**Slip 6: Sample Solutions and Explanations**

**Q1. Inline Function to Calculate Area of a Circle (with Default Argument)**

**Approach**

* Define an inline function to calculate the area of a circle.
* Use a default argument for the radius so the function can be called with or without a parameter.
* Demonstrate both default and user-supplied radius in main.

**Code**

#include <iostream>  
using namespace std;  
  
// [Inline Function with Default Argument]  
inline double area(double r = 1.0) {  
 return 3.14159 \* r \* r;  
}  
  
int main() {  
 cout << "Default radius area: " << area() << endl;  
 double rad;  
 cout << "Enter radius: ";  
 cin >> rad;  
 cout << "Area with radius " << rad << ": " << area(rad) << endl;  
 return 0;  
}

**Explanation**

* The area function is marked inline and has a default argument (r = 1.0).
* If called without an argument, it uses the default radius of 1.0.
* The user can also input a custom radius, which is then used in the calculation.

**Syntax Definitions**

* **inline**: Suggests the compiler to expand the function at the call site for efficiency.
* **Default Argument**: Allows a function parameter to have a default value if not provided by the caller.

**Q2. Manager Class (Handles Administrative & Technical Details)**

**Approach**

* Create a Manager class with attributes for name, department (administrative), and skillset (technical).
* Provide methods to accept and display all details.

**Code**

#include <iostream>  
using namespace std;  
  
// [Manager Class Definition]  
class Manager {  
 string name, dept, skillset;  
public:  
 void accept() {  
 cout << "Name: "; cin >> name;  
 cout << "Department: "; cin >> dept;  
 cout << "Skillset: "; cin >> skillset;  
 }  
 void display() {  
 cout << name << " " << dept << " " << skillset << endl;  
 }  
};  
  
int main() {  
 Manager mgr;  
 mgr.accept();  
 mgr.display();  
 return 0;  
}

**Explanation**

* The Manager class encapsulates both administrative (department) and technical (skillset) details.
* The accept method reads all fields from the user, and display prints them.

**Syntax Definitions**

* **class**: A user-defined type that groups data and functions.

**Q3. Parking Management System (Constructor/Destructor Case Study)**

**Approach**

* Define a Vehicle class with attributes for plate number, owner, and entry time.
* Use a constructor to initialize and display entry, and a destructor to display exit.
* Demonstrate object lifecycle in main.

**Code**

#include <iostream>  
using namespace std;  
  
// [Vehicle Class with Constructor/Destructor]  
class Vehicle {  
 string plate, owner;  
 int entryTime;  
public:  
 Vehicle(string p, string o, int t) : plate(p), owner(o), entryTime(t) {  
 cout << "Vehicle Entered: " << plate << endl;  
 }  
 ~Vehicle() {  
 cout << "Vehicle Exited: " << plate << endl;  
 }  
 void display() { cout << plate << " " << owner << " " << entryTime << endl; }  
};  
  
int main() {  
 Vehicle v("MH12AB1234", "Rahul", 1005);  
 v.display();  
 return 0; // Destructor called automatically  
}

**Explanation**

* The constructor initializes the vehicle and prints an entry message.
* The destructor prints an exit message when the object goes out of scope (at the end of main).
* The display method shows all vehicle details.

**Syntax Definitions**

* **Constructor**: A special function that initializes an object when it is created.
* **Destructor**: A special function that is called when an object is destroyed, used for cleanup.
* **Member Initializer List**: The : plate(p), owner(o), entryTime(t) syntax initializes members before the constructor body runs.