FAST School of Computing

Object Oriented Programming – Spring 2025

Software Engineering Department

LAB 03

Pointers in C++

Learning Outcomes

In this lab you are expected to learn the following:

- Pointers vs Reference variables
- Pass by Reference variable/ Reference to a pointer
- A function pointer as a parameter
- Dynamic Memory Allocation
- 2D and 3D Pointers

1. Pointers vs Reference Variables

- Pointers store the memory address of a variable. They need dereferencing (*) to access the actual value.
- Reference variables act as an alias for another variable. They do not store addresses and are directly used like normal variables.

🔹 **Real-Life Example:**

- Pointer: Like a GPS location that tells you where a house is.
- Reference Variable: Like a nickname for a person—you use the nickname, but it refers to the same person.

2. Reference to a pointer / Pass by Reference Variable

- Pass by Reference: A function directly modifies the original variable without making a copy**.**
- A "reference to a pointer" means that you are passing a pointer to a function in such a way that the function can modify the pointer itself, not just the data it points to. This is done by using a reference to the pointer.

🔹 **Real-Life Example:**

- Pointer Return: Imagine calling a hotel, and they give you the address of an available room instead of describing it.
- Pass by Reference: Giving someone your actual car keys instead of a duplicate, so they use your car directly.

3. A Function Pointer as a Parameter

A function pointer is a pointer that stores the address of a function. It allows a function to be passed as an argument to another function.

🔹 **Real-Life Example:**

- A TV remote (function pointer) can switch channels (functions). The remote doesn't store channels; it just calls the function to change them.

4. Dynamic Memory Allocation

Allocating memory at runtime (instead of compile time) using malloc, calloc, new, etc.

🔹 **Real-Life Example:**

- A dynamic parking lot that expands when more cars arrive and shrinks when they leave, instead of a fixed number of parking spots.

5. 2D and 3D Pointers

- 2D Pointers: Used to store arrays of pointers (like a table) in DMA.
- 3D Pointers: Used to store a set of 2D arrays (like a cube) in DMA.

- **Real-Life Example:**
  - 2D Pointer: A spreadsheet with rows and columns, where each cell points to data.
  - Used in dynamic tables, and gaming grids where the number of rows/columns is unknown at compile time.
  - 3D Pointer: A Rubik’s cube where each small cube represents a pointer to values.
  - Used in image processing, medical imaging (3D MRI scans), and 3D graphics where we need to store depth information dynamically.

Class Activity 1:

A library management system requires a program to handle book titles dynamically. The number of books and the lengths of their titles are unknown at compile time, so memory must be allocated dynamically. Write a program using **1D pointers and Dynamic Memory Allocation (DMA)** to:

1. **Find and display all book titles that contain the word "Programming".**
2. **Replace all spaces in each title with underscores ('_').**

Class Activity 2:

Write a C++ program that checks whether a given 2D dynamic matrix is an identity matrix (i.e., a square matrix where all diagonal elements are 1, and all non-diagonal elements are 0).

Class Activity 4:

Imagine you have a **friend (pointer)** who lives in a house. You give them an address (the pointer to the house). Normally, if you change the house number (or the data they live in), it affects their location. But **if you want to change the house entirely** (i.e., move them to a new address), you need to **change the address itself**.

- **Regular Pointer**: You're telling your friend to move from one room to another inside the same house (changing the data the pointer points to).
- **Reference to a Pointer**: You are changing the **address itself**, meaning you move your friend to an entirely new house (changing where the pointer itself points). e.g:
- Before: friend = "House1"
- After: friend = "House2"

Class Activity 4:

In a company, an interactive calculator application is designed to perform various arithmetic operations like addition and multiplication based on user input. The company wants to give the user the flexibility to choose the operation dynamically. Instead of hardcoding the operations, the calculator will use **function pointers** to call different functions at runtime.

- The program should accept two numbers from the user.
- The user should choose whether to perform an addition or multiplication.
- The function to perform the chosen operation should be called dynamically via function pointers.

Class Activity 3:

- Write a function reverseString that takes a string as input and reverses it using pointers. The function should modify the original string.

Basic Problems (1-4)

Problem 1:

Write a program that declares a 2D dynamic matrix in the main function. Pass this matrix to a function and check whether it is a symmetric matrix or not (i.e., matrix[i][j] == matrix[j][i]). If it is symmetric, return "Symmetric"; otherwise, return "Not Symmetric".

Problem 2:
A text processing application requires dynamic handling of sentences. The number of sentences and their lengths are not known at compile time, so you must allocate memory dynamically.

How to Implement It:

You are given an array of strings where each string represents a sentence then write a program using **1D pointers** and **DMA** to:

1. Count the total number of words in all sentences.
2. Find the longest sentence.
3. Sort the sentences in alphabetical order.

Problem 3:

A company assigns the manager to an office, represented by a pointer. The company wants to dynamically relocate the manager to a new office while allowing flexibility in updating the office contents. Should the company only modify the interior of the office (content), or relocate the manager to a different office entirely (address)?

Problem 4:

Case Study 1: Online Food Ordering System

An online food app dynamically applies different discount rules based on the user's order history.

Pass-by-reference: Updates order price instantly without copying data.

Function pointers: Switch between different discount strategies (e.g., new user, bulk order, festival sale).

1: A new user gets a flat 20% discount.

2: A bulk order gets a special price cut.

3: A holiday sale applies an extra discount.

Intermediate Problems (1-2)

1: Dynamic Word Frequency Counter
A text analysis tool requires dynamic handling of word frequencies. The number of words and their lengths are not known at compile time, so you must allocate memory dynamically.

How to Implement It:

Write a program using **2D pointers** and **DMA** to:

1. Count the frequency of each word in the paragraph.
2. Store the words and their frequencies in a dynamically allocated 2D array.
3. Sort the words in descending order of frequency.

2: Dynamic Movie Review System

Scenario:
A movie review platform requires dynamic handling of reviews. The number of movies, reviews, and sentences are not known at compile time, so you must allocate memory dynamically.

How to Implement It:

You are tasked with creating a system to store movie reviews. Each movie can have multiple reviews, and each review can have multiple sentences. Write a program using **3D pointers** and **DMA** to:

1. Dynamically allocate memory for the review system.
2. Add reviews for movies dynamically.
3. Find the movie with the most reviews.

Hard Level Problems (Case Study 1 - Case Study 2)

Case Study 1: 2D Pointer - Dynamic Seating Arrangement in a Cinema

This case focuses on a cinema hall with a dynamic number of rows and columns for seats. The system allows customers to book, cancel, and view seat availability in real-time. Using a 2D pointer (int** seats), the seating layout is created and modified dynamically based on user input. Pointer arithmetic is used to efficiently navigate and update the seating arrangement. The system ensures memory is allocated and freed properly to prevent memory leaks, providing an efficient way to manage seating reservations.

Case Study 2: 3D Pointer - Weather Data Storage

In this case study, a weather forecasting system collects and manages temperature readings for multiple cities over several days at different times of the day (morning, afternoon, evening). The data is efficiently stored using a **3D pointer (float*** weatherData)**, enabling dynamic memory allocation to accommodate varying input sizes. The system leverages **pointer-based 3D array traversal** for efficient data access and retrieval. Users can dynamically store, retrieve, and display temperature readings using pointer arithmetic, ensuring flexible and optimized weather data management.