Contents

1 Function Overloading in C++

Function overloading is a feature in C++ that allows multiple functions to have the same name but with different parameters (number or type of arguments). The compiler determines which function to call based on the function signature.

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

// Function with one integer parameter
void display(int num) {
    cout << "Integer: " << num << endl;
}

// Function with one double parameter
void display(double num) {
    cout << "Double: " << num << endl;
}

// Function with two parameters
void display(int num1, int num2) {
    cout << "Two Integers: " << num1 << " and " << num2 << endl;
}

int main() {
    display(10);         // Calls display(int)
    display(5.5);        // Calls display(double)
    display(3, 7);       // Calls display(int, int)
    return 0;
}
```

1.1 Rules for Function Overloading

1. Functions must have the same name but different parameter lists.
2. Overloaded functions can differ by:
    - Number of parameters
    - Type of parameters
    - Order of parameters (if different types)
3. Return type alone cannot differentiate functions. Because return type is NOT part of the function signature when resolving function calls.

1.1.1 Example of Invalid Overloading (Compilation Error)

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

// Function returning int
int func(int x) {
    return x;
}
```

```
// Function with same parameter list but different return type (Error!)
double func(int x) {
    return x * 1.5;
}

int main() {
    int a = func(10);  // Compiler confusion: Which 'func' should be called?
    return 0;

}
```

Why Does This Fail?

- The compiler sees two functions with the same name and identical parameter lsit:
  - `int func(int x)`
  - `double func(int x)`
- Since C++ only looks at the function name and parameters to resolve calls, it does not consider the return type.
- When `func(10);` is called, the compiler can't decide which one to use, causing an error.

How to Fix This?

To properly overload a function, you must change the parameter list.

```
int func(int x) { return x; }
double func(double x) { return x * 1.5; } // Now it's valid
```

Now, `func(10)` calls `func(int)`, and `func(10.5)` calls `func(double)`, avoiding confusion.

2 TP – Function Overloading

TP 1) Write a C++ program to simulate a router using function overloading, but without using classes or structures. The router should handle packet forwarding based on different types of input parameters, such as:

i. An integer IP address
ii. A string IP address
iii. A source and destination IP address
iv. A source IP, destination IP, and Protocol (e.g., TCP/UDP)

Implement different overloaded functions named `routePacket()` to handle these cases. In the `main()` function, call these functions with appropriate arguments and display the corresponding routing behavior.

TP 2) Write a C++ program to simulate a football team on the ground using function overloading only (without using classes or structures). The program should allow assigning players to different positions and support the following features.

i. **Positions** - Include roles like Attacker, Midfielder, Defender, Goalkeeper, Winger, Striker, and Sweeper.

ii. **Overloaded Functions** – Implement different variations of `assignPosition()` to handle:

- Assigning players by name and position.
- Assigning players by jersey number and position.
- Assigning players with specific team sides (left, right, center).
- Assigning a captain.
- Assigning a substitute.

iii. Randomized Team Assignment – Implement a function that assigns a random positions to players.

iv. User Input for Player Assignments – Allow the user to enter a player's name, number, and position dynamically.

v. Better Output Formatting – Display the team formation and present the assigned players in a structured format.

Your program should include a main() function where multiple function calls demonstrate all the above features.

Expected Output

```
Ronaldo is playing as Attacker.
Player #10 is playing as Midfielder.
Messi (Jersey #30) is playing as Midfielder.
Van Dijk (Jersey #4) is playing as Defender on the left side.
Alisson (Jersey #1) is playing as Goalkeeper.
Kane (Jersey #9) is playing as Striker (Captain).
Hazard (Jersey #7) is playing as Winger (Substitute).

Team is playing in a 4-4-2 formation.
Random Player Assignments:
De Bruyne (Jersey #17) is playing as Midfielder.
Mbappe (Jersey #7) is playing as Striker.
Salah (Jersey #11) is playing as Sweeper.

Enter player name: Neymar
Enter player number: 10
Enter position: Forward
Neymar (Jersey #10) is playing as Forward.
```

TP 3) Write a C++ program that simulates a car manufacturing system using function overloading only (without using classes or structures). The system should allow the creation of different car models with various specifications, including brand, model, engine type, transmission type, custom features, and price estimation.

Requirements:

i. **Basic Car Creation**
   - Implement a function to create a car using only its brand and model.

ii. **Car with Engine Type**
   - Overload the function to allow specifying the engine type (e.g., Petrol, Diesel, Electric, Hybrid)

iii. **Car with Transmission Type**
   - Extend the overloaded function to support selecting a transmission type (Manual or Automatic)

iv. **Car with Custom Features**
   - Allow users to specify additional car attributes, including color, number of doors, and seating capacity.

v. **Car with Price Estimation**
   - Implement a function that calculates and displays an estimated price based on selected features.
   - The price should vary based on factors like engine type, transmission, and additional features.

vi. **User Input for Car customization**
   - Allow users to input car details dynamically, enabling them to create car with preferred specifications.

Expected Output

The program should display detailed information about each manufactured car, including its brand, model, specifications, and estimated price. Additionally, the program should allow users to enter car details manually and generate customized car models.

```
Manufacturing Car: Tesla Model S
Manufacturing Car: Toyota Camry with Hybrid engine.
Manufacturing Car: Ford F-150 with Diesel engine and Automatic
transmission.
Manufacturing Car: BMW X5 with Petrol engine, Manual transmission,
Black color, 5 doors, and seating for 5 people.
Manufacturing Car: Mercedes E-Class with Electric engine, Automatic
transmission, White color, 4 doors, and seating for 5 people.
Estimated Price: $55000

Randomly Generated Car:
Manufacturing Car: Toyota F-150 with Petrol engine, Manual
transmission, Red color, 4 doors, and seating for 5 people.
Estimated Price: $30000

Enter Car Brand: Nissan
Enter Car Model: Altima
```

```
Enter Engine Type (Petrol/Diesel/Electric/Hybrid): Hybrid
Enter Transmission (Manual/Automatic): Automatic
Enter Color: Gray
Enter Number of Doors: 4
Enter Seating Capacity: 5
Enter Base Price: $28000
Manufacturing Car: Nissan Altima with Hybrid engine, Automatic
transmission, Gray color, 4 doors, and seating for 5 people.
Estimated Price: $30000
```

TP 4) Write a C++ program building a social media platform that allows users to create posts. Users can create posts with varying levels of detail:

- Text-only posts: A simple post containing only text.
- Text and image posts: A post containing both text and an image.
- Text, image, and hashtags posts: A post containing text, an image, and a list of hashtags.

Implement the code that handles these different types of posts using function overloading (without Classes or Structures). The code should allow users to create posts with the appropriate details and display the post information in a structured format.

TP 5) Write a C++ program that implements an Access Control List (ACL) simulation using function overloading only (without classes or structures). This implementation includes:

- Standard ACL (filters based on source IP)
- Extended ACL (filters based on source IP, destination IP, and protocol)
- Wildcard mask support (for subnet-based filtering)
- User Input (Users can enter their own ACL rules dynamically)

Expected Output

```
Enter number of ACL rules: 2
Enter ACL Rule 1:
permit 192.168.1.0 0.0.0.255 10.0.0.0 0.0.0.255 tcp
Enter ACL Rule 2:
deny 192.168.1.10 0.0.0.0 any any any
Enter source IP to test: 192.168.1.50
Enter Destination IP to test: 10.0.0.5
Enter Protocol (tcp/udp/any): tcp
ACCESS GRANTED
```

TP 6) Write a C++ code using function overloading only (without classes or structures) to build a system for aircraft manufacturing and maintenance scheduling. The system should handle different types of aircraft configurations, and maintenance tasks.

- **Basic configuration:** Only the aircraft model is required.
- **Configuration with engine type:** The aircraft model and engine type are required.
- **Full configuration:** The aircraft model, engine type, and seating capacity are required.
- **Basic maintenance:** only the aircraft ID and maintenance type are required.

- **Maintenance with duration:** The aircraft ID, maintenance type, and duration (in hours) are required.
- **Maintenance with technician details:** The aircraft ID, maintenance type, and technician name are required.

Expected Output

```
Manufactured basic aircraft. Model: Airbus A380
Manufactured aircraft. Model: Airbus A380, Engine Type: Rolls-Royce
Trent 900
Manufactured aircraft. Model: Airbus A380, Engine Type: Rolls-Royce
Trent 900, Seating Capacity: 850

Scheduled maintenance for Aircraft N12345. Type: Routine Check
Scheduled maintenance for Aircraft N12345. Type: Engine Overhaul,
Duration: 8 hours
Scheduled maintenance for Aircraft N12345. Type: Avionics Repair,
Technician: John Doe
```

TP 7) Write a C++ code for Airline Ticket Booking and Flight Status Notification System using function overloading (without classes or structures). The system should handle different types of bookings and flight status notifications.

- **One-way ticket**: Only the departure flight details are required.
- **Round-trip ticket**: Both departure and return flight details are required.
- **Multi-city ticket:** Multiple flight segments are required.
- **Basic notification**: Only the flight number and status are required.
- **Notification with delay time**: The flight number, status, and delay time are required.
- **Notification with gate change**: The flight number , status, and new gate numbers are required.

Expected Output

```
Booked one-way ticket. Departure Flight: AA123
Booked round-trip ticket. Departure Flight: AA123, Return Flight:
AA456
Booked multi-city ticket. Flights: AA123 -> AA789 -> AA456

Flight AA123 status: On Time
Flight AA123 status: Delayed, Delayed by: 30 minutes
Flight AA123 status: Boarding, New Gate: Gate B12
```