# ****Intern Intelligence: C++ Custom Library Documentation****

* **Author**: Muhammad Ali Sajjad
* ****ID: lDiovyubhLEhXS2waSbJ July 7, 2025****

## ****Project Title:****

### Custom C++ Library for Data Structures, Algorithms, Math Utilities, and File Handling

## ****Internship:****

**Intern Intelligence (Summer Internship Program)**

## ****Name:****

**Muhammad Ali Sajjad**

## ****Table of Contents****

1. Project Overview

2. Library Architecture

3. Implemented Modules

4. Usage Instructions

5. Sample Demonstrations

6. Conclusion and Learning

7. CROW API Setup & Execution Instructions

8. API Testing Checklist

9. Testing Options

## Project Overview

This document presents a comprehensive C++ library developed as part of the **Intern Intelligence** internship program. The library contains a wide set of functionalities including:

* Fundamental data structures (arrays, linked lists, stacks, queues)
* Advanced trees and graph traversal algorithms
* Mathematical utilities such as factorial, GCD/LCM, isPrime
* Classic sorting and searching algorithms
* File handling operations for reading/writing records

The goal of this project was to develop a **reusable, modular, and testable** codebase that demonstrates core object-oriented and algorithmic skills using C++ templates and manual memory management.

## Library Architecture

The system is structured into individual header files for separation of concerns:

| Module | Header File | Purpose |
| --- | --- | --- |
| Math Functions | Math.h | GCD, LCM, factorial, isPrime, reverse |
| Sorting | Sorting.h | Bubble, Selection, Insertion Sort |
| Searching | Searching.h | Linear Search, Binary Search |
| Dynamic Array | VectorArray.h | Resizable array (like std::vector) |
| Linked Lists | SinglyLL.h, etc. | Singly & Doubly Linked Lists |
| Stack/Queue | Stack.h, Queue.h | Based on linked list |
| Tree Structures | BST.h, AVL.h | Insertion, deletion, traversal |
| Graphs | Graph.h | BFS, DFS, shortest path |
| File Handling | FileHandling.h | Read/Write leaderboard from file |

Each module was tested using a unified main.cpp demo file where all functionalities were tested with sample input/output to ensure correctness and usability.

## Implemented Modules

### 1. Math Utilities (Math.h)

* Abs(float)
* isPrime(int)
* factorial(int)
* ReverseNumber(int)
* GCD(int, int)
* LCM(int, int)
* power(base, exponent)
* isPalindrome(int)
* sumOfDigits(int)
* isEven(int)

A set of frequently used numeric operations was implemented to support general-purpose computations:

* Absolute value, factorial, prime-checking
* Number reversal, GCD, LCM using recursion and iterative logic
* Power calculation, sum of digits, even/odd checker
* Palindrome check for numeric inputs

The Math class encapsulates these as modular functions suitable for integration with algorithmic tasks or validation procedures.

### 2. Sorting Algorithms (Sorting.h)

* SelectionSort
* BubbleSort
* InsertionSort
* Used on both arrays and file data for leaderboard

Three classic sorting techniques have been implemented:

* **Selection Sort** – Used for in-place sorting (e.g., leaderboard)
* **Bubble Sort** – Useful for early-exit optimization
* **Insertion Sort** – Ideal for nearly sorted arrays

These were tested on both integer arrays and file-based score data.

### 3. Searching Algorithms (Searching.h)

* LinearSearch
* BinarySearch (recursive)

The searching module includes:

* **Linear Search** – Iterative check across the array
* **Binary Search** – Recursive implementation assuming sorted input
* (Optional future: Jump Search, Interpolation Search)

All searches are type-generic and tested on both custom dynamic arrays and static arrays.

### 4. File Handling (FileHandling.h)

* ReadFile(string filename) — reads and displays top 10 scores
* WriteFile(string filename, int score, string name) — writes top scores
* Uses internal sorting for top scores management

A self-built vector-like class with:

* push\_back, pop\_back, insert, erase, clear, resize
* Operator overloading for []
* Index-based assignment and value search
* Range-based operations (insert/assign multiple elements)

Memory is managed manually to demonstrate deep understanding of dynamic allocation and resizing logic.

### 5. Dynamic Array (VectorArray.h)

* Custom implementation similar to std::vector
* Functions: push\_back, insert, resize, erase, assign, find, clear, operator[]

### 6. Linked Lists

* Singly Linked List
* Circular Linked List
* Doubly Linked List
* Basic operations: insert, delete, search, display

Operations include insertions (head/tail/position), deletions, traversal, and search, all handled via a generic Node class.

### 7. Stack & Queue (template-based)

* Linked list implementation
* Methods: push, pop, peek, isEmpty, show

Implemented as:

* Stack (LIFO): push, pop, peek, isEmpty
* Queue (FIFO): enqueue, dequeue, peek, isEmpty

Both use a Node<T> class, making them memory-safe and flexible.

### 8. Binary Search Tree (BST.h)

* Insertion
* Deletion
* Inorder, Preorder, Postorder traversal
* Search functionality
* **Binary Search Tree (BST)**: Supports insertion, deletion, and all traversal methods (inorder, preorder, postorder), along with node search.
* **AVL Tree**: Extended BST with self-balancing via rotations (LL, RR, LR, RL). Automatically maintains height balance on insertion.

Both are implemented with pointers and tested via interactive menus in main.cpp.

### 9. AVL Tree (AVL.h)

* Balanced version of BST
* Rotations implemented
* Search and insert

### 10. Graphs (Graph.h)

* BFS and DFS traversal
* Shortest path logic
* Represented using adjacency list and queue
* BFS and DFS implemented using custom Queue and Recursion
* Shortest Path logic discussed (e.g., BFS-based level count)
* Ready for expansion to Dijkstra or A\* in future versions

### 11. Basic FileHandling (FileHandling.h)

Handles leaderboard management:

* **ReadFile**: Displays top 10 scores in formatted layout
* **WriteFile**: Sorts new entries and truncates file to top N scores
* Internally uses Selection Sort to maintain order

## Usage Instructions

1. Clone or copy all .h files into your project directory.
2. Include headers in your main.cpp:

#include "Math.h"

#include "VectorArray.h"

#include "Sorting.h"

#include "Searching.h"

#include "BST.h"

#include "AVL.h"

#include "Graph.h"

1. Create objects and call functions. Example:

VectorArray<int> v;

v.push\_back(10);

v.insert(5, 1);

v.show();

1. For math:

Math m;

cout << "Factorial: " << m.factorial(5);

## Sample Demonstrations

A complete main.cpp file has been used to demo:

* All data structure methods (push, pop, insert, traversal)
* Sorting and searching usage
* File read/write demo
* Math utilities output
* Graph BFS/DFS traversal on sample nodes
* BST and AVL insertion/traversal

## CROW API Setup & Execution Instructions

Here’s how you should write it in your Word file:

### 7. CROW API Setup & Execution Instructions

To extend the functionality of this C++ library with modern web-based access, a lightweight API layer was developed using the **CROW framework** (a C++ micro web framework similar to Flask for Python). This allows various library functions to be accessed through RESTful endpoints on localhost.

#### 🔨 Crow API Environment Setup

To run the API version of this project:

1. **Download** asio.zip (from the official Asio standalone release).
2. **Extract it** anywhere (e.g., C:\libraries\asio-1.28.1).
3. In **Visual Studio** (or your IDE):
   * Right-click your project > **Properties**
   * Go to **C/C++ > Language**
     + Set **C++ Language Standard** to C++17 or later
   * Go to **C/C++ > General > Additional Include Directories**
     + Add:
       - $(ProjectDir)
       - The **include** folder path of the extracted asio folder  
         (e.g., C:\libraries\asio-1.28.1\include)
4. Apply changes and **build the project**.

If no errors occur, your API server is ready to launch via main.cpp.

#### ▶️ Running the API Server

* Run the project (or build and execute).
* Open your browser and visit:

## API Testing Checklist:

Here is a **checklist of the endpoints** you can test. Open them in tabs or paste into your browser after starting your Crow app.

Replace localhost:18080 with your port (if changed).

### ****Math API****

| Feature | URL | Expected Output |
| --- | --- | --- |
| Abs | http://localhost:18080/math/abs/-5 | 5 |
| Factorial | http://localhost:18080/math/fact/5 | 120 |
| GCD | http://localhost:18080/math/gcd/18/24 | 6 |
| LCM | http://localhost:18080/math/lcm/4/6 | 12 |
| Palindrome Check | http://localhost:18080/math/palindrome/121 | true |
| Reverse Number | http://localhost:18080/math/reverse/432 | 234 |
| isEven | http://localhost:18080/math/even/10 | true |
| Sum of Digits | http://localhost:18080/math/sumdigits/1234 | 10 |

### ****VectorArray API****

| Feature | URL | Output |
| --- | --- | --- |
| Push | http://localhost:18080/vector/push/10 | Pushed |
| Show | http://localhost:18080/vector/show | 10 |
| Pop | http://localhost:18080/vector/pop | Last element removed |
| Size | http://localhost:18080/vector/size | 1 (or current size) |
| Find | http://localhost:18080/vector/find/10 | Found at index: X |

### ****Stack API****

| Feature | URL | Output |
| --- | --- | --- |
| Push | http://localhost:18080/stack/push/12 | Pushed |
| Pop | http://localhost:18080/stack/pop | Popped |
| Peek | http://localhost:18080/stack/peek | 12 |
| isEmpty | http://localhost:18080/stack/empty | false or true |

### ****Search API****

| Feature | URL | Output |
| --- | --- | --- |
| Linear Search | http://localhost:18080/search/linear/25 | Found at index X or Not found |

### ****Logging (Console Print)****

| Feature | URL | Output |
| --- | --- | --- |
| Log Value | http://localhost:18080/log/99 | "Check your console output!" (but value shows in terminal) |

### ****Greeting (with parameter)****

| Feature | URL | Output |
| --- | --- | --- |
| With name | http://localhost:18080/greet?name=Ali | Hello, Ali |
| No name | http://localhost:18080/greet |  |

Returns Hello, Guest (if no name parameter is provided)

For more endpoints, see the “API Testing Checklist” or README file.

#### Testing Options

The program supports **two execution modes**:

* **CLI Testing Mode:** Calls all data structure functions through the console
* **Crow API Mode:** Starts the server and allows HTTP-based testing

You can choose the mode at runtime.

## Conclusion and Learning

Through this internship task, I practiced:

* Advanced OOP in C++ using templates
* Manual memory management
* Functional decomposition using .h modules
* Algorithmic thinking and recursion
* STL-inspired custom implementations

I am confident this project demonstrates my ability to:

* Build clean, reusable modules
* Structure large-scale C++ systems
* Understand core CS fundamentals (DSA, File I/O)

This project enhanced my skills in:

* Advanced **template programming**
* Full understanding of **pointer-based memory management**
* Building STL-like structures manually (without using vector, list, stack)
* Recursive and iterative logic in real scenarios
* File operations with persistence using .txt files
* Testing and validating logic across a wide variety of edge cases

This experience enhanced my understanding of **data abstraction**, **template programming**, and the **modular design** approach that is vital in scalable software engineering projects.