# Department of Computing

# School of Electrical Engineering and Computer Science

**CS-250: Data Structure and Algorithms**

**Class: BSAI-14**

# 

# Lab 1: Introduction to Version Control (Git)

**Date: 09th September, 2025**

**Time: 02:00 pm – 04:50 pm**

# Instructor: Dr. Aimal Tariq

# Lab Engineer: Ms. Areeba Rameen

# Lab 1: Introduction to Version Control (Git)

**Introduction**

Before starting with Git and GitHub, we need a proper coding environment. In this lab, we will:

1. Install **VS Code** – a popular, lightweight IDE.
2. Install **Git** – a version control system for tracking code changes.
3. Configure Git with username and email.
4. Create a **GitHub account** to store projects online and collaborate

**Objectives**

* Learn what Git and GitHub are, and why they are important.
* Install and configure Git.
* Connect a repository with GitHub.
* Understand the workflow of version control within VS Code.

**Tools/Software Requirement**

Visual Studio C++, Git

**Description**

1. **What is Version Control?**

Version control is a **system that records changes to files** (usually code) so you can recall specific versions later.

**Why is Version Control Important?**

* **History Tracking:** Every change is saved with who made it and when.
* **Collaboration:** Multiple developers can work on the same project without overwriting each other’s work.

1. **What is Git?**

Git is a **distributed version control system (VCS)** used to track changes in source code. It allows multiple developers to collaborate, revert mistakes, and maintain a history of changes.

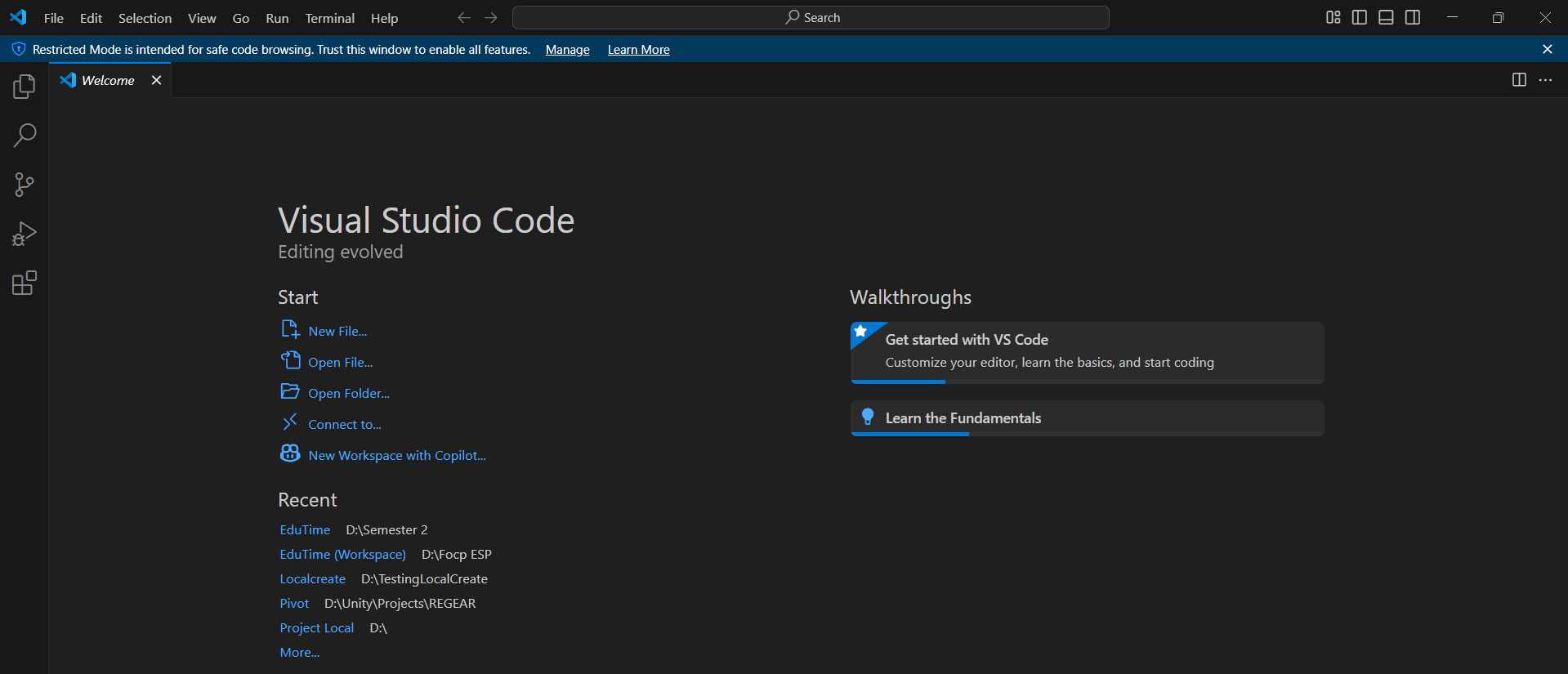
* Keeps a **history of every change** in your project.
* Allows you to **rollback** to older versions.
* Makes **team collaboration** easy.

1. **What is GitHub?**

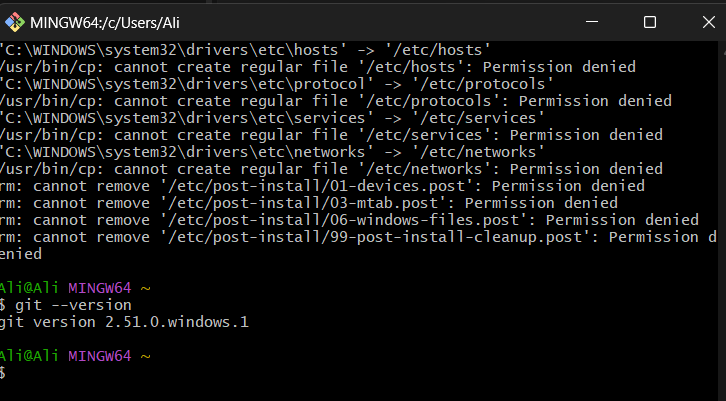
GitHub is an **online hosting platform for Git repositories** where developers can store and share their repositories.

* Git works on your local machine, but **GitHub lets you store and share code in the cloud**.
* It provides additional features:
  + **Collaboration** – Teams can work together, review code, and comment.
  + **Open Source** – Millions of projects are publicly available to learn and contribute to.
  + **Issue Tracking** – Manage bugs, tasks, and feature requests.

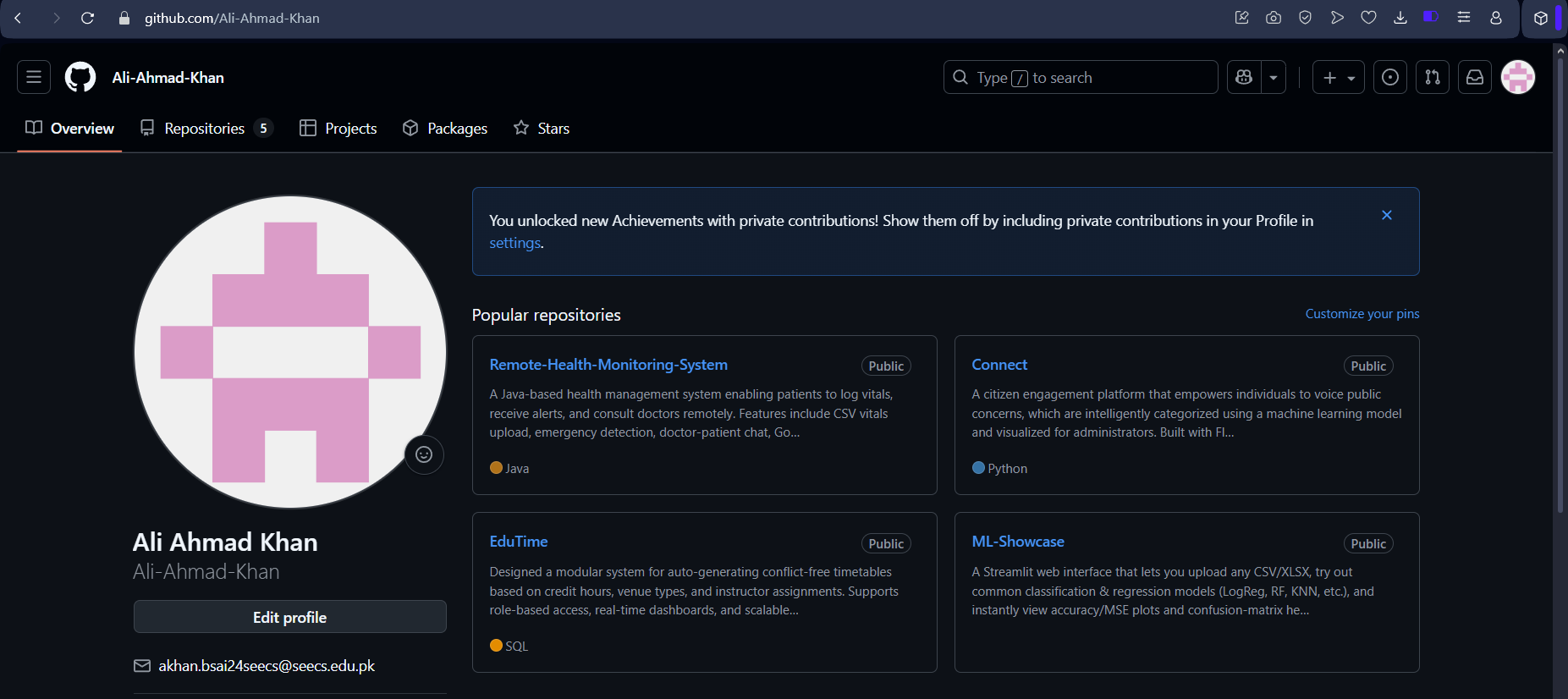
**Task 1: Installing Visual Studio Code (VS Code)**



**Task 2: Installing Git (on Windows)**



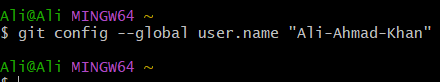
**Task 3: Creating a GitHub Account**



**Task 4: Configure Git**

**Step 1: Set Username**

git config --global user.name "Your Full Name"

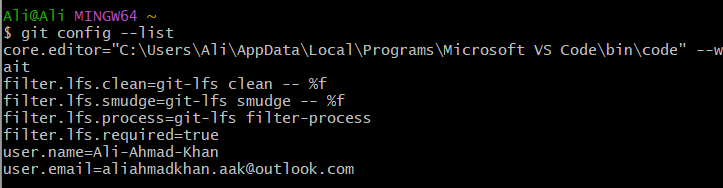


**Step 2: Set Email**

git config --global user.name "Your email"



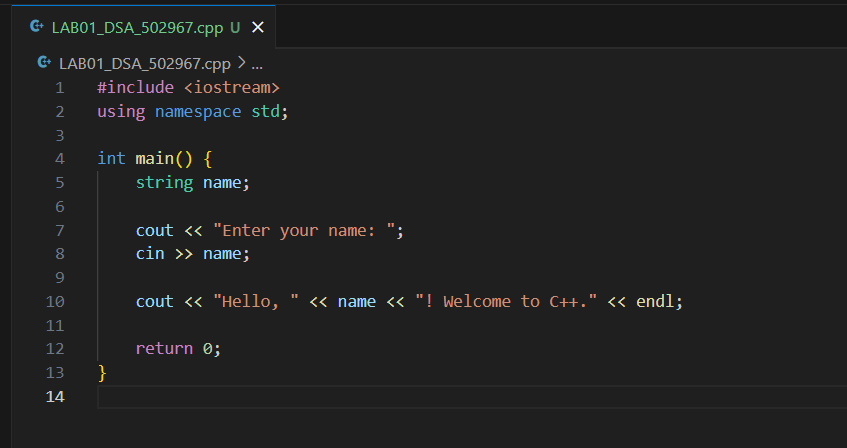
**Step 3: Verify Settings**



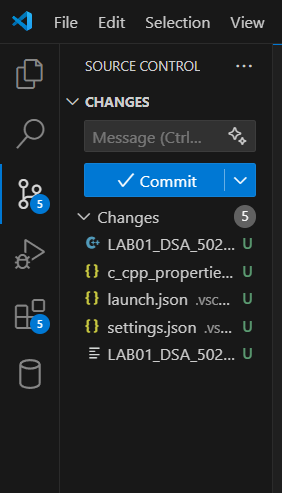
**Task 5: Setting up Git in VS Code & Creating Repositories**

**Method 1**

**Step 1: Create a Project Folder**



**Step 2: Initialize Git Repository**.

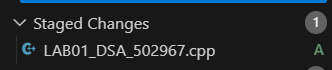


**Step 3: Rename the default branch**

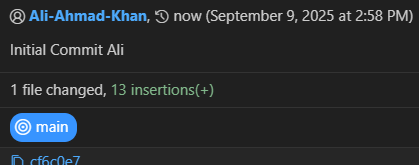
* By default, Git names the initial branch "master." You can change this to "main" (or



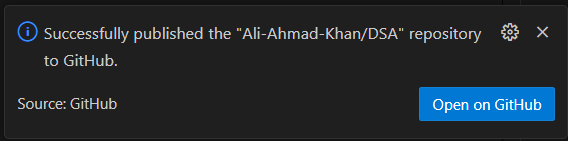
**Step 4: Stage your file changes**

\

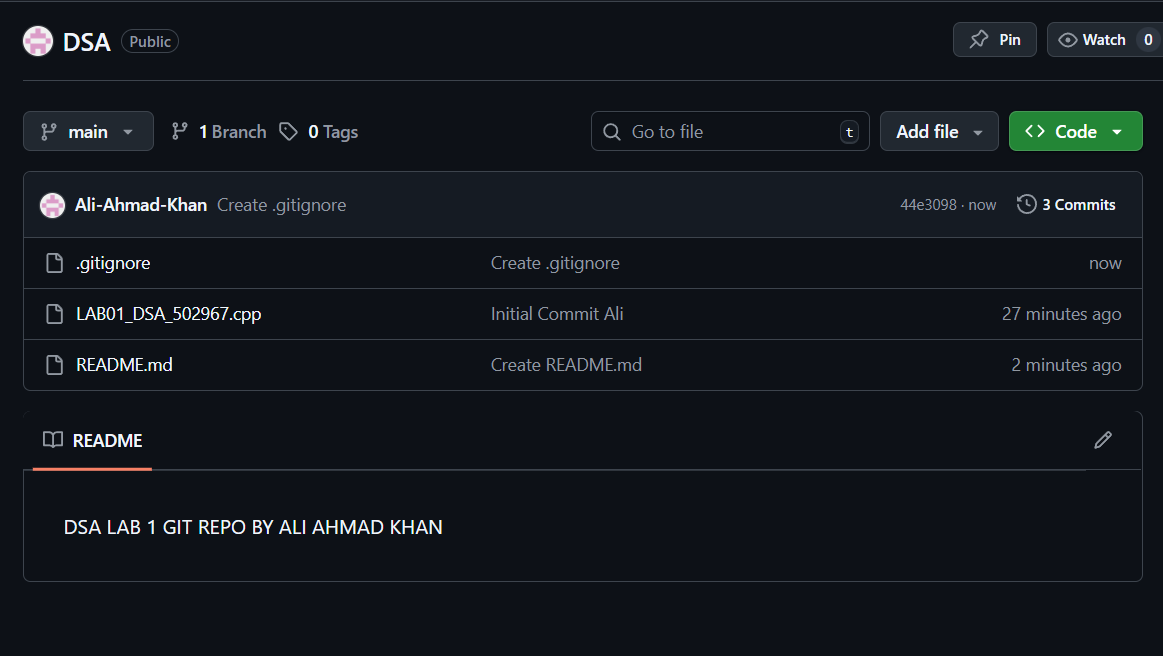
**Step 5: Commit your changes**



**Step 6: Push to a remote repository**

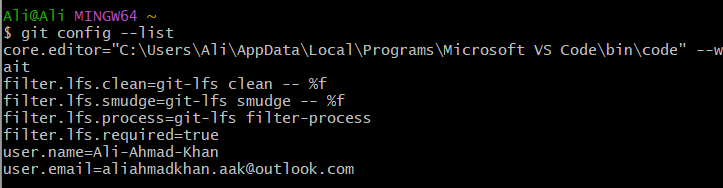


**Method 2**

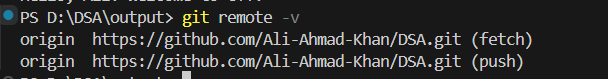


**Clone the repository to a local machine**



**Set Git identity (first time only on a machine)**

### **Verify the remote**



### **Create a simple C++ starter file**

#include <iostream>

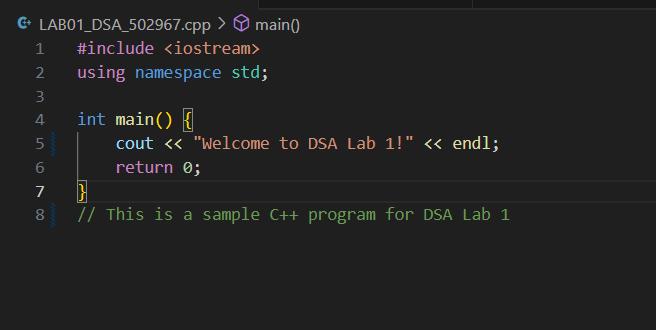
using namespace std;

int main() {

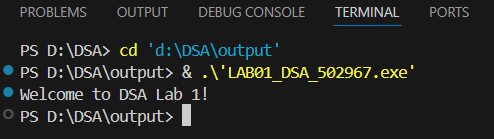
cout << "Welcome to DSA Lab 1!" << endl;

return 0;

}

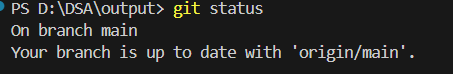


### **Build/run locally (example using g++)**



### **Stage, commit, and push**

git status



git add hello.cpp

git commit -m "Add hello.cpp starter program"

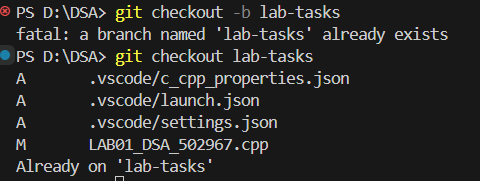
git push origin main # If your default branch is master, use 'master'

**ALREADY PUSHED THE CHANGES IN METHOD 1 AND THE REPO IS LIVE**

**Task 6: Git Workflow Practice (Branch → Implement → PR → Merge)**

### **Create and switch to a feature branch**

git checkout -b lab-tasks



### **Add module files**

* 1. Create utilities.h:

#ifndef UTILITIES\_H

#define UTILITIES\_H

int add(int a, int b); // placeholder example

#endif

**CREATED**

* 1. Create utilities.cpp:

#include "utilities.h"

int add(int a, int b) { return a + b; }

**CREATED**

### **Add a quick test driver (temporary, will be replaced by full test file later)**

File: test\_utilities.cpp

#include <iostream>

#include "utilities.h"

using namespace std;

int main() {

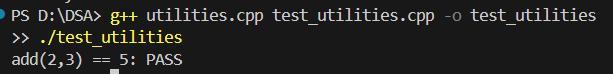
cout << "add(2,3) == 5: " << (add(2,3) == 5 ? "PASS" : "FAIL") << endl;

return 0;

}

g++ utilities.cpp test\_utilities.cpp -o test\_utilities

./test\_utilities



### **Stage and commit incrementally**

git add utilities.h utilities.cpp

git commit -m "Add utilities module with add(a,b)"

git add test\_utilities.cpp

git commit -m "Add smoke test for utilities::add"

### **Push the branch and create a PR**

git push -u origin lab-tasks

* On GitHub: Open Pull Request → compare lab-tasks into main.
* Add a clear PR title & description (what changed, why, how tested).
* Request a review (peer or instructor as applicable).
* Make changes locally->commit->push

### **Merge the PR**

### **Sync the local main**

### **Simulate a simple merge conflict**

* 1. On GitHub: edit README.md in main (add a new line).
  2. Locally on a new branch, edit the same line in README.md, commit, and push.
  3. Open a PR; GitHub will show a conflict.
  4. Students will resolve via the GitHub conflict editor or locally:

git fetch origin

git checkout lab-tasks

git merge origin/main # triggers conflict locally

# Edit the conflicted files to keep the correct content

git add README.md

git commit # completes the merge

git push # PR becomes mergeable

**Task 7: Find All Indices of an Element**

Write a function that finds **all indices** of a given element in an array.

* Input: Array and a key.
* Output: Vector of indices.
* Test cases:
  + Multiple occurrences.
  + Key not present.
  + Empty array.

## **Task 8 – String Pattern Matching (Naive Algorithm)**

Write a function that finds the first occurrence of a substring (pattern) in a given text. If not found, the function will return -1.

* Test cases:
  + Pattern at the beginning.
  + Pattern at the end.
  + Pattern not present.
  + Empty pattern.

**Git Workflow Practice**

* Stage and commit the initial version of your code.
* Modify the code (e.g., add more test cases or comments).
* Commit again with a new commit message.
* Push all commits to GitHub.

**Deliverables**

* A single Word document containing:
  + Source code of all tasks with proper indentation and comments.
  + Screenshots of program outputs.
  + Answers to written questions (if any).
* A GitHub repository containing:
  + All source code files (.cpp, .h).
  + A separate test file with unit tests for each function.
* Students must submit the Word document on LMS before the deadline.
* File naming convention: YourFullName(Reg#)\_Lab1.docx.

**Note:** Students are required to upload the lab on LMS before deadline.

Use proper indentation and comments. Lack of comments and indentation will result in deduction of marks.

**Lab Rubrics**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Assessment** | **Doesn’t meet Expectation**  **(1-2)** | **Meets Expectation**  **(3-4)** | **Exceeds Expectation**  **(5)** | **Marks** |
| **Software Problem Realization**  **(CLO2 – PLO2)** | The Student is unable to understand and **outline** the problem and doesn’t use the relevant method to solve it. | The student requires some guidance to completely comprehend the problem and to **differentiate** the data structure and algorithm comprehensively. | The student fully understands the given problem, is able to **analyze** the relevant method to solve it, and develops a detailed program flow. |  |
| **Software Tool Usage**  **(CLO3 – PLO5)** | The student has no idea on how to use the basic tools of the software. The codes have syntax errors, and parts of the codes are missing. Also, they are unable to **imitate** the required output | The student has a limited command on the basic tools of the software and **operated it under supervision**. The codes are correct in terms of their syntax, however, the program output is not always correct. | The student has full command on various tools available in the software. Furthermore, his/her coding is complete and functional, and the program output is correct. Moreover, they can easily **manipulate the code** to design a particular solution |  |
| **Ethics and Adherence to Laboratory Safety Rules**  **(CLO4 - PLO 8)** | The student does not **behave according to** the professional ethics by following ethical norms applicable to the software industry such as acknowledgement while using publicly available data/ code. Disturbs the lab environment, doesn’t take care of safety measures, and/or isn’t punctual. | The student partially demonstrate their commitment to professional ethics by following ethical norms applicable to the software industry such as referencing and acknowledgement while using publicly available data/ code. **Exhibits** better behavior, works by taking into account the safety measures, and is punctual. | The student clearly **express** the commitment to professional ethics by following ethical norms applicable to the software industry such as referencing and acknowledgement while using publicly available data/ code. Encourages others to maintain lab decorum, and alerts them to follow safety measures. |  |