

# University of Information Technology & Sciences

Department of  
Computer Science and Engineering



## Project Report

Course Title: Compiler Lab  
Course Code: CSE-352

### Submitted By

Name : Md. Tamim Iqbal  
Id : 0432220005101023  
Batch : 52

### Submitted To

Md. Tasnin Tanvir  
Lecturer, Department of CSE

## Problem Description:

This lab project demonstrates the use of C++ functions to perform a wide range of basic arithmetic, logical, bitwise, and comparison operations. The project is modularly designed with a clear separation between the main program logic (code.cpp) and the function implementations (function.h). It acts as a mini calculator and logic simulator for integer and Boolean operations using hardcoded test values.

## Introduction:

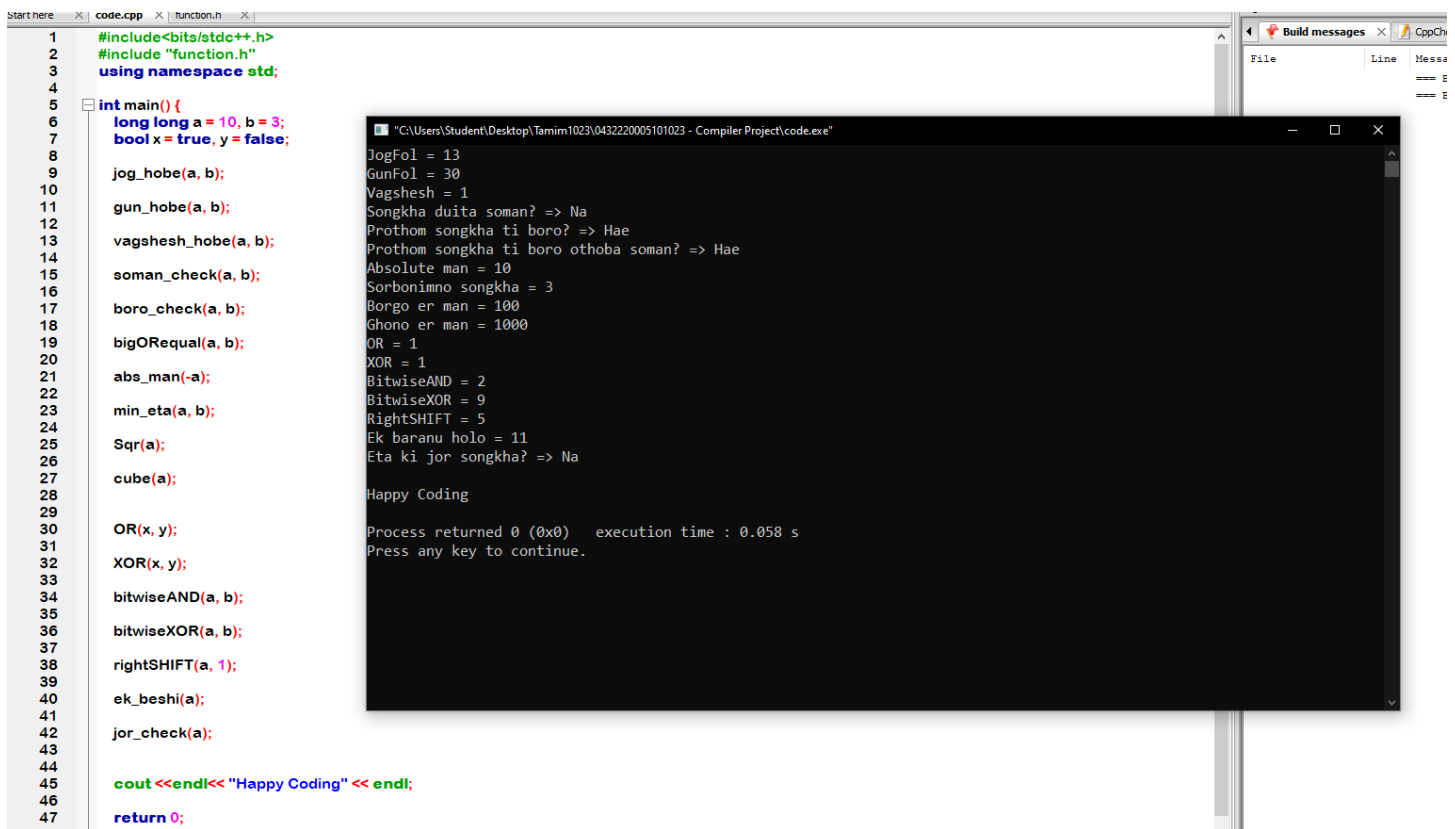
In computer programming, mastering arithmetic and logical operations is crucial, especially in low-level or system-level development such as embedded systems or compiler design. This project aims to reinforce those foundational concepts by implementing a set of functions in C++ that perform mathematical operations (addition, subtraction, multiplication, division, modulus), logical operations (AND, OR, NOT, XOR), comparison checks (equal, greater, smaller), and bit-level operations (bitwise AND, OR, XOR, NOT, shifts). It is developed using standard C++ with an emphasis on simplicity and clarity, ideal for beginners learning the fundamentals of computation and logic.

## Methodology:

The project follows a modular approach and includes the following components:

- Header File (function.h):
  - Contains the definitions of all the operations as separate reusable functions, including:
    - Arithmetic functions: jog\_hobe, biyog\_hobe, gun\_hobe, vag\_hobe, vagshesh\_hobe
    - Comparison functions: soman\_check, boro\_check, choto\_check, bigORequal, smallORequal
    - Utility functions: abs\_man, max\_eta, min\_eta, Sqr, cube
    - Logical functions: AND, OR, NOT, XOR
    - Bitwise functions: bitwiseAND, bitwiseOR, bitwiseXOR, bitwiseNOT, leftSHIFT, rightSHIFT
    - Increment/Decrement: ek\_beshi, ek\_kom
    - Parity check: jor\_check, bijor\_check
- Main File (code.cpp):
  - This file contains the main() function, where fixed values (a = 10, b = 3, x = true, y = false) are used to call and test each of the functions in a sequential manner. The outputs of each function call are printed to the console to validate correctness.
- Tools Used:
  - Language: C++
  - Libraries: Standard C++ library (<bits/stdc++.h>)
  - Compilation & Execution: Any standard C++ compiler (e.g., GCC, g++)

## Result:



The image shows a C++ IDE with two windows. The left window displays the source code for `code.cpp`, and the right window shows the execution output.

```
1 #include<bits/stdc++.h>
2 #include "function.h"
3 using namespace std;
4
5 int main() {
6     long long a = 10, b = 3;
7     bool x = true, y = false;
8
9     jog_hobe(a, b);
10
11     gun_hobe(a, b);
12
13     vagshesh_hobe(a, b);
14
15     soman_check(a, b);
16
17     boro_check(a, b);
18
19     bigORequal(a, b);
20
21     aba_man(-a);
22
23     min_eta(a, b);
24
25     Sqr(a);
26
27     cube(a);
28
29
30     OR(x, y);
31
32     XOR(x, y);
33
34     bitwiseAND(a, b);
35
36     bitwiseXOR(a, b);
37
38     rightSHIFT(a, 1);
39
40     ek_beshi(a);
41
42     jor_check(a);
43
44
45     cout<<endl<<"Happy Coding"<<endl;
46
47     return 0;
48 }
```

The output window shows the following results:

```
JogFol = 13
GunFol = 30
Vagshesh = 1
Songkha duita soman? => Na
Prothom songkha ti boro? => Hae
Prothom songkha ti boro othoba soman? => Hae
Absolute man = 10
Sorbonimno songkha = 3
Borgo er man = 100
Ghono er man = 1000
OR = 1
XOR = 1
BitwiseAND = 2
BitwiseXOR = 9
RightSHIFT = 5
Ek baranu holo = 11
Eta ki jor songkha? => Na
Happy Coding
Process returned 0 (0x0)   execution time : 0.058 s
Press any key to continue.
```

## Conclusion:

This project successfully demonstrates how basic computational and logical operations can be modularized and reused in C++. It highlights the importance of function decomposition and promotes clean code structure by isolating logic into a header file. While the project uses simple fixed inputs, the structure can easily be extended to support dynamic user input or integration into larger systems. It provides a solid foundation for learning not only arithmetic but also low-level logical and bit manipulation essential for understanding computer architecture and microcontroller programming.