**Slip 21: Sample Solutions and Explanations**

**Q1. Generic Data Management System (Template-based Record System)**

**Approach**

* Use a template class to add and display any kind of record.
* Demonstrate with Student data; easily extensible to Faculty or Course records.

**Code**

#include <iostream>  
#include <vector>  
using namespace std;  
  
// [Template Class Definition]  
template <typename T>  
class DataManager {  
 vector<T> data;  
public:  
 void add(T rec) { data.push\_back(rec); }  
 void display() { for(auto &r : data) r.display(); }  
};  
  
// [Student Class Definition]  
class Student {  
public:  
 string name; int roll; double grade;  
 void accept() { cin >> name >> roll >> grade; }  
 void display() { cout << name << " " << roll << " " << grade << endl; }  
};  
  
int main() {  
 DataManager<Student> dm;  
 int n; cin >> n;  
 for(int i=0;i<n;++i){ Student s; s.accept(); dm.add(s); }  
 dm.display();  
 return 0;  
}

**Explanation**

* Template class supports flexible record management with a generic interface.
* Here, it is used for Student records, but can be extended to other types.
* The add method stores records, and display prints all records.

**Syntax Definitions**

* **template <typename T>**: Allows the class to work with any data type specified at instantiation.
* **vector**: A dynamic array from the C++ Standard Library.

**Q2. Student Class: Accept, Display, and Search by Grade**

**Approach**

* Create a Student class with attributes: roll number, name, and grade.
* Accept details for n students and store them in a vector.
* Display details of students with a specific grade (e.g., 'A').

**Code**

#include <iostream>  
#include <vector>  
using namespace std;  
  
// [Student Class Definition]  
class Student {  
 int roll;  
 string name;  
 char grade;  
public:  
 void accept() {  
 cout << "Roll: "; cin >> roll;  
 cout << "Name: "; cin >> name;  
 cout << "Grade: "; cin >> grade;  
 }  
 void display() { cout << roll << " " << name << " " << grade << endl; }  
 char getGrade() { return grade; }  
};  
  
int main() {  
 int n;  
 cout << "Number of students: ";  
 cin >> n;  
 vector<Student> students(n);  
 for(auto &s : students) s.accept();  
 char searchGrade;  
 cout << "Enter grade to search: "; cin >> searchGrade;  
 cout << "Students with grade " << searchGrade << ":\n";  
 for(auto &s : students)  
 if(s.getGrade() == searchGrade) s.display();  
 return 0;  
}

**Explanation**

* The Student class encapsulates student data and provides methods to accept and display it.
* The program reads n students, then displays those with the specified grade.
* The getGrade method is used for filtering.

**Syntax Definitions**

* **class**: A user-defined type that groups data and functions.
* **vector**: A dynamic array from the C++ Standard Library.

**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() { cin >> name >> roll >> grade; }  
 void display() { cout << name << " " << roll << " " << grade << endl; }  
};  
  
int main() {  
 DataManager<Student> dm;  
 int n; cin >> n;  
 for(int i=0; i<n; ++i) {  
 Student s; s.accept(); dm.add(s);  
 }  
 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.