

***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**  **Ashish Sanu**  **Bilal Shakil** | **01FB16ECS073**  **01FB16ECS075**  **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)

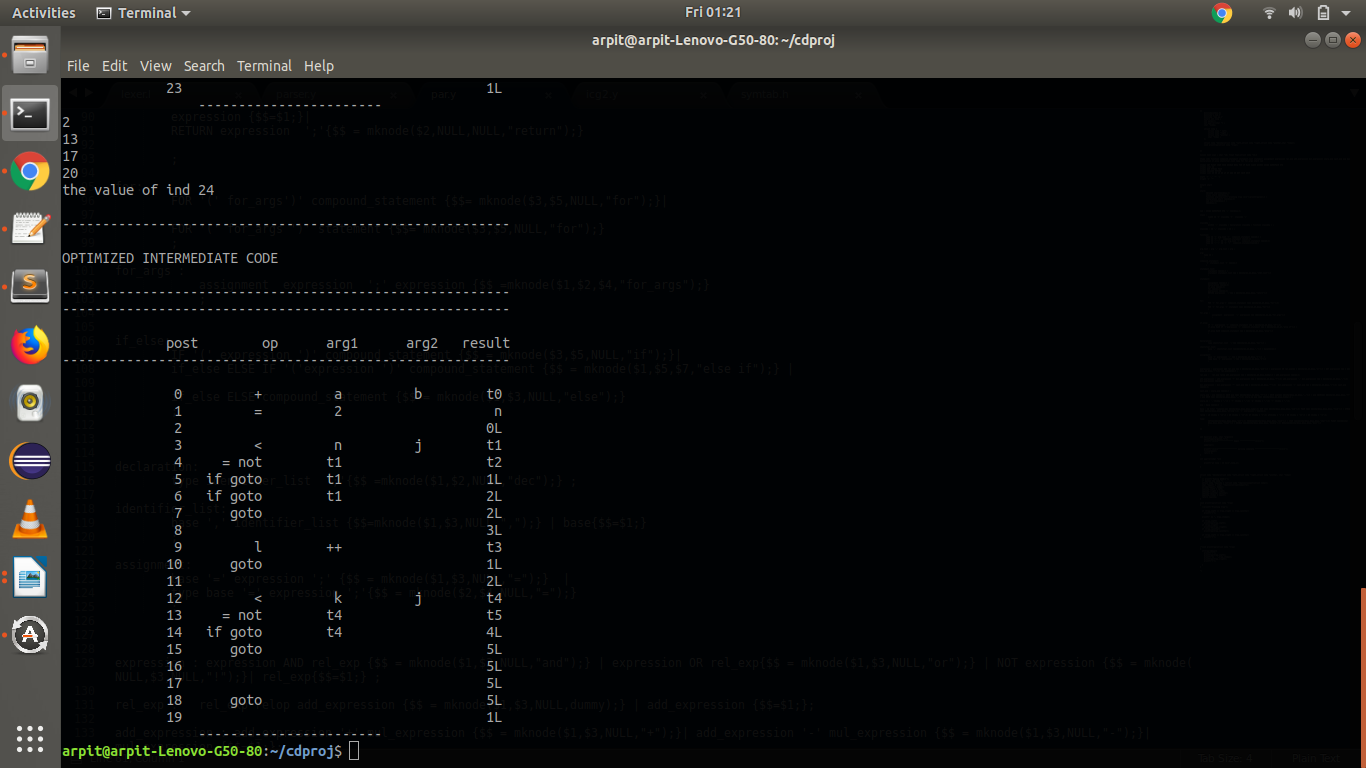
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

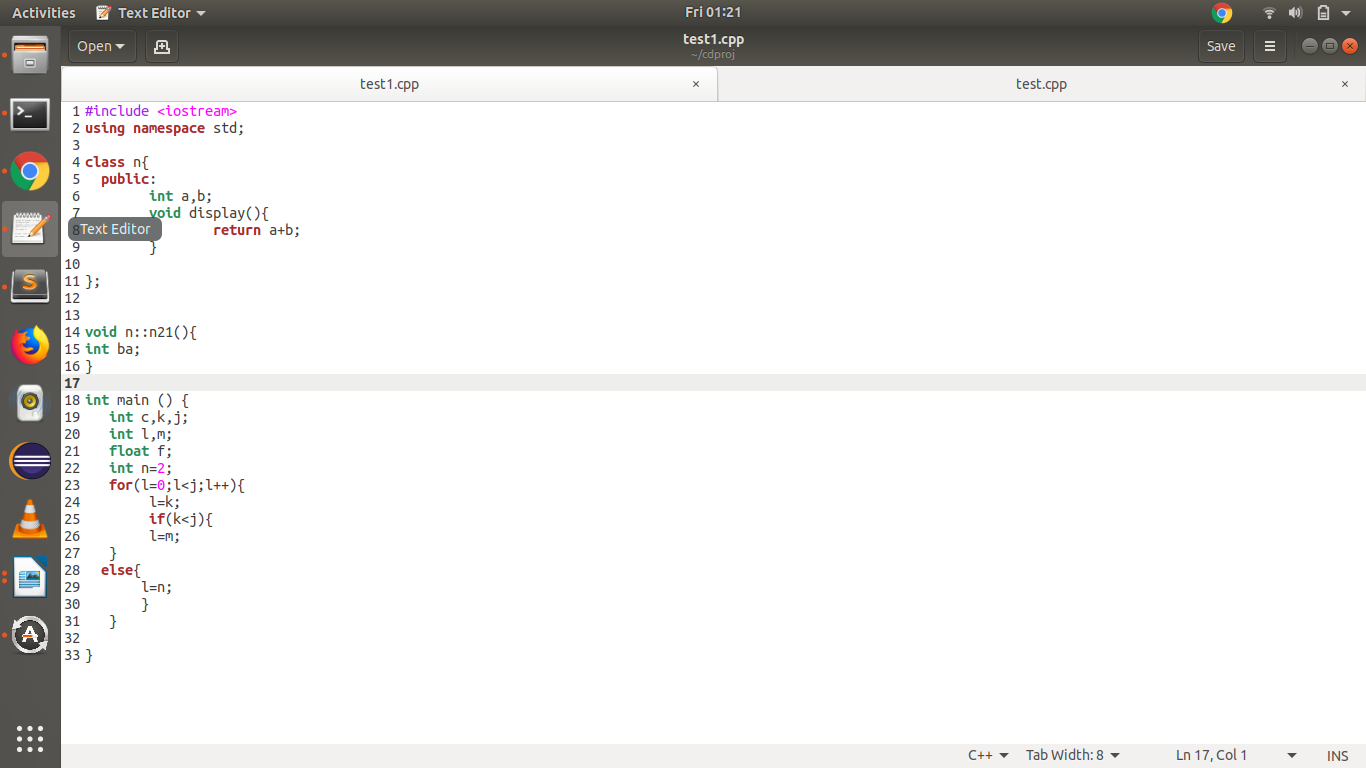
**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Chapter No.** | **Title** | **Page No.** |
|  | **INTRODUCTION** | **01** |
|  | **ARCHITECTURE OF LANGUAGE:** | **02** |
|  | **LITERATURE SURVEY** | **03** |
|  | **CONTEXT FREE GRAMMAR** |  |
|  | **DESIGN STRATEGY**   * **SYMBOL TABLE CREATION** * **ABSTRACT SYNTAX TREE** * **INTERMEDIATE CODE GENERATION** * **CODE OPTIMIZATION** * **ERROR HANDLING** |  |
|  | **IMPLEMENTATION DETAILS (TOOL AND DATA STRUCTURES USED in order to implement the following):**   * **SYMBOL TABLE CREATION** * **ABSTRACT SYNTAX TREE** * **INTERMEDIATE CODE GENERATION** * **CODE OPTIMIZATION** * **ERROR HANDLING** |  |
|  | **RESULTS AND possible shortcomings of your Mini-Compiler** |  |
|  | **SNAPSHOTS** |  |
|  | **CONCLUSIONS** |  |
|  | **FURTHER ENHANCEMENTS** |  |
| **REFERENCES/BIBLIOGRAPHY** | |  |

**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:**

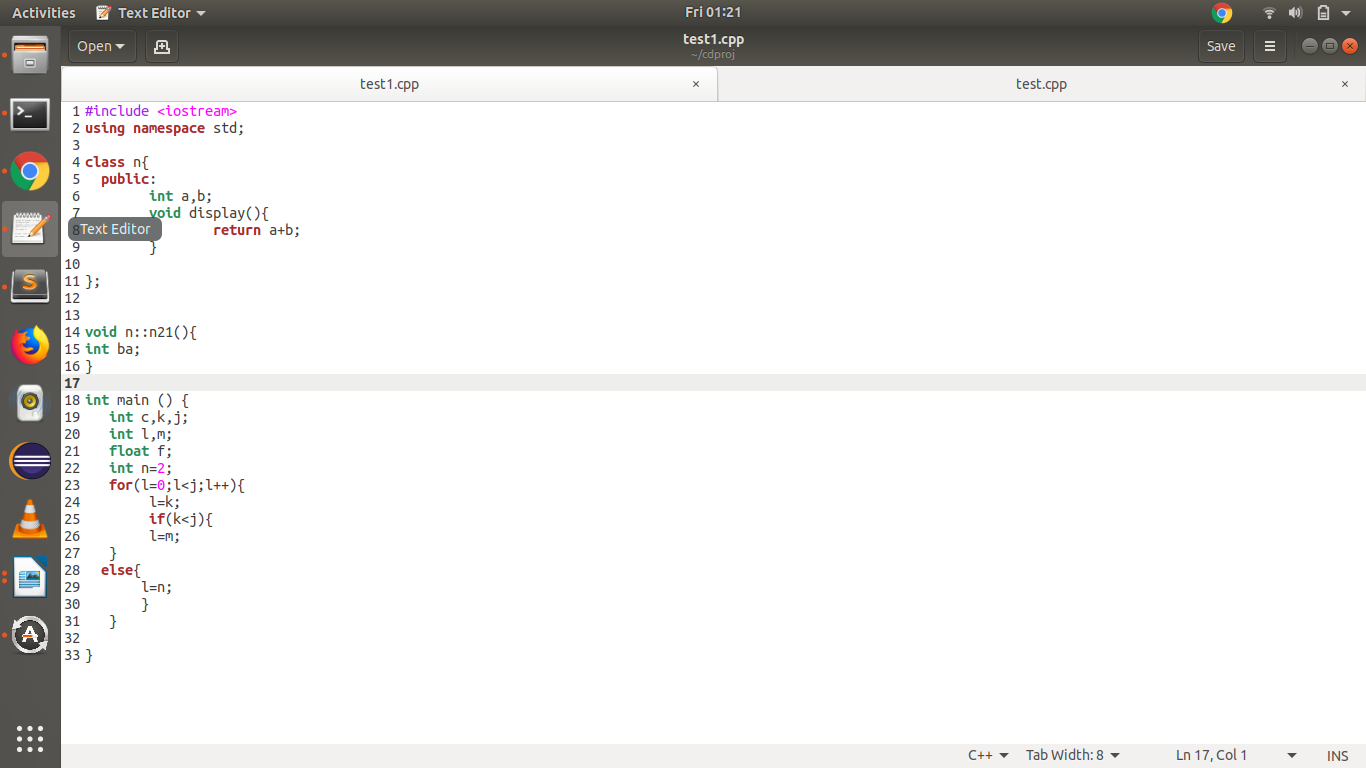
* A Simple Mini c++ compiler that features Symbol Table, Intermediate Code Genration, Abstract Syntax tree and Code Optimization.
* Detects Syntax errors and prints the lines.
* Stores identifier information in Symbol Table.
* Intermediate Code Generation for expressions, if else statements, for loops.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

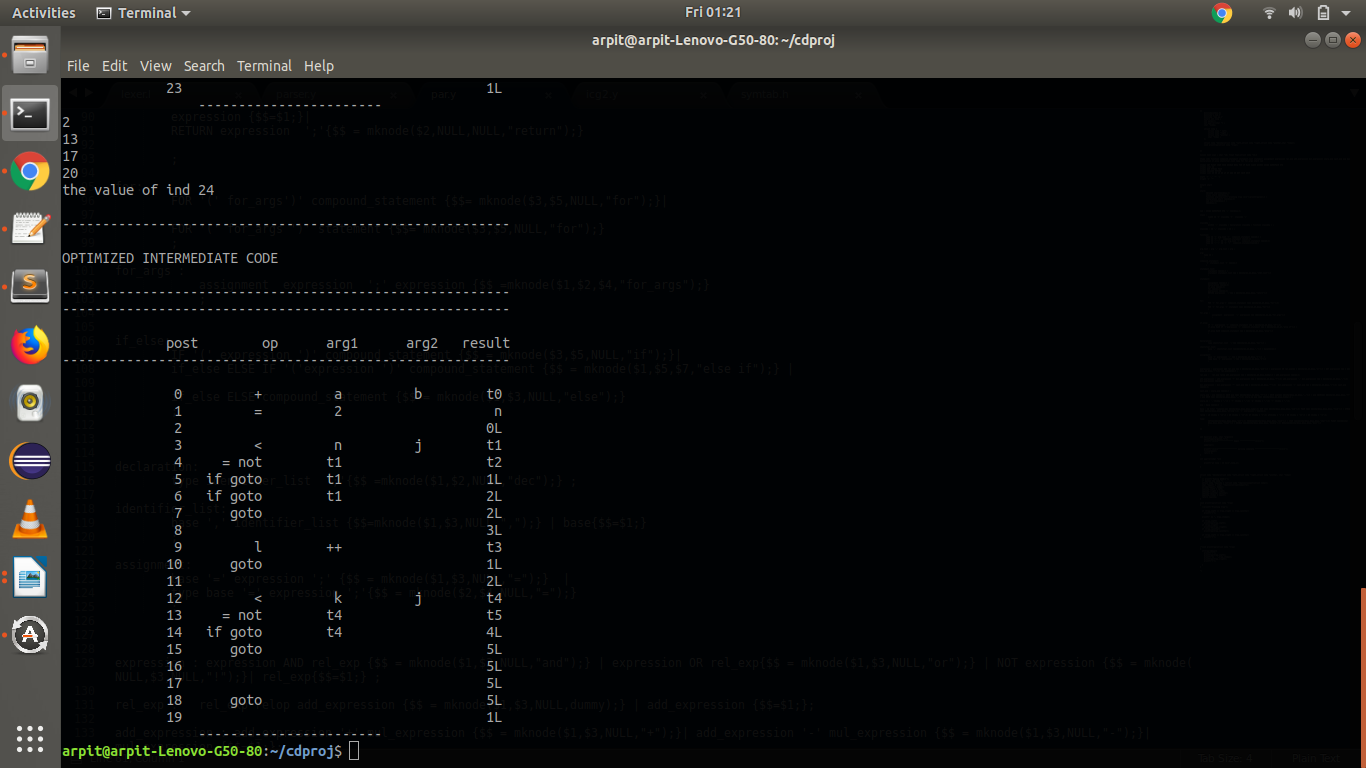
***Shotcomings -***

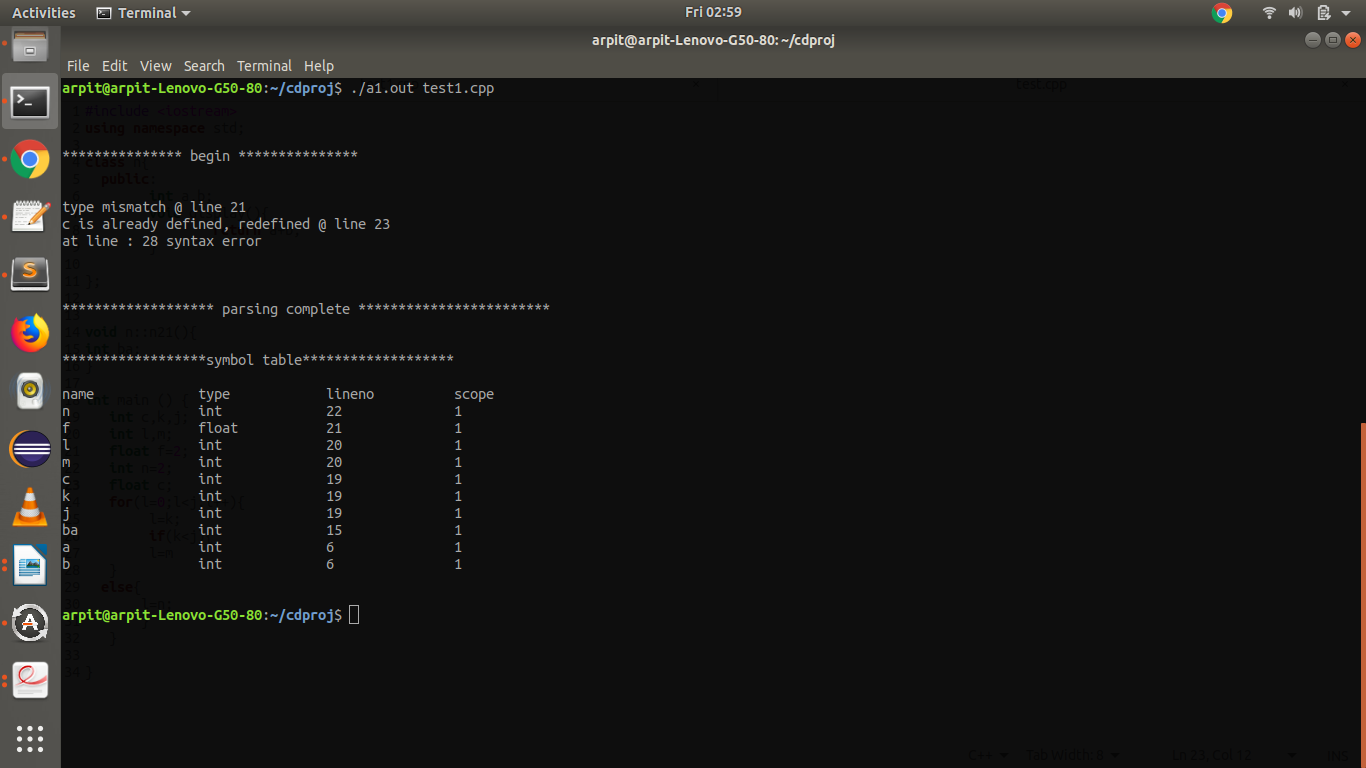
1. *Grammar , AST , ICG and optimization implemented only for expressions , if else statements and for loops.*
2. *Only copy propogation and constant propogation optimization techniques applied.*

***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 :**





**COMPILER DESIGNER PROJECT**

**TEAM MEMBERS :**

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