**C++ basic and advanced training 08-Jan18 to 12-Jan-18**

**Practice Problems:**

**Basic Concepts:**

1. Print "Hello, World!" in C++.
2. Take user input and print it.
3. Swap two numbers without using a temporary variable.
4. Check if a number is even or odd.
5. Find the largest among three numbers.

**Control Flow:**

1. Implement a simple calculator (addition, subtraction, multiplication, division).
2. Check if a year is a leap year.
3. Find the factorial of a number.
4. Generate Fibonacci series up to n terms.
5. Implement a simple guessing game.

**Arrays and Strings:**

1. Reverse an array.
2. Find the sum of elements in an array.
3. Check if a given array is sorted.
4. Find the second largest element in an array.
5. Implement a function to concatenate two strings.

**Pointers and References:**

1. Swap two numbers using pointers.
2. Find the length of a string using pointers.
3. Reverse a string using pointers.
4. Implement a function to find the maximum element in an array using pointers.
5. Create a dynamic array of integers and perform operations.

**Functions:**

1. Check if a number is prime using a function.
2. Find the GCD (Greatest Common Divisor) of two numbers using a function.
3. Implement a function to calculate the power of a number.
4. Write a function to check if a string is a palindrome.
5. Calculate the area of a circle using function overloading.

**Classes and Objects:**

1. Create a class representing a student with attributes like name, roll number, and marks.
2. Implement a constructor and destructor in a class.
3. Create a class for a bank account with deposit and withdrawal methods.
4. Use inheritance to model different shapes (circle, square, rectangle).
5. Implement a simple calculator using class and object concepts.

**Operator Overloading:**

1. Overload the '+' operator to concatenate two strings.
2. Implement a complex number class and overload the '+' and '-' operators.
3. Overload the '<<' operator to display information about an object.
4. Implement a matrix class and overload the '+' operator for matrix addition.
5. Overload the '==' operator to compare two objects.

**File Handling:**

1. Read data from a file and display it.
2. Write data to a file and read it back.
3. Copy the contents of one file to another.
4. Implement a program to count the number of words in a text file.
5. Serialize and deserialize an object to/from a file.

**Dynamic Memory Allocation:**

1. Implement a program to allocate memory for an integer dynamically.
2. Create a dynamic array of objects and perform operations.
3. Allocate memory for a 2D array dynamically.
4. Implement a stack using dynamic memory allocation.
5. Reverse a linked list.

**Recursion:**

1. Implement factorial using recursion.
2. Find the nth Fibonacci number using recursion.
3. Check if a string is a palindrome using recursion.
4. Calculate the sum of digits of a number using recursion.
5. Implement the Tower of Hanoi problem.

**Templates:**

1. Create a template function for finding the maximum of two elements.
2. Implement a generic stack using templates.
3. Use template specialization to handle different data types.
4. Create a generic linked list using templates.
5. Implement a template class for a mathematical operation.

**Standard Template Library (STL):**

1. Use vectors to implement a simple database.
2. Implement a queue using the queue container from the STL.
3. Sort an array using the sort algorithm from the STL.
4. Use the map container to count the frequency of words in a text.
5. Implement a custom comparator for sorting a vector of custom objects.

**Exception Handling:**

1. Implement a try-catch block for handling division by zero.
2. Create a custom exception class and throw an exception using it.
3. Handle multiple exceptions in a single catch block.
4. Implement RAII (Resource Acquisition Is Initialization) using exception handling.
5. Implement a program to demonstrate the concept of stack unwinding.

**Multithreading:**

1. Create a simple multithreaded program.
2. Implement thread synchronization using mutex.
3. Use condition variables to synchronize threads.
4. Implement a producer-consumer problem using threads.
5. Implement a multithreaded program for parallel matrix multiplication.

**Networking:**

1. Implement a simple client-server program using sockets.
2. Create a chat application using socket programming.
3. Implement a file transfer program using sockets.
4. Use TCP sockets for communication between a client and server.
5. Implement a simple HTTP server.

**Data Structures:**

1. Implement a binary search tree (BST).
2. Perform operations on a binary tree (insertion, deletion, traversal).
3. Implement a hash table with collision handling.
4. Use a trie to implement a dictionary.
5. Implement a graph and perform DFS (Depth-First Search) and BFS (Breadth-First Search).

**Design Patterns:**

1. Implement the Singleton design pattern.
2. Use the Observer pattern to implement a simple event system.
3. Implement the Factory Method pattern.
4. Use the Strategy pattern to implement different sorting algorithms.
5. Implement the Decorator pattern for extending functionality.

**Advanced Topics:**

1. Implement move semantics for a custom class.
2. Use smart pointers to manage memory.
3. Implement a custom allocator.
4. Create a custom memory pool.
5. Implement a garbage collector.

**C++11/14/17 Features:**

1. Use auto keyword for type inference.
2. Implement lambda expressions.
3. Use range-based for loops.
4. Implement smart pointers (unique\_ptr, shared\_ptr).
5. Use constexpr and noexcept.

**Software Development Practices:**

1. Implement unit tests for a C++ program using a testing framework.
2. Use version control (e.g., Git) for managing code repositories.
3. Follow coding standards and best practices.
4. Implement logging in a C++ application.
5. Perform code review and discuss code optimization strategies.