**Slip 4: Sample Solutions and Explanations**

**Q1. Template Program to Find Area of Circle & Rectangle (Different Data Types)**

**Approach**

* Use function templates to calculate area for both circle and rectangle.
* Demonstrate the template with different data types (e.g., int, double).

**Code**

#include <iostream>  
using namespace std;  
  
// [Template Function for Area of Circle]  
template <typename T>  
T areaCircle(T r) {  
 return 3.14159 \* r \* r;  
}  
// [Template Function for Area of Rectangle]  
template <typename T>  
T areaRectangle(T l, T b) {  
 return l \* b;  
}  
int main() {  
 cout << "Area of circle (double, r=2.5): " << areaCircle(2.5) << endl;  
 cout << "Area of rectangle (int, l=5, b=6): " << areaRectangle(5, 6) << endl;  
 return 0;  
}

**Explanation**

* The areaCircle and areaRectangle functions are templates, so they work with any numeric type.
* The code demonstrates usage with both double and int types.
* The formulas for area are directly implemented in the respective functions.

**Syntax Definitions**

* **template <typename T>**: Allows the function to operate with any data type specified at the call site.
* **endl**: Outputs a newline and flushes the output buffer.

**Q2. Multilevel Inheritance: Employee/Manager/SeniorManager**

**Approach**

* Use a base class Employee with common attributes.
* Derive Manager from Employee, and SeniorManager from Manager.
* Each class adds its own specific data and methods.

**Code**

#include <iostream>  
using namespace std;  
// [Base Class: Employee]  
class Employee {  
protected:  
 string name; int id;  
public:  
 void accept() { cout << "Name: "; cin >> name; cout << "ID: "; cin >> id; }  
 void display() { cout << name << " " << id << endl; }  
};  
// [Derived Class: Manager]  
class Manager : public Employee {  
protected:  
 string department;  
public:  
 void accept() { Employee::accept(); cout << "Department: "; cin >> department; }  
 void display() { Employee::display(); cout << "Department: " << department << endl; }  
};  
// [Further Derived: SeniorManager]  
class SeniorManager : public Manager {  
 int yearsExp;  
public:  
 void accept() { Manager::accept(); cout << "Years Experience: "; cin >> yearsExp; }  
 void display() { Manager::display(); cout << "Experience: " << yearsExp << endl; }  
};  
  
int main() {  
 SeniorManager sm;  
 sm.accept();  
 sm.display();  
 return 0;  
}

**Explanation**

* Employee is the base class with name and id.
* Manager adds a department, and SeniorManager adds years of experience.
* Each class's accept and display methods call the parent class's method to handle inherited data.

**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. Publishing Company Case Study (Inheritance & Exception Handling)**

**Approach**

* Use a base class Publication with title and price.
* Derive Book (adds pages) and Tape (adds play time).
* Use exception handling to set values to zero if invalid input is given.

**Code**

#include <iostream>  
using namespace std;  
class Publication {  
protected:  
 string title; float price;  
public:  
 void accept() { cout<<"Title: "; cin>>title; cout<<"Price: "; cin>>price; }  
 void display() { cout<<"Title: "<<title<<" Price: "<<price<<endl; }  
};  
class Book: public Publication {  
 int pages;  
public:  
 void accept() {  
 try {  
 Publication::accept();  
 cout << "Pages: "; cin >> pages;  
 if (pages <= 0) throw 1;  
 } catch (...) {  
 title = ""; price = 0; pages = 0;  
 cout << "Error! Book values set to zero.\n";  
 }  
 }  
 void display() { Publication::display(); cout<<"Pages: "<<pages<<endl; }  
};  
class Tape: public Publication {  
 float playTime;  
public:  
 void accept() {  
 try {  
 Publication::accept();  
 cout << "Play Time: "; cin >> playTime;  
 if (playTime <= 0) throw 1;  
 } catch (...) {  
 title = ""; price = 0; playTime = 0;  
 cout << "Error! Tape values set to zero.\n";  
 }  
 }  
 void display() { Publication::display(); cout<<"Play Time: "<<playTime<<endl; }  
};  
int main() {  
 Book b; Tape t;  
 cout<<"---Book---\n"; b.accept();  
 cout<<"---Tape---\n"; t.accept();  
 b.display(); t.display();  
 return 0;  
}

**Explanation**

* Publication is the base class for common data.
* Book and Tape inherit from Publication and add their own fields.
* Exception handling ensures that if invalid input is given (pages or play time ≤ 0), all values are set to zero and an error message is shown.

**Syntax Definitions**

* **try-catch**: Used for exception handling; code in try is executed, and if an exception is thrown, control passes to catch.
* **public inheritance**: Allows derived classes to access public and protected members of the base class.