National University of Computer and Emerging Sciences

Laboratory Manual

for

Data Structures Lab

Course Instructor	Mr. Muhammad Naveed
Lab Instructor	Mr. Durraiz Waseem
Lab Demonstrator	Ms. Adeela Nasir
Section	BDS-3A
Date	Sept 2, 2025
Semester	Fall 2025

Department of Computer Science

FAST-NU, Lahore, Pakistan

Objectives:

In this lab, students will use Vectors and implement singly link basic operations like insertion and removal of data elements from it

Dynamic Vector Operations

1. **Basics of Vectors:**

Vectors in C++ are part of the Standard Template Library (STL) and provide a dynamic array-like structure that can grow or shrink in size as needed. They are implemented as a sequence container, allowing for efficient access, insertion, and deletion of elements. Unlike traditional arrays, vectors automatically manage their own memory, making them more flexible and easier to use in many scenarios.

2. **Key Operations on Vectors:**

- Insertion and Removal:

Vectors allow for insertion and removal of elements at both the end and at specific positions. The `push_back` function adds elements to the end of the vector, while `pop_back` removes the last element. For insertion at specific positions, `insert` is used, and `erase` removes elements from specified positions.

- Size and Capacity:

The `size()` method returns the number of elements currently stored in the vector, while `capacity()` provides the total number of elements that can be stored without reallocating. Vectors dynamically resize their capacity as needed when elements are added beyond the current capacity.

3. **Swapping Elements:**

Swapping elements within a vector involves exchanging values between two positions. This can be done manually using a temporary variable to hold one of the values during the swap. For instance, swapping the first and last elements of a vector requires access to both elements and exchanging their values.

4. **Practical Example:**

In practical scenarios, you may need to perform operations such as swapping elements, calculating the product of all elements, or finding subarrays with specific properties. For instance, in a vector of integers, you might swap elements to rearrange the vector, compute the product of elements to analyze their combined effect, or find subarrays that maximize the product for optimization problems.

5. **Avoiding Built-in Functions:**

While C++ provides several built-in functions and algorithms for common tasks (e.g., reversing, sorting, or finding unique elements), solving problems without these functions helps

deepen your understanding of underlying operations. Implementing these operations manually, such as swapping elements or calculating products, reinforces your grasp of vectors and their manipulations.

Question 1:

You are required to implement a dynamic array class, `DynamicArray`, that closely mimics the behavior of `vector`. Your class should be capable of performing various operations efficiently. Additionally, you will need to implement algorithms to manipulate and process the elements stored within this dynamic array. The tasks are outlined as follows:

Part 1: Implementing the `DynamicArray` Class

Class Definition: Implement a class `DynamicArray` that provides the following functionalities:

- Constructor: Initialize the dynamic array.
- Destructor: Properly manage memory when the object is destroyed.
- Copy Constructor and Assignment Operator: Implement deep copying to ensure that objects are copied correctly.
- `push_back(int value)`: Add an integer `value` to the end of the array. If the internal array is full, double the size of the array and copy the existing elements to the new array.
- `pop_back()`:*Remove the last element of the array.
- `size()`: Return the current number of elements in the array.
- `capacity()`: Return the current capacity of the internal array.
- `remove_duplicates()`: Remove all duplicate elements from the array without using any additional data structures. The order of the remaining elements should be preserved.
- `rotate_right(int k)`: Rotate the array to the right by `k` positions in-place

Part 2: Subarray Sum Finder

Function Definition: Implement a member function `find_subarrays_with_sum(int target)` within your `DynamicArray` class that finds and returns all subarrays that sum to a given target value. A subarray is a contiguous portion of the array.

- Input: `target` - The target sum to search for.
- Output: A vector of vectors, where each inner vector represents a subarray that sums to `target`.

Sample Main:

```
int main() {
    DynamicArray arr;
```

```
    arr.push_back(10);
    arr.push_back(20);
    arr.push_back(30);
    arr.push_back(20);
    arr.push_back(10);
    arr.push_back(50);

    cout << "Initial array: ";
    arr.print();
    arr.remove_duplicates();
    cout << "Array after removing duplicates: ";
    arr.print();

    arr.rotate_right(2);
   cout << "Array after rotating right by 2 positions: ";
    arr.print();

    cout << "Array size: " << arr.size() << endl;
    cout << "Array capacity: " << arr.capacity() << endl;


    int target = 30;
    auto subarrays = arr.find_subarrays_with_sum(target); cout
    << "Subarrays with sum " << target << ": " << endl; for
    (const auto& subarray : subarrays) {
         for (int num : subarray) {
             cout << num << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Question 2:

Given a vector of integers, perform the following tasks:

Swapping Elements:

- Swap the first and last elements of the vector. Swap the second and second-to-last elements, and so on, until you reach the middle of the vector.
- Calculate Product.After swapping, calculate the product of all the elements in the modified vector.
- Find Maximum Product Subarray. Find the contiguous subarray within the vector that has the maximum product and return this product.

Example:
Input Vector: [2, 3, -2, 4, 6]

Swapped Vector: [6, 4, -2, 3, 2]
Product of all elements: 6 * 4 * -2 * 3 * 2 = -288
Maximum Product Subarray: The subarray [6, 4] has the maximum product 24.

Linked List Overview:

A linked list is a way to store a collection of elements. Like an array these can be character or integers. Each element in a linked list is stored in the form of a node. A linked list is formed when many such nodes are linked together to form a chain.

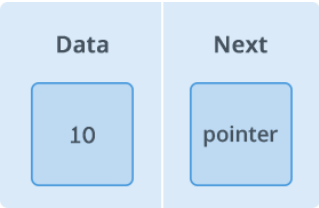

Figure 1. Node

It is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image:

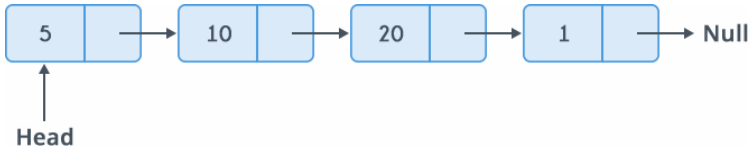

Figure 2. Linked list

- ✓ List is a collection of components, called nodes. Every node (except the last node) contains the address of the next node. Every node in a linked list has two components:
  - One to store the relevant information that is, **data**
  - One to store the address, called the link or **next**, of the next node in the list.
- ✓ The address of the first node in the list is stored in a separate location, called the **head** or first.
- ✓ The address of the last node in the list is stored in a separate location, called the **tail** or last.

Question 3:

Part-1: Implement a Struct '*Node'* that contains two data members: An int variable '*data'* and Node pointer '*next'*.

Part-2: Now implement a simple linked list class having two private data member Node pointer '*head'* and Node pointer '*tail'*. Please note that Node class is a nested class of linked list class. (Note that Struct Node is defined inside the List class)

Part-3: Now implement the following operations for linked list class:

a. Insert at start, void **insertAtHead**(T const element);
b. Insert at end, void **insertAtTail**(T const element);
c. Update the data element at specific position, void **update**(int Kth_position).