**Slip 13: Sample Solutions and Explanations**

**Q1. Inline Function to Print Cube of a Number**

**Approach**

* Define an inline function that takes an integer and prints its cube.
* Call the function from main with user input.

**Code**

#include <iostream>  
using namespace std;  
  
// [Inline Function Definition]  
inline void printCube(int n) {  
 cout << "Cube: " << n\*n\*n << endl;  
}  
  
int main() {  
 int x;  
 cout << "Enter a number: ";  
 cin >> x;  
 printCube(x);  
 return 0;  
}

**Explanation**

* The printCube function is marked inline, suggesting the compiler to expand it at the call site for efficiency.
* The function calculates the cube and prints it directly.
* User input is read in main and passed to the function.

**Syntax Definitions**

* **inline**: A keyword that suggests the compiler to insert the function's code at each call site, potentially improving performance for small functions.

**Q2. Person Class: Accept/Display/Search by Contact Number**

**Approach**

* Create a Person class with attributes: name and contact number.
* Accept details for n persons and store them in a vector.
* Display all persons and search for a person by contact number.

**Code**

#include <iostream>  
#include <vector>  
using namespace std;  
  
// [Class Definition]  
class Person {  
 string name, contact;  
public:  
 void accept() {  
 cout << "Enter name: "; cin >> name;  
 cout << "Enter contact number: "; cin >> contact;  
 }  
 void display() { cout << name << " " << contact << endl; }  
 string getContact() { return contact; }  
};  
  
int main() {  
 int n;  
 cout << "How many persons? ";  
 cin >> n;  
 vector<Person> people(n);  
 for(auto &p : people) p.accept();  
 cout << "All persons:\n";  
 for(auto &p : people) p.display();  
 string searchContact;  
 cout << "Enter contact to search: "; cin >> searchContact;  
 bool found = false;  
 for(auto &p : people) {  
 if(p.getContact() == searchContact) {  
 p.display();  
 found = true;  
 }  
 }  
 if(!found) cout << "No record found.\n";  
 return 0;  
}

**Explanation**

* The Person class encapsulates name and contact number, with methods to accept and display data.
* The program reads n persons, displays all, and then searches for a person by contact number.
* If a match is found, the details are displayed; otherwise, a message is shown.

**Syntax Definitions**

* **class**: A user-defined type that groups data and functions.
* **vector**: A dynamic array from the C++ Standard Library that can grow or shrink in size.

**Q3. Generic Data Management System (Template Case Study)**

**Approach**

* Use a template class to manage different types of data (e.g., students, faculty, courses).
* Demonstrate with a Student class and a template DataManager class.
* Add and display records generically.

**Code**

#include <iostream>  
#include <vector>  
using namespace std;  
  
// [Template Class Definition]  
template <typename T>  
class DataManager {  
 vector<T> records;  
public:  
 void add(const T& rec) { records.push\_back(rec); }  
 void display() { for(const auto& r : records) r.display(); }  
};  
  
// [Student Class Definition]  
class Student {  
public:  
 string name; int roll; double grade;  
 void accept() { cout << "Name: "; cin >> name;  
 cout << "Roll: "; cin >> roll;  
 cout << "Grade: "; cin >> grade; }  
 void display() { cout << name << " " << roll << " " << grade << endl; }  
};  
  
int main() {  
 DataManager<Student> dm;  
 int n; cout << "No. of students: "; cin >> n;  
 for(int i=0; i<n; ++i) {  
 Student s; s.accept(); dm.add(s);  
 }  
 cout << "All students:" << endl;  
 dm.display();  
 return 0;  
}

**Explanation**

* The DataManager template class can store and manage any type of record.
* Here, it is used with the Student class to add and display student records.
* The template allows for easy extension to other types (e.g., faculty, courses) by changing the type parameter.

**Syntax Definitions**

* **template <typename T>**: Allows the creation of generic classes or functions that work with any data type.
* **vector**: A dynamic array from the C++ Standard Library.