# A9504 – Data Structures Laboratory

| Teaching and Learning Scheme | | | | Hours | Credits | Assessment Marks | | |
|---|---|---|---|---|---|---|---|---|
| CI | | LI | TW+SL | H | C | CIE | SEE | Total |
| L | T | P | SL | | | | | |
| 0 | 0 | 30 | 0 | 30 | 1 | 40 | 60 | 100 |

## Course Description

### Course Overview

This course enables to design and implement efficient C++ programs. Students learn object-oriented concepts such as classes, objects, and dynamic memory. It covers data structures including arrays, linked lists, stacks, queues, trees, and graphs. Hands-on exercises emphasize insertion, deletion, traversal, and searching operations. Students explore algorithms to improve performance and optimize data handling. Practical work with STL containers, iterators, and templates is included. Hashing and dictionary implementations demonstrate efficient data retrieval techniques. The course develops analytical thinking and systematic problem-solving skills. Students gain experience in building reliable, maintainable, and scalable software. By the end, students can apply programming concepts to solve real-world computational problems.

### Course Pre/Co-requisites

A9501 – Programming for Problem Solving
A9502 – Programming for Problem Solving Laboratory

### Relevant Sustainable Development Goals (SDGs)

SDG 4: Quality Education
SDG 9: Industry, Innovation, and Infrastructure

## Course Outcomes

After the completion of the course, the student will be able to:

A9504.1. Implement programs that efficiently manage and manipulate data using dynamic programming techniques in C++.
A9504.2. Apply various linked list techniques to perform insertion, deletion and traversal on given data.
A9504.3. Develop programs using linear data structures stack and queue to handle data processing tasks efficiently.
A9504.4. Implement nonlinear data structures to solve real time aplications.
A9504.5. Choose appropriate hashing and dictionary methods to efficiently store, retrieve, and manipulate data.

## Course Syllabus

### List of Experiments:

1. Implementing Classes, Objects, and Dynamic Memory Allocation
   a. Define a Student class with attributes: rollNumber, name, marks
   b. Implement a constructor to initialize objects and a destructor to display a message when an object is deleted
   c. Dynamically allocate an array of Student objects using new
   d. Input details of n students and display them
   e. Release allocated memory using delete

2. Function Overloading and Templates
   a. Implement two overloaded functions add() that can add:Two integers, Two floating-point numbers
   b. Define a function template swapValues() that swaps two variables of any type.
   c. Test swapValues() with integer, float, and string types
3. Using STL Containers and Iterators
   a. Create a vector of integers, insert elements, and display them using an iterator
   b. Create a list of strings, perform insertion and deletion, and traverse using an iterator
   c. Use a map to store StudentID -> Name pairs and display all elements
   d. Use a set to store unique integers and print them in sorted order
   e. Apply STL algorithms like sort(), find(), and count() on the containers
4. Implementing Singly Linked List (ADT) Operations
   a. Define a Node structure containing data and a next pointer Implement functions to:
   b. Insert a node at the beginning, end, and at a given position
   c. Delete a node from the beginning, end, and a specified position..
   d. Traverse the linked list and display all elements
   e. Search for an element in the list and return its position.
   f. Demonstrate all operations with sample inputs
5. Implementing Doubly Linked List (ADT) Operations
   a. Define a DoublyNode structure containing data, prev, and next pointers. Implement functions to:
   b. Insert a node at the beginning, end, and any position
   c. Delete a node from the beginning, end, and a specified position
   d. Traverse the list forward and backward
   e. Search for an element in the list.
   f. Demonstrate all operations with sample inputs
6. Circular Linked Lists (ADT) Operations
   a. Define a Node structure for circular singly linked lists with a next pointer pointing to the first node. Implement functions to:
   b. Insert a node at the beginning and end
   c. Delete a node from the beginning and end
   d. Traverse the list starting from any node and print all elements.
   e. Extend the above to circular doubly linked lists with prev and next pointers
   f. Demonstrate operations with sample inputs.
7. Implementing Stack (ADT) Using Array and Linked Lists Implement the following operations:
   a. push() – insert an element onto the stack
   b. pop() – remove the top element from the stack
   c. peek() – view the top element without removing it
   d. isEmpty() and isFull() – check stack status
8. Expression Conversion and Evaluation Using Stack
   a. Implement infix to postfix conversion using a stack
   b. Implement evaluation of postfix expressions using a stack
   c. Test with different arithmetic expressions (including parentheses)

9. Implementing Queues (ADT) Using Array and Linked Lists Implement a linear queue using:
   a. Array with enqueue() and dequeue() operations
   b. Linked list dynamically allocating nodes for each element
   c. Display or traverse list using array and linked list
10. Implementing Queues (ADT) Using Circular Linear List Implement a linear queue using:
    a. Array with enqueue() and dequeue() operations.
    b. Display or traverse list using array.
11. Binary Tree (ADT) Implementation and Traversals
    a. Define a Node structure with data, left, and right pointers. Implement functions to:
    b. Insert nodes into a binary tree
    c. Traverse the tree using:
    d. Inorder Traversal
    e. Preorder Traversal
    f. Postorder Traversal
    g. Demonstrate traversal operations with a sample binary tree
12. Binary Search Tree (BST) and AVL Tree Operations Implement BST operations:
    a. Insert a node
    b. Delete a node
    c. Search for a value
    d. Display the tree using Inorder traversal to verify correctness
13. AVL Tree Operations Implement BST operations:
    a. Insert a node
    b. Delete a node
    c. Search for a value
    d. Implement AVL tree insertion and deletion with rotations to maintain balance
    e. Display the tree using Inorder traversal to verify correctness
14. Hash Table Implementation and Collision Handling
    a. Implement a hash table using an array
    b. Design and apply a simple hash function
    c. Implement collision resolution techniques: Separate Chaining using linked lists Open Addressing: linear probing, quadratic probing, and double hashing
    d. Perform insertion, deletion, and searching operations.
    e. Demonstrate handling of collisions with sample inputs.
15. Dictionary Implementation Using Linear List and Skip List
    a. Implement a dictionary using a linear list: Perform insertion, deletion, and search operations.
    b. Implement a dictionary using a skip list for faster search: Include multiple levels with forward pointers. Implement insertion, deletion, and search.
    c. Compare the efficiency of linear list and skip list implementations with sample data order

**Laboratory Equipment/Software/Tools Required:**

1. Computer Systems (PCs) installed with Ubuntu OS (Open source/ Freeware)
2. GCC Compiler (Open source/ Freeware)

## Books and Materials

**Text Books:**

1. Horowitz, Ellis, Sartaj Sahni, and Dinesh Mehta. *Fundamentals of Data Structures in C++*, 2nd ed., Schaum's Outlines, Universities Press, 2019.
2. Malik D.S.*Data Structures and Algorithms in C++.* , 5th ed., Course Technology, 2010.

**Reference Books:**

1. Drozdek, Adam. *Data Structures and Algorithms in C++.* , 5th ed., Cengage Learning, 2025.
2. Dale, Nell, Chip Weems, and Tim Richards. *C++ Plus Data Structures.*, 6th ed., Jones & Bartlett Learning, 2018.