

# ECE 4122/6122 Advanced Software Engineering

## Homework Assignment 0: C++ Fundamentals

**Due Date:** Feb 3<sup>rd</sup>, 2026 by midnight.

**Total Points:** 100

**Submission:** Submit all .cpp source files to Canvas. Each problem should be in a separate file named Problem1.cpp, Problem2.cpp, etc.

---

### Learning Objectives

Upon completion of this assignment, students will demonstrate proficiency in: - Writing syntactically correct C++ programs with proper structure - Implementing iterative solutions using for, while, and do-while loops - Applying conditional logic with if, else if, else, and switch statements - Understanding the difference between pass-by-value and pass-by-reference - Creating and using function templates for generic programming

---

### Problem 1: Statistical Analysis with Pass-by-Reference (25 points)

Write a C++ program that analyzes a collection of floating-point sensor readings. Your program must:

1. Prompt the user to enter the number of readings (between 5 and 100)
2. Dynamically read that many double values from standard input
3. Implement the following functions that compute statistics **using pass-by-reference** for output parameters:

```
void computeStatistics(const double data[], int size,
                      double& mean, double& variance, double& stdDev);
```

```
void findExtrema(const double data[], int size,
                 double& minVal, double& maxVal, int& minIndex,
                 int& maxIndex);
```

4. Display all computed statistics with appropriate formatting (2 decimal places)

**Requirements:** - Use a **for** loop to read input values - Use appropriate loops for statistical calculations - Validate that the number of readings is within the specified range using conditional statements - The **const** keyword must be used appropriately to prevent modification of input arrays

### Problem 1 Sample Output:

```
Enter number of readings (5-100): 6
Enter reading 1: 23.5
Enter reading 2: 18.2
Enter reading 3: 31.7
Enter reading 4: 25.0
Enter reading 5: 19.8
Enter reading 6: 27.3
```

Statistical Analysis Results:

```
-----
Mean:                24.25
Variance:            22.69
Standard Deviation:  4.76
Minimum Value:       18.20 (at index 1)
Maximum Value:       31.70 (at index 2)
```

---

### Problem 2: Matrix Operations with Pass-by-Value vs Pass-by-Reference (25 points)

Write a program that demonstrates the difference between pass-by-value and pass-by-reference through matrix operations. Implement the following:

1. A 3x3 matrix represented as a 2D array
2. The following functions:

*// Pass-by-value: Returns a scaled copy, original unchanged*

```
void scaleMatrixByValue(double matrix[3][3], double scalar);
```

*// Pass-by-reference: Modifies the original matrix in place*

```
void scaleMatrixByReference(double (&matrix)[3][3], double scalar);
```

*// Demonstrate with a struct*

```
struct Matrix3x3 {
    double data[3][3];
};
```

```
Matrix3x3 scaleMatrixCopy(Matrix3x3 m, double scalar);
```

*// By value*

```
void scaleMatrixInPlace(Matrix3x3& m, double scalar);
```

*// By reference*

3. Your main() function must:
  - Query the user to initialize the 3x3 matrix with values
  - Call each function and display the matrix before and after each call
  - Use nested **for** loops for matrix traversal
  - Clearly demonstrate (with printed output) which functions modify the original and which do not

**Requirements:** - Include comments explaining why the array version of pass-by-value doesn't work as expected in C++.

---

### Problem 3: Number Classification with Control Structures (20 points)

Write a program that classifies integers based on multiple mathematical properties. For each number entered:

1. Determine if it is:
  - Positive, negative, or zero
  - Even or odd
  - Prime or composite (for positive integers > 1)
  - A perfect square
  - A Fibonacci number (within the first 50 Fibonacci numbers)

2. Implement the following functions:

```
bool isPrime(int n);
bool isPerfectSquare(int n);
bool isFibonacci(int n);
void classifyNumber(int n, bool& isPositive, bool& isEven,
                   bool& prime, bool& perfectSquare, bool& fibonacci);
```

3. Use a while loop to continuously prompt for numbers until the user enters a sentinel value (e.g., -9999)
4. Use a switch statement to categorize numbers into ranges:
  - "Small" (1-100)
  - "Medium" (101-10000)
  - "Large" (10001-1000000)
  - "Very Large" (> 1000000)

**Requirements:** - Use appropriate loop structures for primality testing - Combine multiple conditions using logical operators (&&, ||, !) - Handle edge cases (0, 1, negative numbers) appropriately

#### Sample Output:

Enter an integer (-9999 to quit): 28

Analysis of 28:

Sign:	Positive
Parity:	Even
Primality:	Composite
Perfect Square:	No
Fibonacci:	No
Magnitude:	Small

Enter an integer (-9999 to quit): 13

Analysis of 13:

Sign:	Positive
Parity:	Odd
Primality:	Prime
Perfect Square:	No
Fibonacci:	Yes
Magnitude:	Small

---

#### Problem 4: Generic Programming with Function Templates (30 points)

Implement a set of function templates that work with multiple data types. This problem emphasizes code reusability through generic programming.

##### *Part A: Basic Function Templates (10 points)*

Implement the following function templates:

*// Returns the larger of two values*

```
template <typename T>
T maximum(T a, T b);
```

*// Returns the smaller of two values*

```
template <typename T>
T minimum(T a, T b);
```

*// Swaps two values using references*

```
template <typename T>
void swapValues(T& a, T& b);
```

*// Returns the absolute value*

```
template <typename T>
T absoluteValue(T value);
```

Demonstrate each template with at least three different data types: int, double, and float.

##### *Part B: Array Processing Templates (10 points)*

Implement function templates for array operations:

*// Finds and returns the sum of all elements*

```
template <typename T>
T arraySum(const T arr[], int size);
```

*// Finds and returns the average (always returns double)*

```
template <typename T>
double arrayAverage(const T arr[], int size);
```

*// Reverses the array in place*

```
template <typename T>
```

```
void reverseArray(T arr[], int size);

// Linear search - returns index or -1 if not found
template <typename T>
int linearSearch(const T arr[], int size, T target);

// Bubble sort in ascending order
template <typename T>
void bubbleSort(T arr[], int size);
```

You can use code from online for Bubble sort i.e.:

<https://www.geeksforgeeks.org/cpp/bubble-sort-in-cpp/>

**Requirements for Part B:** - Use for loops for array traversal - Use nested loops for bubble sort - Include conditional logic for search and sort comparisons

### Submission Guidelines

1. **Code Style:**
  - Use meaningful variable and function names
  - Include header comments with your name, GTID, and problem description
  - Indent code consistently (4 spaces recommended)
  - Comment complex logic
2. **Compilation:**
  - All code must compile without errors on pace-ice using: `g++ -std=c++17 -Wall -Wextra -o program ProblemX.cpp`
  - Warnings should be addressed
3. **File Naming:**
  - Problem1.cpp, Problem2.cpp, Problem3.cpp, Problem4.cpp
4. **Testing:**
  - Test your programs with various inputs including edge cases
  - Include sample output in comments at the end of each file

### Grading Rubric

*There is an automatic 40% loss per problem if the code does not compile on the pace-ice system.*

Criteria	Points
<b>Problem 1</b>	<b>25</b>
Correct statistical calculations	10
Proper use of pass-by-reference	8
Input validation and loop usage	4

Criteria	Points
Code style and formatting	3
<b>Problem 2</b>	<b>25</b>
Correct matrix operations	8
Clear demonstration of value vs reference	10
Menu system with do-while loop	4
Comments explaining C++ array behavior	3
<b>Problem 3</b>	<b>20</b>
Correct classification functions	10
Proper use of all control structures	6
Edge case handling	4
<b>Problem 4</b>	<b>30</b>
Part A: Basic templates working correctly	15
Part B: Array templates with proper loops	15

LATE POLICY Element	Percentage Deduction	Details
Late Deduction Function	score – 0.5 * H	H = number of hours (ceiling function) pass deadline

---

### Academic Integrity

This is an individual assignment. You may discuss concepts with classmates, but all code must be your own. Use of AI coding assistants, copying from online sources, or sharing code with other students constitutes an Honor Code violation.

---

**Questions?** Post on the course Piazza or attend office hours.

## Appendix A: Coding Standards

### Indentation:

When using if/for/while statements, make sure you indent 4 spaces for the content inside those. Also make sure that you use spaces to make the code more readable.

For example:

```
for (int i ; i < 10 ; i++)
{
```

```
    j = j + i;  
}
```

If you have nested statements, you should use multiple indentions. Each { should be on its own line (like the for loop) If you have else or else if statements after your if statement, they should be on their own line.

```
for (int i; i < 10; i++)  
{  
    if (i < 5)  
    {  
        counter++;  
        k -= i;  
    }  
    else  
    {  
        k +=1;  
    }  
    j += i;  
}
```

### **Camel Case:**

This naming convention has the first letter of the variable be lower case, and the first letter in each new word be capitalized (e.g. firstSecondThird).

This applies for functions and member functions as well!

The main exception to this is class names, where the first letter should also be capitalized.

### **Variable and Function Names:**

Your variable and function names should be clear about what that variable or function represents. Do not use one letter variables, but use abbreviations when it is appropriate (for example: "imag" instead of "imaginary"). The more descriptive your variable and function names are, the more readable your code will be. This is the idea behind self-documenting code. There are exceptions such as the index variable of loops.

### **File Headers:**

Every file should have the following header at the top

```
/*
```

```
Author: your name
```

```
Class: ECE4122 or ECE6122 (section)
```

```
Last Date Modified: date
```

```
Description:
```

```
What is the purpose of this file?
```

```
*/
```

### **Code Comments:**

1. Every function must have a comment section describing the purpose of the function, the input and output parameters, the return value (if any).
2. Every class must have a comment section to describe the purpose of the class.
3. Comments need to be placed inside of functions/loops to assist in the understanding of the flow of the code.