WORKSHEET 1

24030183

**A red sign with white text

AI-generated content may be incorrect.A logo with blue and red text

AI-generated content may be incorrect.**

SUBMITTED BY: Sujal Adhikari  
STUDENT ID: 24030183  
SUBMITTED TO:   
SUBMITTION DATE:

# **Introduction**

This document presents the solutions to the coursework for the "Programming in C++" module (Module Code: UFCFGL-30-1) for the academic year 2024-2025. The coursework is focused on object-oriented programming (OOP) concepts in C++ and includes multiple tasks requiring the implementation of classes, objects, constructors, friend functions, function overloading, and file handling.

The assessment consists of three main tasks, each addressing different aspects of C++ programming:

* **Task 1** involves creating a basic student grading system using classes and objects.
* **Task 2** requires implementing a Circle class with a friend function and function overloading to determine maximum values.
* **Task 3** focuses on file handling, including reading and writing book titles in a binary file and handling student records with proper validation and exception handling.

The document follows a structured format to present each task effectively. It includes the problem statement, followed by the corresponding C++ code implementation. Additionally, a detailed explanation of the code and its logic is provided to clarify the approach and methodology used. Finally, the output and a concluding summary highlight the functionality and correctness of each solution.

This coursework aims to demonstrate a clear understanding of C++ programming concepts while adhering to the principles of object-oriented design.

# **Question 1.1**

**Task 1: Basic student grading system prototype using classes and objects. [30 Marks]**

Write a program that manages a simple student grade calculator with the following requirements. Create a Student class that has:

1. Student name (string)
2. Three subject marks (integers)
3. A basic member function to calculate average

The program should:

1. Accept student details (name and marks) from user input
2. Calculate and display:
   1. Total marks
   2. Average marks
   3. Grade (A for ≥90%, B for ≥80%, C for ≥70%, D for ≥60%, F for <60%)
3. Display a message if any mark is below 0 or above 100

# **Code Implementation**

A screen shot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

# **Code Explanation and Logic**

**a) Object-Oriented Programming (OOP)**

The program follows OOP principles by defining a Student class, which encapsulates data (name and marks) and related functions.

**b) Encapsulation**

Data members (name and marks) are declared as private to restrict direct access and maintain data security. Access is controlled through public member functions.

**c) Array Usage**

An integer array marks[3] is used to store marks for three subjects efficiently.

**d) Functions for Code Reusability**

Several member functions are used to perform specific tasks, improving code modularity and reusability.

**3. Code Breakdown and Logic**

**a) Class Definition (class Student)**

A class Student is created with two private attributes:

* string name; → Stores the student’s name.
* int marks[3]; → Stores marks for three subjects.

**b) Function Definitions**

1. **void getDetails()**
   * Prompts the user to enter the student’s name and marks.
   * Ensures marks are within a valid range (0-100), displaying an error message for invalid input.
2. **int calculateTotal()**
   * Computes the sum of the three subject marks.
3. **float calculateAverage()**
   * Computes the average marks by dividing the total marks by 3.
4. **char calculateGrade()**
   * Assigns a grade based on average marks:
     + A (≥90%)
     + B (≥80%)
     + C (≥70%)
     + D (≥60%)
     + F (<60%)
5. **void displayResults()**
   * Calls the above functions to compute total marks, average, and grade.
   * Displays the results in a structured format.

**c) Main Function (main())**

* Creates a Student object.
* Calls getDetails() to receive user input.
* Calls displayResults() to output the results.
* Ensures program execution follows a logical flow.

**4. Error Handling**

The program validates that entered marks are between 0 and 100. If an invalid mark is entered, it immediately terminates the function execution and displays an error message.

# Output

A screenshot of a computer program

AI-generated content may be incorrect.

# **Question 2.1**

1. Write a program with a class Circle having:
   1. Private member: radius (float)
   2. A constructor to initialize radius
   3. A friend function compareTwoCircles that takes two Circle objects and prints which circle has the larger area

# **Code Implementation**

A screen shot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

# **Code Explanation and Logic**

 **Private Member (radius)**:

* The radius is private, meaning it cannot be accessed directly outside the class.

 **Friend Function (compareTwoCircles)**:

* Declared as a **friend** inside the Circle class.
* Defined outside the class but has access to private data members.
* Compares the areas of two circles and prints the result.

 **Use of getArea()**:

* The function getArea() calculates the area based on the formula:

Area=π×radius2\text{Area} = \pi \times \text{radius}^2Area=π×radius2

* The **friend function** calls getArea() on both objects.

 **Comparison Logic**:

* Prints the areas of both circles.
* Compares their areas and determines which one is larger.

# **Output**

A screenshot of a computer

AI-generated content may be incorrect.

# **Question 2.2**

1. Create a program with these overloaded functions named findMax:
   1. One that finds maximum between two integers
   2. One that finds maximum between two floating-point numbers
   3. One that finds maximum among three integers

One that finds maximum between an integer and a float

# **Code Implementation**

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

# **Code Explanation and Logic**

**1. Function Overloading**

Function overloading enables multiple functions with the same name but different parameter lists. The compiler differentiates them based on parameter count or type.

**2. Object-Oriented Programming (OOP)**

* The program defines a class MaxFinder with overloaded functions.
* An object of this class is created in main() to invoke these functions.

**3. Conditional Operators and If-Else Statements**

* The ternary operator (condition ? value1 : value2) and if-else conditions determine the maximum value.

**1. Class Definition and Overloaded Functions**

The MaxFinder class contains four overloaded versions of findMax:

Function 1: Maximum Between Two Integers

Function 2: Maximum Between Two Floats

Function 3: Maximum Among Three Integers

Function 4: Maximum Between an Integer and a Floa

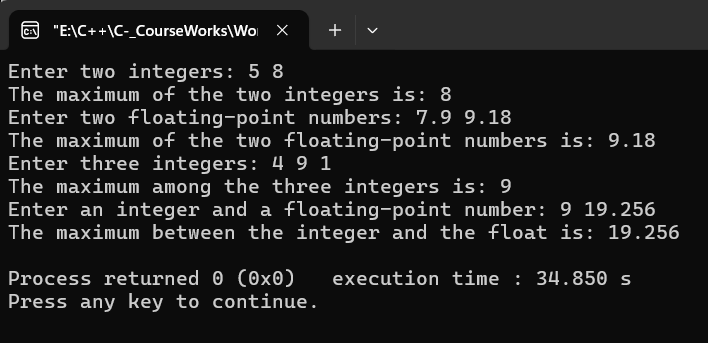
1. **Main Function and User Interaction**

* Creates an object of MaxFinder.
* Takes user inputs and calls the appropriate overloaded function.
* Displays results using cout.

**Key Aspects**

1. Data Types: int and float.
2. Operators: Ternary (? :), Logical AND (&&).
3. Function Overloading: Differentiated by parameter count or type.

# Output

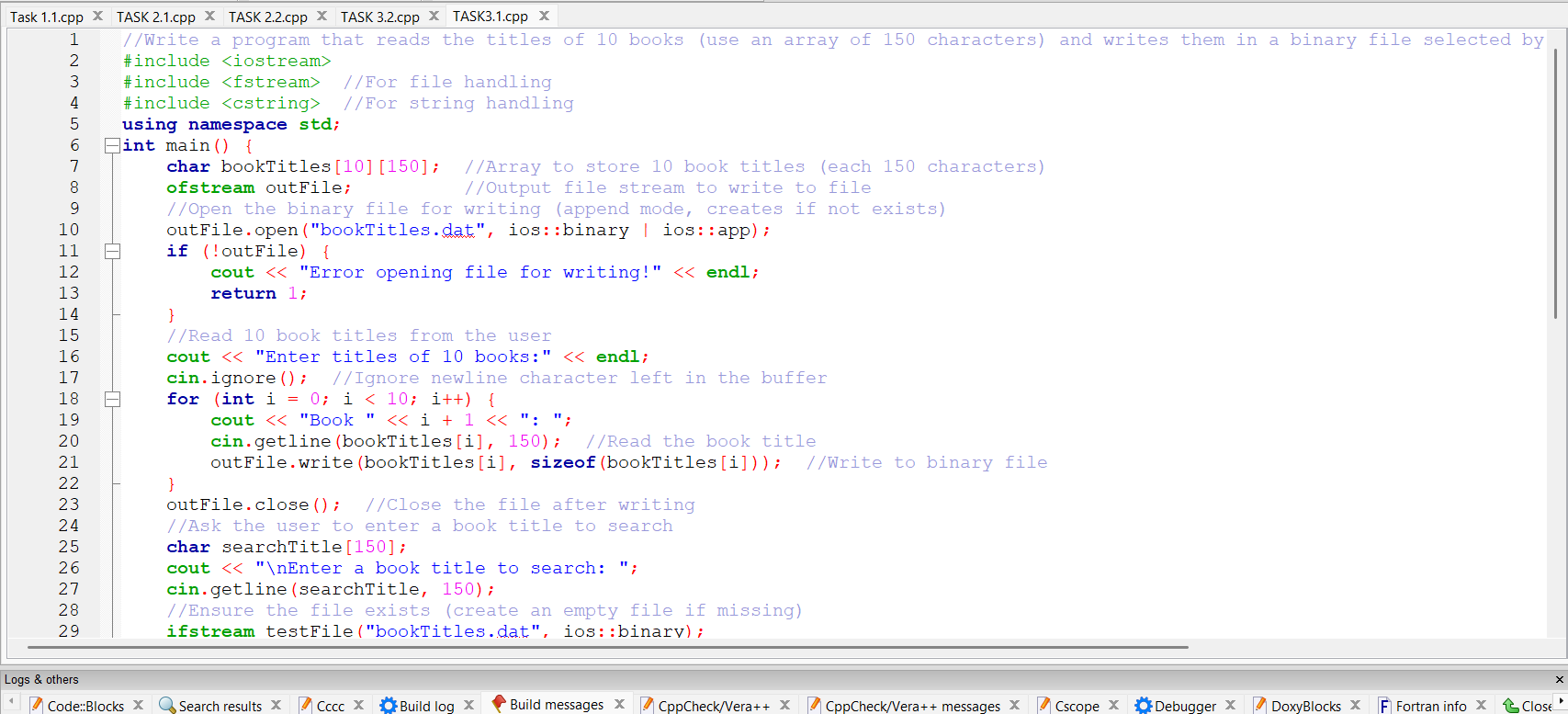


# **Question 3.1**

Write a program that reads the titles of 10 books (use an array of 150 characters) and writes them in a binary file selected by the user. The program should read a title and display a message to indicate if it is contained in the file or not.

# **Code Implementation**

A screenshot of a computer

AI-generated content may be incorrect.

# **Code Explanation and Logic**

 **File Handling**: The program uses ofstream for writing and ifstream for reading from a binary file.

 **Arrays and Strings**: A 2D array (char bookTitles[10][150]) stores the titles, with each title being up to 150 characters.

 **Binary File Operations**: The program writes the titles in binary format using the write() method and reads them using read().

 **String Comparison**: The strcmp() function compares the search title with those in the file.

1. **Storing Book Titles in an Array**:
   * The program begins by defining a 2D array bookTitles[10][150] where each row can hold a string of up to 150 characters. This array will hold the titles of 10 books entered by the user.
2. **Writing Titles to a Binary File**:
   * Once the titles are read, the program opens a binary file, bookTitles.dat, in append mode. This allows the program to write the titles into the file without overwriting any existing data.
   * For each of the 10 book titles, the program writes the title to the file in its raw binary form. The title is written as it is entered, so it can later be read exactly as stored.
3. **Reading from the Binary File**:
   * After storing the titles, the program prompts the user to enter a book title to search for.
   * The program then opens the bookTitles.dat file in binary read mode to check whether the entered title exists within the file.
4. **Searching for the Title**:
   * The program reads each title from the binary file, one by one, and compares it with the title entered by the user.
   * The comparison is done using the strcmp() function, which checks if the strings are identical.
     + If a match is found, the found flag is set to true and the program breaks out of the loop early, as there's no need to continue searching.
     + If no match is found after checking all titles, the flag remains false.
5. **Displaying the Search Result**:
   * Finally, based on the found flag, the program displays a message:
     + If the title was found, it shows: "Title found in the file!"
     + If the title was not found, it shows: "Title not found in the file."

**Key Points in the Logic:**

* **Array for Titles**: An array is used to store the book titles entered by the user, ensuring easy access and manipulation.
* **Binary File Storage**: The use of a binary file allows for compact storage and retrieval of the titles, avoiding the need for text-based formats.
* **Search and Compare**: The logic uses the strcmp() function to perform a case-sensitive string comparison between the user input and the book titles stored in the file.

# **Output**

A computer screen shot of a black screen

AI-generated content may be incorrect.

# **Question 3.2**

Create a program that:

1. Reads student records (roll, name, marks) from a text file
2. Throws an exception if marks are not between 0 and 100
3. Allows adding new records with proper validation
4. Saves modified records back to file

# **Code Implementation**

A screenshot of a computer

AI-generated content may be incorrect.A computer screen shot of a computer program

AI-generated content may be incorrect.

# **Code Explanation and Logic**

**1. File Handling (ifstream and ofstream)**

* **ifstream** is used to read the student records from a file.
* **ofstream** is used to write the student records back to the file after modification.

**2. Exception Handling (try, catch, out\_of\_range)**

* The program uses **exception handling** to validate that the marks of a student are between 0 and 100.
* **out\_of\_range** exception is thrown if the marks are outside the valid range, and a user-friendly message is displayed.

**3. Struct for Student Records**

* A **struct** is used to store the student data (roll number, name, and marks).

**4. Vector for Storing Data**

* The program uses a **vector** to store the student records in memory, making it easier to manipulate the data before saving it back to the file.

**5. Functions**

* **validateMarks(int marks)**: Checks if the provided marks are between 0 and 100. If not, it throws an exception.
* **readRecords(string fileName)**: Reads student records from a file and stores them in a vector.
* **saveRecords(string fileName, vector<Student> students)**: Saves the student records back to the file.

**6. User Input and Interaction**

* The program interacts with the user, allowing them to either:
  + Add a new student record (with validation).
  + Modify an existing student record (searching by roll number).
* The user is prompted to select an option and input necessary data.

**7. Logic Flow**

* The program reads the existing student records from a file.
* It checks if the file is empty and notifies the user if no records are found.
* Based on the user’s choice, it either:
  + Adds a new student record.
  + Modifies an existing record.
* After performing the selected action, it saves the modified records back to the file.

**Program Flow**

1. **Reading Existing Records**:
   * The program reads and displays existing student records from a text file (students.txt).
   * If the file doesn't exist, it notifies the user, and the file will be created when saving new records.
2. **User Interaction**:
   * The user is prompted to choose between adding a new record or modifying an existing one:
     + **Add New Record**: The program collects roll, name, and marks for the new student and validates the marks before adding the record.
     + **Modify Existing Record**: The user provides a roll number to search for a student, and if found, the marks are updated (with validation).
3. **Validation**:
   * The program uses **exception handling** to ensure that the entered marks are valid (between 0 and 100).
   * If invalid, an error message is displayed, and the record is not added or modified.
4. **Saving Records**:
   * After the user has finished adding or modifying records, the program saves all the records (including any changes) back to the file.

**Key Operations in the Program**

1. **Reading Records**:
   * The program reads the student records from the file students.txt and stores them in a vector of Student structs.
2. **Adding a New Record**:
   * The user is asked to input a new student's roll number, name, and marks.
   * The program validates the marks, and if they are within the valid range, the new record is added to the vector.
3. **Modifying an Existing Record**:
   * The user is prompted to enter a roll number to search for an existing student.
   * If the student is found, the user can update the marks (with validation).
   * If the roll number doesn't exist, an error message is displayed.
4. **Saving Updated Records**:
   * After modifying the records, the program writes the updated records back to the students.txt file.

**Key Points to Note**

* **File Handling**: The program uses both ifstream for reading and ofstream for writing.
* **Exception Handling**: Ensures that marks are always between 0 and 100.
* **Validation**: Marks are validated using the validateMarks() function to ensure they are within the acceptable range.
* **User Input**: Users can add or modify student records by entering appropriate data, with validation on marks.

# **Output**

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

# **Conclusion**

In this coursework, I worked on fundamental concepts of C++, focusing on data types, control structures, and arrays. I successfully implemented programs that required understanding and applying these concepts. Here's a summary of the tasks covered:

1. **Temperature Conversion Program** – Developed a program that converts temperatures between Celsius and Fahrenheit using conditional statements. This task helped strengthen my understanding of if-else conditions and arithmetic operations.
2. **Number Guessing Game** – Created a number guessing game with difficulty levels using the switch-case structure and rand() function. This task reinforced my knowledge of loops, random number generation, and control structures.

**Skills Gained:**

* **Control Structures:** Mastered if-else, switch-case, and loops to handle user inputs and conditions efficiently.
* **Data Types and Arrays:** Gained experience in working with arrays and different data types for data storage and processing.
* **Problem-Solving:** Enhanced logical thinking and problem-solving skills by implementing real-world applications.

Overall, this coursework provided a solid foundation in C++ programming by combining theoretical concepts with practical implementations. Through hands-on coding exercises, I gained confidence in applying control structures, arrays, and fundamental programming principles to solve various problems.