Design a class/struct named 'Student' which will have the following member variables and functions:

Variables:

* Name
* ID no
* Batch no
* CGPA
* Friends ID (A list of ID's of one's friends)

Functions:

* int numberOfFriends() -> returns the number of friends of this student
* void printInfo() -> prints all the member variable of a student object
* void printNetwork() -> prints all the friends of friends of friends.... of a student. Let's say, I am Sakib. My friend is Sajib. Sajib's friend is Kuddus. Rahim's friend is Sattar. If I call printNetwork() for Sajib/Sakib/Kuddus, it will print Sakib, Sajib, Kuddus.

Complete the class/struct implementation with the best coding practice (i.e. proper indentation, proper variable naming etc)

Q1. Write compare functions to sort a list of students in ascending order of their:

* ID no
* CGPA (if tie, sort with ID)
* Number of friends

Q2. Given two IDs, write a function to check whether they are connected within the same network.

Solution (C++):

| #include <bits/stdc++.h> using namespace std;  class Student {  /// member variables are private. you should not allow others to access or modify them without appropriate getters/setters.  string name;  int id\_no, batch\_no;  double cgpa;  vector<int>friends\_id; public:  /// Constructor  Student(string \_name, int \_id, int \_batchNo, double \_cgpa)  {  name = \_name;  id\_no = \_id;  batch\_no = \_batchNo;  cgpa = \_cgpa;  }  unsigned int numberOfFriends()  {  return friends\_id.size();  }  void addFriendId(int \_id)  {  friends\_id.push\_back(\_id);  }   /// Setters  void setId(int \_id)  {  id\_no = \_id;  }   /// Getters  string getName()  {  return name;  }  int getId()  {  return id\_no;  }  double getCgpa()  {  return cgpa;  }  vector<int>getFriendIdList()  {  return friends\_id;  }  void printInfo()  {  cout << "ID No: " << id\_no << "\n";  cout << "Batch No: " << batch\_no << "\n";  cout << "Name: " << name << "\n";  cout << "Cgpa: " << cgpa << "\n";  cout << "Friend's ID: \n";  for(int id : friends\_id)  {  cout << id << " ";  }  cout << "\n";  } };  /// global storage for all students vector<Student>students;  void connect(Student s1, Student s2) {  int id1 = s1.getId();  int id2 = s2.getId();  students[id1].addFriendId(id2);  students[id2].addFriendId(id1); }  bool comp\_by\_id(Student &s1, Student &s2) {  return s1.getId() < s2.getId(); }  bool comp\_by\_cgpa(Student &s1, Student &s2) {  return s1.getCgpa() == s2.getCgpa() ? comp\_by\_id(s1, s2) : s1.getCgpa() < s2.getCgpa(); }  bool comp\_by\_friendList(Student &s1, Student &s2) {  return s1.numberOfFriends() < s2.numberOfFriends(); }  void addStudent(Student student) {  students.push\_back(student); }  void printNetwork(Student &student, vector<bool>&visited) {  visited[student.getId()] = true;  cout << student.getId() << " , " << student.getName() << "\n";  if(student.numberOfFriends() == 0) return;  vector<int>friendsId = student.getFriendIdList();  for(int id : friendsId)  {  if(!visited[id])  {  printNetwork(students[id], visited);  }  } } void explore\_network(Student &student, vector<bool>&visited) {  visited[student.getId()] = true;  if(student.numberOfFriends() == 0) return;  vector<int>friendsId = student.getFriendIdList();  for(int id : friendsId)  {  if(!visited[id])  {  explore\_network(students[id], visited);  }  } }  bool isFriend(Student &s1, Student &s2) {  vector<bool>visited(students.size(), 0);  explore\_network(students[s1.getId()], visited);  return visited[s2.getId()]; }  void printNetwork(Student student) {  vector<bool>visited(students.size(), 0);  printNetwork(students[student.getId()], visited); }  void sort\_by\_id() {  sort(students.begin(), students.end(), comp\_by\_id); }  void sort\_by\_cgpa() {  sort(students.begin(), students.end(), comp\_by\_cgpa); }  void sort\_by\_friendList() {  sort(students.begin(), students.end(), comp\_by\_friendList); }  int main() {  Student stud0("Sakib", 0, 0, 3.30);  Student stud1("Rakib", 1, 0, 3.80);  Student stud2("Sajib", 2, 1, 3.37);  Student stud3("Rafi", 3, 1, 3.00);   addStudent(stud0);  addStudent(stud1);  addStudent(stud2);  addStudent(stud3);   connect(stud0, stud1);  connect(stud1, stud3);  printNetwork(stud0);   sort\_by\_friendList();  sort\_by\_cgpa();  for(auto student : students)  student.printInfo();  return 0; } |
| --- |
|  |