CS224 Object Oriented Programming and Design Methodologies Lab Manual

Lab 02 - Arrays and Switch Statements

Department of Computer Science

Dhanani School of Science and Engineering

Habib University

Fall 2025

Copyright @ 2025 Habib University

Contents

1 Introduction

1.1 Guidelines

1. Use of AI is strictly prohibited. This is not limited to the use of AI tools for code generation, debugging, or any form of assistance. If detected, it will result in immediate failure of the lab, student will be awarded 0 marks, reported to the academic integrity board and appropriate disciplinary action will be taken.
2. Absence in lab regardless of the submission status will result in 0 marks.
3. All assignments and lab work must be submitted by the specified deadline on Canvas. Late submissions will not be accepted.

1.2 Objectives

Following are the lab objectives of this lab:

- To get familiar with switch statements, their uses and fall through cases
- To get familiar with arrays in C++.
- To know how to apply operations on arrays in C++.

1.3 Directory Structure

Labs will have following directory structure:

```
lab
├── manual
│   └── lab.pdf
├── src
│   ├── *.hpp
│   └── *.cpp
├── tests (optional)
│   └── *.cpp
└── makefile (optional)
```

`manual` will contain the lab manual pdf. `src` will contain the source code files, and `tests` (if present) will contain the test files. `makefile` (if present) will contain the makefile for testing and running.

1.4 Lab 01 Recap

Before we start this lab, Let's Recap Lab 01. Have a look at the C++ program given below.

```cpp
// A simple C++ program to teach basics: variable declaration, printing, input, comments, and conditions
#include <iostream>  // Required for input and output
using namespace std; // To avoid writing std:: repeatedly

int main()
```

```
{
    // --- COMMENTS ---
    // Single-line comment starts with //
    /* Multi-line comments look like this
       They can span multiple lines */

    // --- PRINTING ---
    cout << "Hello, World!" << endl; // Prints text to the screen

    // --- VARIABLES ---
    int age = 20;          // Integer variable
    double height = 5.9; // Decimal number
    char grade = 'A';     // Single character

    // Print variables
    cout << "Age: " << age << endl;
    cout << "Height: " << height << endl;
    cout << "Grade: " << grade << endl;

    // --- TAKING INPUT ---
    int userAge;

    cout << "Enter your age: ";
    cin >> userAge; // Takes integer input

    cout << "\nYou are " << userAge << " years old." << endl;

    // --- IF / ELSE STATEMENTS ---
    if (userAge < 18)
    {
        cout << "You are a minor." << endl;
    }
    else if (userAge >= 18 && userAge < 60)
    {
        cout << "You are an adult." << endl;
    }
    else
    {
        cout << "You are a senior citizen." << endl;
    }

    return 0; // End of program
}
```

Listing 1.1: lab01_recap.cpp

1.5 Switch statements

A switch statement in C++, just like if-then-else statements, is a control-flow structure that allows you to choose one out of multiple possible options based on the value of a single variable or expression.

Instead of writing multiple if-else statements, you can use switch for cleaner, more readable code when dealing with discrete values (like numbers or characters).

The following code block shows the general syntax of a switch statement.

```
switch(expression)
{
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

How does this work:

- expression is evaluated once.
- Control jumps to the matching case label.
- Code under that case executes until a break statement is found.
- If no case matches, default (if present) executes.

Let's look at an example.

```cpp
#include <iostream>
using namespace std;

int main() {
    char grade;
    cout << "Enter your grade (A, B, C, D, F): ";
    cin >> grade;

    switch(grade) {
        case 'A':
            cout << "Excellent!" << endl;
            break;
        case 'B':
            cout << "Good job!" << endl;
            break;
        case 'C':
            cout << "Fair performance." << endl;
            break;
        case 'D':
            cout << "Needs improvement." << endl;
            break;
        case 'F':
            cout << "Failed." << endl;
            break;
        default:
            cout << "Invalid grade." << endl;
    }

    return 0;
}
```

Listing 1.2: `switch_tutorial.cpp`

Switch statements work with int, char, enum or constant expressions, not with floats or strings. Each case must be unique and *break* should follow it otherwise it will fall through.

Let's look at one example of that.

```cpp
#include <iostream>
using namespace std;

int main() {
    int option;
    cout << "Enter an option (1-3): ";
    cin >> option;

    switch(option) {
        case 1:
        case 2:
            cout << "Option 1 or 2 selected." << endl;
            break;
        case 3:
            cout << "Option 3 selected." << endl;
            break;
        default:
            cout << "Invalid option." << endl;
    }

    return 0;
}
```

Listing 1.3: `fall_through_example.cpp`

1.6 Arrays

An array in C++ is a collection of fixed-size, sequential elements of the same data type, stored in contiguous memory locations. Think of it like a list of items placed one after another in memory. For example:

```
int marks[5] = {90, 85, 70, 88, 76};
```

Here:

- marks is an array of size 5
- It can store 5 integers
- Indices go from 0 to 4

The following code block gives the general syntax for initializing an array:

```
data_type array_name[n] = {element_0, element_1, ... element_n-1};
```

Here, n is the array size and element_0, element_1 are its elements. Please note that putting elements is optional. Arrays can be declared like this as well.

```
data_type array_name[n];
```

Let's look at an example:

```
// File: array_example.cpp
#include <iostream>
using namespace std;

int main() {
    // Example of 1D array in C++ without using loops
    int marks[5] = {90, 85, 70, 88, 76};

    // Accessing elements directly
    cout << "First mark: " << marks[0] << endl;
    cout << "Third mark: " << marks[2] << endl;

    // Modifying an element
    marks[2] = 75;
    cout << "Updated third mark: " << marks[2] << endl;

    return 0;
}
```

Listing 1.4: `array_example.cpp`

2 Exercises

2.1 Simple calculator - Using Switch [25 Points]

You are to implement a simple calculator in C++ that gives out the result of addition, subtraction, multiplication, division and power of two integers depending on the operation that is asked to be performed.

For example, *int_one* = 4, *int_two* = 5, *op* = '+'. The result should then be $4 + 5 = 9$. If an operator other than the four above is input then you should output the following: "Error! The operator is not correct"

Constraints

- All inputs are positive integers. Do input validations.

Note: You must solve this question using the switch statement.

Input Format For Custom Testing:
The first two lines contain the integers upon which you must perform the operation. The third line contains a character denoting which operation to perform (+, -, *, /,ˆ).
Sample Case 0

```
>>> Enter first number: 4
>>> Enter second number: 3
>>> Enter operation: ^
4 ^ 3 = 64
```

Sample Case 1

```
>>> Enter first number: 13
>>> Enter second number: 3
>>> Enter operation: /
13 / 3 = 4
```

Explanation: 13/3 = 4.333 = 4 (integer division in c++).

2.2 Zaika-e-Pizza Ordering System [25 Points]

Zaika-e-Pizza offers a variety of pizza flavors and sizes with different prices, as well as special weekly deals. You are required to write a C++ program using the switch statement to calculate the final bill for a customer.

Pizza Menu:
The following flavors and sizes are available:

Flavors (input as a single character):

- c = Chicken
- b = Beef
- v = Vegetarian
- p = Plain
- a = Assorted

Sizes (input as a single character):

- s = Small
- m = Medium
- l = Large

Price in PKR:

Flavor	Small (s)	Medium (m)	Large (l)
Chicken (c)	800	1000	1200
Beef (b)	900	1100	1300
Vegetarian (v)	700	900	1100
Plain (p)	600	800	1000
Assorted (a)	1000	1200	1400

Weekly Deals
Zaika-e-Pizza also offers daily deals:

1. **Monday:** Buy 1 Medium Chicken Pizza, get 1 Small Chicken Pizza free
2. **Tuesday:** Buy 1 Large Pizza (any flavor), get 1 free (same flavor and size)
3. **Wednesday:** Buy 2 Small Pizzas (any flavors), get 20% off
4. **Thursday:** Buy 1 Beef Pizza of any size, get 1 Small Plain Pizza free
5. **Friday:** Buy 1 Large Assorted Pizza, get 1 Medium Vegetarian Pizza free
6. **Saturday:** No deal
7. **Sunday:** Buy 1 Medium Pizza (any flavor), get 1 Small Vegetarian Pizza free

Program Requirements:
Your program must:

1. Ask the user to enter the index of the day of the week.
2. Ask the user to enter an order consisting of two pizzas. For each pizza, the user provides:
   - Flavor code (c, b, v, p, a)
   - Size code (s, m, l)
3. Use the `switch` statement to determine the price of each pizza.
4. Apply the deal for the given day (if applicable).
5. Display the final bill, clearly mentioning any free pizzas or discounts applied.

Input Format For Custom Testing:
There are five lines of input. The first line contains the index of the day of the week. 1 for Monday, 2 for Tuesday and so on. Hence, 7 for Sunday. Second and third line contain pizza flavor and size of first pizza and fourth and fifth line contain pizza flavor and size of second pizza respectively. For both flavor and size, user inputs just a character.
Sample Case 0

```
>>> Enter day of week: 1
>>> Enter first pizza flavor (c/b/v/p/a): c
>>> Enter first pizza size (s/m/l): m
>>> Enter second pizza flavor (c/b/v/p/a): b
>>> Enter second pizza size (s/m/l): l
You ordered:
- Medium Chicken Pizza: 1000 PKR
- Large Beef Pizza: 1300 PKR
Monday Deal Applied: Buy 1 Medium Chicken Pizza, get 1 Small Chicken
Pizza free.
Final Bill: 2300 PKR
```

Sample Case 1

```
>>> Enter day of week: 2
>>> Enter first pizza flavor (c/b/v/p/a): v
>>> Enter first pizza size (s/m/l): l
>>> Enter second pizza flavor (c/b/v/p/a): v
>>> Enter second pizza size (s/m/l): l
You ordered:
- Large Vegetarian Pizza: 1100 PKR
- Large Vegetarian Pizza: 1100 PKR
Tuesday Deal Applied: Buy 1 Large Pizza (any flavor), get 1 free (same
flavor and size)
Final Bill: 1100 PKR.
```

2.3 Vector operations [25 Points]

In mathematics a vector is an element of a vector space. The set $\mathbb{R}$ denotes the set of real numbers and the cartesian product $\mathbb{R}^n = \underbrace{\mathbb{R} \times \mathbb{R} \times \cdots \times \mathbb{R}}_{n \text{ times}} = \{(a_1, a_2, \ldots, a_n) | \forall i \in \{1, 2, \ldots n\},\ a_i \in \mathbb{R}\}$. As $\mathbb{R}^n$ forms a vector space under the field $\mathbb{R}$ with the binary operation $+$ (vector addition) and $\cdot$ (scalar multiplication).

Vector addition $+$ is defined as: for vectors $x = (a_1, a_2, dots, a_n)$ and $y = (b_1, b_2, dots, b_n)$, $x + y = (a_1 + b_1, a_2 + b_2, dots, a_n + b_n)$ (subtraction is defined similarly). Vectors $x = (a_1, a_2, dots, a_n)$ and $y = (b_1, b_2, dots, b_n)$ are equal if and only if $\forall i \in \{1, 2, \ldots n\},\ a_i = b_i$.

We are going to write a program to perform arithmetic on vectors, specifically to add and subtract them. We will store a vector using an integer array.

You have to write the following functions.

- *print_vector*: prints the contents of the vector passed to it.
- *input_vector*: populates the vector passed to it with values input from the console.
- *add_vectors*: takes 3 vectors as parameters and stores the sum of the first 2 vectors in the third one
- *subtract_vectors*: takes 3 vectors as parameters and stores the difference of the first 2 vectors in the third one
- *compare_vectors*: takes 2 vectors as parameters and returns true if they are equal and false otherwise.

You may pass any additional parameters to your functions as needed.

Input Format For Custom Testing:
The first line contains an integer, n, denoting the dimension of the 2 vectors that follow. The next line contains n space separated integers which represent the first vector. The next line contains n space separated integers which represent the second vector. The next line contains a single character which is one of: + - =. Depending on the character, you have to output the sum or the difference of the vectors or if the vectors are equal.

Sample Case 0

```
>>> 3
>>> 6 5 4
>>> 1 2 3
>>> +
7 7 7
```

Explanation: The sum of the vectors [6 5 4] and [1 2 3] is [7 7 7].

Sample Case 1

```
>>> 3
>>> 6 5 4
>>> 1 2 3
>>> -
5 3 1
```

Explanation: The difference of the vectors [6 5 4] and [1 2 3] is [5 3 1].

2.4 Courier Company Price Analysis using Simulated 2-D Arrays [25 Points]

A new courier company wants to analyze the pricing strategies of its competitors. Courier services typically offer three types of delivery options:

- U: Urgent — fastest but most expensive.
- N: Normal — balanced in cost and time.
- E: Economy — cheapest but slowest.

For simplicity, we will consider only three cargo categories:

1. Documents (any weight)
2. Parcels < 5kg
3. Parcels ≥ 5kg

Thus, there are a total of 3 categories of cargo and 3 classes of service, giving a 3×3 price table for each company.

The three companies under study are:

1. North Heights Courier Company
2. Central Plains Courier Company
3. Southern Coastal Courier Company

Example Price Tables (in PKR):

North Heights Courier Company

Service	**Documents**	**Parcel <5kg**	**Parcel ≥5kg**
Urgent	500	900	1600
Normal	300	600	1100
Economy	200	450	800

Central Plains Courier Company

Service	**Documents**	**Parcel <5kg**	**Parcel ≥5kg**
Urgent	550	950	1700
Normal	320	620	1150
Economy	220	470	850

Southern Coastal Courier Company

Service	**Documents**	**Parcel <5kg**	**Parcel ≥5kg**
Urgent	600	1000	1800
Normal	350	650	1200
Economy	250	500	900

Write a C++ program that:

1. Stores the above price tables in **1-D arrays** while simulating 2-D indexing. For example, you can use the formula:

   $$\text{index} = \text{row} \times \text{number\_of\_columns} + \text{column}$$

   to access elements as if they were in a 2-D array.

2. Calculates, for each service and cargo category, the following statistics across the three companies:
   - Average price
   - Maximum price
   - Minimum price

3. Displays the results in a formatted comparative table.

Input Format For Custom Testing:
The first line contains delivery option (U, N, E) and second line contains cargo category (1, 2, 3).

Sample Case 0

```
>>> Service: N
>>> Cargo: 2
Category: Normal Service, Parcel < 5kg
Prices: 600, 620, 650
Average Price: 623.33 PKR
Maximum Price: 650 PKR
Minimum Price: 600 PKR
```

Sample Case 1

```
>>> Service: U
>>> Cargo: 1
Category: Urgent Service, Documents
Prices: 500, 550, 600
Average Price: 550 PKR
Maximum Price: 600 PKR
Minimum Price: 500 PKR
```

Sample Case 2

```
>>> Service: E
>>> Cargo: 3
Category: Economy Service, Parcel  5kg
Prices: 800, 850, 900
Average Price: 850 PKR
Maximum Price: 900 PKR
Minimum Price: 800 PKR
```