out test1.cpp
********** begin **********
****************** parsing complete ************************
*****************symbol table******************
name type lineno scope
n int 22 1
f float 21 1
l int 20 1
m int 20 1
c int 19 1
k int 19 1
j int 19 1
ba int 15 1
a int 6 1
b int 6 1
arpit@arpit-Lenovo-G50-80:~/cdproj$


Activities
Terminal
Fri 01:21
arpit@arpit-Lenovo-G50-80: ~/cdproj
File Edit View Search Terminal Help
23 1L
2
13
17
20
the value of ind 24
OPTIMIZED INTERMEDIATE CODE
post op arg1 arg2 result
0 + a b t0
1 = 2 n
2 0L
3 < n j t1
4 = not t1 t2
5 if goto t1 1L
6 if goto t1 2L
7 goto 2L
8 3L
9 l ++ t3
10 goto 1L
11 2L
12 < k j t4
13 = not t4 t5
14 if goto t4 4L
15 goto 5L
16 5L
17 5L
18 goto 5L
19 1L
arpit@arpit-Lenovo-G50-80:~/cdproj$



COMPILER DESIGNER PROJECT

TEAM MEMBERS :

NAME	USN
ARPIT SINGH	01FB16ECS073
ASHISH SANU	01FB16ECS075
BILAL SHAKIL	01FB16ECS091