Programming Fundamentals

BS (CS) _Fall_2025

Lab_13 Tasks

Learning Objectives:

1. Functions II (Pass by reference, Default Arguments, Overloading)
2. Static variables in Function

Lab Tasks

Submission Instructions

1. Name each Task question as **i25XXXX_Task<NO>** **e.g.** i250000_Task1.cpp
2. Compress all **.cpp** files into a **.zip** file, and name it as *ROLLNO_SEC_LAB13* **e.g. i25XXXX_A_LAB13.**
3. Now you have to submit this zipped file on Google Classroom.
4. If you don't follow the above-mentioned submission instruction, you will be marked **zero.**
5. Plagiarism in the Lab Task will result in **zero** marks in the whole category.

Zero Tasks

Q1. Dry Run the code

```
#include <iostream>
using namespace std;
void displayStars(int = 3, int = 1);
int main()
{
    displayStars();

    displayStars(5);

    displayStars(4, 3);
    return 0;
 }
void displayStars(int cols, int rows)
{
    for (int down = 0; down < rows; down++)
    {
        for (int across = 0; across < cols; across++)
        cout << "*";
        cout << endl;
    }
}
```

Q2. Write multiple overloaded functions named area. Each function calculates the area of a different shape (rectangle, circle, square, triangle).

- area of a rectangle (length(int) * width(int)).
- area of a circle (πr^2). where pi can be defined inside the function and radius is a float.
- area of a square. length(int) * length(int)
- area of a triangle. (base × height × ½) where base and height are double.

In main, demonstrate calling each version of area with appropriate arguments.

Lab Tasks

Q3. You are building a payroll module for a company that pays employees under three different payment structures. Because this module must integrate with an older system, every salary calculation routine must use the same function name **Calculate_Payment.**

Each employee type requires different information, so the function will be invoked with different parameter sets depending on the case.

All versions of this function must compute the employee's salary and update internal tallies. No version returns a value.

Employee categories:

1. **Full-Time Employee:**
   Paid a monthly amount based on pay scale:
   - A → 1000/day
   - B → 2000/day
   - C → 3000/day
   - D → 4000/day

   A month normally counts as 30 days, but the number of paid days may be adjusted in special cases.

2. **Daily Employee:**
   Paid by days worked and work hours per day.

3. **Hourly Employee:**
   Paid by total hours worked and hourly wage.

In main, the user repeatedly selects an employee type and enters the needed information.

The program must internally keep track and display:

how many full-time employees were processed

how many daily employees were processed

how many hourly employees were processed

the total amount paid for each category

Q4. You are part of a development team working on a matrix editing module. The tool uses a 5 × 5 matrix to store processed data(numbers). Your task is to implement functionality that allows users to update specific regions of this matrix through a menu based program.

The user will provide the update region using a 2 × 2 matrix representing the coordinates:

- The top-left (start) index of the region.
- The bottom-right (end) index of the region.

To maintain consistency across the codebase, your lead developer has enforced strict rules on how the program must be structured:

Function Rules

- You may create only two functions in addition to main.
- Both functions must use the same name.
- The two functions must behave as follows:
  - Function 1: Prints the current 5×5 matrix.
  - Function 2: Updates the matrix by adding a value to the specified region.

- The main function is only allowed to:
  - Take user input
  - Display a menu
  - Call the appropriate function

Menu Options

- Add a given value: User enters the value and the 2×2 index matrix.
- Add 1: Add 1 to the region defined by the index matrix.
- Print the current matrix.

Q5. You are developing a component for a image-processing system. As part of an early prototype, the system works with a 2 × 2 integer matrix . The matrix must be rotated(clockwise) based on user instructions.

In the main function, the program should read the 2×2 matrix once. After that, the system repeatedly asks the user how many times they want to rotate the matrix. The program continues to request a rotation count until the user enters a negative number, which ends the process.

All matrix rotations will be handled by a single function. The function must accept only two parameters. The 2×2 matrix, number of rotations requested by the user. No other paramters are allowed.

The function must track and maintain the total number of rotations performed. The matrix is allowed to be rotated a maximum of 10 times in total. If the user requests more rotations than allowed, only the remaining permitted rotations should be applied. After each call, the function must display how many rotations are still available out of the maximum of 10 . The rotated matrix must be printed in the main function, everytime after the function calls.