Index

About this Project :-

A basic student management system in C++, is to provide a simple example of how object-oriented programming (OOP) concepts can be applied to create a program that manages student information. Here are some educational goals and motivations behind this project:

The primary purpose of this project is to illustrate how to create a simple student management system using classes and vectors in C++. It is a starting point for learners to understand the basics of object-oriented programming and data management within the context of a small-scale application. Users can add students, view their details, and search for students based on their ID.

Introduction to OOP: The project introduces fundamental concepts of object-oriented programming, including classes and objects. The Student class represents a real-world entity (a student) with attributes and behavior.

Data Management: The program illustrates how to manage and organize data using classes and data structures. In this case, a vector is used to store a collection of Student objects.

User Interaction: The program involves user interaction through the console. Users can add students, display student details, and search for students by ID. This introduces basic input/output operations and user interface considerations.

Functionality Expansion: The project can serve as a starting point for further expansion. Additional features can be added, such as updating student information, deleting students, or persistently storing data in files.

Understanding Code Structure: The structure of the code demonstrates separation of concerns. The Student class encapsulates student-related data and behavior, and the StudentManager class encapsulates the management of students.

Practice with C++ Syntax: The project provides hands-on practice with C++ syntax, including class definitions, member functions, vectors, and basic console I/O operations.

Introduction to C++ :-

Certainly! C++ is a general-purpose programming language that was created as an extension of the C programming language. Developed by Bjarne Stroustrup at Bell Labs in the early 1980s, C++ is known for its efficiency, performance, and flexibility. It combines procedural programming with object-oriented programming (OOP) features, making it a versatile language suitable for a wide range of applications.

Key Features of C++:

C Compatibility:

C++ is designed to be highly compatible with C, allowing C code to be easily integrated into C++ programs. C++ includes nearly all of C's features and adds new capabilities.

Object-Oriented Programming (OOP):

C++ supports the principles of object-oriented programming, including encapsulation, inheritance, and polymorphism. This allows for the creation of modular and reusable code.

Efficiency and Performance:

C++ provides low-level access to memory, allowing for fine-grained control over system resources. This makes it suitable for systems programming and performance-critical applications.

Standard Template Library (STL):

The STL is a powerful set of C++ template classes and functions that provide common data structures (such as vectors, lists, and queues) and algorithms (sorting, searching, and more). It promotes code reuse and generic programming.

Multi-Paradigm Language:

C++ supports multiple programming paradigms, including procedural programming, object-oriented programming, and generic programming. This versatility allows developers to choose the most appropriate paradigm for a given task.

Portability:

C++ code can be compiled on various platforms, making it a portable language. This is facilitated by the availability of compilers for different operating systems.

Standardization:

The language is standardized by the International Organization for Standardization (ISO). The latest standard, as of my knowledge cutoff in January 2022, is C++17, with ongoing efforts for future standards.

Use Cases:

System Programming: C++ is often used for low-level programming, including operating systems, device drivers, and embedded systems.

Game Development: Many video games are developed using C++ due to its performance and ability to control hardware resources.

Application Development: C++ is used to build a variety of applications, ranging from desktop applications to high-performance servers.

Large Software Systems: Its support for OOP and modular programming makes C++ suitable for large-scale software development.

Performance-Critical Applications: C++ is chosen for applications where performance is crucial, such as real-time systems and scientific computing.

Code :-

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
// Class to represent a student
class Student {
public:
   int id;
   string name;
   int age;
   // Constructor to initialize the student object
   Student(int _id, string _name, int _age) : id(_id), name(_name), age(_age) {}
};
// Class to manage students
class StudentManager {
private:
   vector<Student> students; // Vector to store student objects
public:
   // Function to add a new student
   void addStudent(int id, string name, int age) {
      Student newStudent(id, name, age);
      students.push_back(newStudent);
      cout << "Student added successfully!\n";
   }
```

```
// Function to display all students
   void displayStudents() {
      cout << "Student List:\n";
      for (const auto& student : students) {
         cout << "ID: " << student.id << "\tName: " << student.name << "\tAge: " << student.age <<
endl;
      }
   }
   // Function to search for a student by ID
   void searchStudentById(int id) {
      auto it = find_if(students.begin(), students.end(), [id](const Student& student) {
         return student.id == id;
      });
      if (it != students.end()) {
         cout << "Student found!\n";
         cout << "ID: " << it->id << "\tName: " << it->name << "\tAge: " << it->age << endl;
      } else {
         cout << "Student not found!\n";
      }
   }
};
int main() {
   StudentManager studentManager;
   // Adding sample students
   studentManager.addStudent(1, "Alice", 20);
   studentManager.addStudent(2, "Bob", 22);
   studentManager.addStudent(3, "Charlie", 21);
```

```
// Displaying all students
   studentManager.displayStudents();
   // Searching for a student by ID
   studentManager.searchStudentById(2);
   return 0;
}
```

Explanation :-

1. Header Files

```
#include <iostream>
#include <vector>
```

Explanation: These lines include necessary header files. <iostream> is included for input/output operations, and <vector> is included for using the std::vector class.

2. using namespace std;

Explanation: This line is used to avoid writing std:: before standard C++ identifiers. It makes code more concise. However, note that using namespace std in large projects can lead to naming conflicts.

3. class Student

```
            class Student {
            public:
   int id;
   string name;
   int age;

   Student(int _id, string _name, int _age) : id(_id), name(_name), age(_age) {}
};
```

Explanation: This defines a class named Student. It has three public attributes (id, name, and age). The constructor initializes these attributes when a Student object is created.

4. class StudentManager

```
   class StudentManager {
private:
   vector<Student> students;
public:
   void addStudent(int id, string name, int age) {
       Student newStudent(id, name, age);
       students.push_back(newStudent);
       cout << "Student added successfully!\n";
   }


   void displayStudents() {
       cout << "Student List:\n";
       for (const auto& student : students) {
           cout << "ID: " << student.id << "\tName: " << student.name << "\tAge: " << student.age <<
endl;
       }
   }
   void searchStudentById(int id) {
       bool found = false;
       for (const auto& student : students) {
           if (student.id == id) {
               found = true;
               cout << "Student found!\n";
               cout << "ID: " << student.id << "\tName: " << student.name << "\tAge: " << student.age <<
endl;
               break;
           }
       }
       if (!found) {
           cout << "Student not found!\n";
       }
   }
};
```

Explanation: This defines a class named StudentManager. It has a private vector of Student objects. It includes three member functions:

addStudent: Adds a new student to the vector.

displayStudents: Displays details of all students in the vector.

searchStudentById: Searches for a student by ID and displays details if found.

5. int main()

```
int main() {
   StudentManager studentManager;
   studentManager.addStudent(1, "Alice", 20);
   studentManager.addStudent(2, "Bob", 22);
   studentManager.addStudent(3, "Charlie", 21);
  studentManager.displayStudents();
  studentManager.searchStudentById(2);

   return 0;
}
```

Explanation: This is the main function, the entry point of the program. It creates an instance of StudentManager, adds sample students, displays all students, and searches for a student by ID.

Output:-

```
Student added successfully!
Student added successfully!
Student added successfully!
Student List:
ID: 1    Name: Alice   Age: 20
ID: 2    Name: Bob     Age: 22
ID: 3    Name: Charlie Age: 21
Student found!
ID: 2    Name: Bob     Age: 22
```

Adding Sample Students:

Three students are added to the StudentManager using the addStudent function.

Messages "Student added successfully!" are printed to indicate successful additions.

Displaying All Students:

The displayStudents function is called to show details (ID, name, and age) of all students.

The output displays the information for each student in the vector.

Searching for a Student by ID:

The searchStudentById function is called to search for a student with ID = 2.

The output indicates that the student with ID = 2 (Bob) is found, displaying the details.