**Slip 10: Sample Solutions and Explanations**

**Q1. Class Template to Compare Numbers of Different Types**

**Approach**

* Use a class template to compare two values of any type (int, float, etc.).
* Provide a method to display which value is greater or if they are equal.
* Demonstrate with different data types in main.

**Code**

#include <iostream>  
using namespace std;  
  
// [Template Class for Comparison]  
template <typename T>  
class Compare {  
 T a, b;  
public:  
 Compare(T x, T y) : a(x), b(y) {}  
 void displayGreater() {  
 if(a > b)  
 cout << a << " is greater than " << b << endl;  
 else if(b > a)  
 cout << b << " is greater than " << a << endl;  
 else  
 cout << a << " and " << b << " are equal." << endl;  
 }  
};  
  
int main() {  
 Compare<int> cmp1(5, 10);  
 cmp1.displayGreater();  
 Compare<float> cmp2(2.5, 1.5);  
 cmp2.displayGreater();  
 return 0;  
}

**Explanation**

* The Compare template class works for any data type that supports comparison operators.
* The displayGreater method compares the two values and prints the result.
* Demonstrated with both int and float types in main.

**Syntax Definitions**

* **template <typename T>**: Allows the class to work with any data type specified at instantiation.
* **Class Template**: A blueprint for creating classes that operate with generic types.

**Q2. Employee and Manager Inherited Class**

**Approach**

* Define a base class Employee with name, code, and designation.
* Derive a Manager class from Employee that adds years of experience and salary.
* Accept and display all details for a manager.

**Code**

#include <iostream>  
using namespace std;  
  
// [Employee Base Class]  
class Employee {  
protected:  
 string name, designation;  
 int code;  
public:  
 void accept() {  
 cout << "Name: "; cin >> name;  
 cout << "Code: "; cin >> code;  
 cout << "Designation: "; cin >> designation;  
 }  
 void display() { cout << name << " " << code << " " << designation << endl; }  
};  
  
// [Manager Derived Class]  
class Manager : public Employee {  
 int exp;  
 double salary;  
public:  
 void accept() {  
 Employee::accept();  
 cout << "Years Experience: "; cin >> exp;  
 cout << "Salary: "; cin >> salary;  
 }  
 void display() {  
 Employee::display();  
 cout << "Experience: " << exp << " Salary: " << salary << endl;  
 }  
};  
  
int main() {  
 Manager m;  
 m.accept();  
 m.display();  
 return 0;  
}

**Explanation**

* The Manager class inherits from Employee and adds its own fields.
* The accept and display methods in Manager call the base class methods to handle inherited data.
* Demonstrates inheritance and method overriding.

**Syntax Definitions**

* **protected**: Members are accessible in the class and its derived classes.
* **public inheritance**: Derived class inherits all public and protected members of the base class.

**Q3. Employee Payroll System (Inheritance Case Study)**

**Approach**

* Use a base class Employee for common data (name, id).
* Derive FullTimeEmployee and PartTimeEmployee classes for specific attributes.
* Accept and display details for both types of employees.

**Code**

#include <iostream>  
using namespace std;  
  
// [Employee Base Class]  
class Employee {  
protected:  
 string name; int id;  
public:  
 void accept() { cout << "Name: "; cin >> name; cout << "ID: "; cin >> id; }  
 void display() { cout << name << " " << id << endl; }  
};  
  
// [FullTimeEmployee Derived Class]  
class FullTimeEmployee : public Employee {  
 double salary;  
public:  
 void accept() { Employee::accept(); cout << "Salary: "; cin >> salary; }  
 void display() { Employee::display(); cout << "FullTime Salary: " << salary << endl; }  
};  
  
// [PartTimeEmployee Derived Class]  
class PartTimeEmployee : public Employee {  
 double hourlyRate;  
public:  
 void accept() { Employee::accept(); cout << "Hourly Rate: "; cin >> hourlyRate; }  
 void display() { Employee::display(); cout << "PartTime Rate: " << hourlyRate << endl; }  
};  
  
int main() {  
 FullTimeEmployee f; PartTimeEmployee p;  
 f.accept(); f.display();  
 p.accept(); p.display();  
 return 0;  
}

**Explanation**

* The base class Employee holds common data.
* FullTimeEmployee and PartTimeEmployee inherit from Employee and add their own fields.
* Each derived class calls the base class methods for shared data.

**Syntax Definitions**

* **Inheritance**: Mechanism by which one class acquires the properties and behaviors of another class.
* **Method Overriding**: Redefining a base class method in a derived class.