```
pragma solidity ^0.5.16;

contract DecentralizedUniversity {

address universityAdmin;
uint256 public totalNoOfColleges;
uint256 public totalNoOfStudents;

constructor() public {
  universityAdmin = msg.sender;
}
modifier onlyAdmin(){
  require(msg.sender == universityAdmin);
  _;}

struct College{
  string cName;
  address cAddress;
  address cAdmin;
  string cRegNo;
  bool isAllowedToAddStudents;
  uint totalNoOfStudents;
}

struct Student{
  string sName;
  uint phoneNo;
  string courseEnrolled;
}
```

```
mapping (address => College) colleges; //  Mapping a college's address to college

mapping (string => Student) students; //  Mapping a student's name to student




function addNewCollege(string memory collegeName, address add, address cAdmin, string memory regNo) public onlyAdmin {

  require(!areBothStringSame(colleges[add].cName,collegeName), "College already exists with same name");

  colleges[add]= College(collegeName,add,cAdmin,regNo,true,0);

  totalNoOfColleges++;

}


function viewCollegeDetails(address add) public view returns (string memory, string memory, uint) {

  return (colleges[add].cName,colleges[add].cRegNo, colleges[add].totalNoOfStudents);

}


function blockCollegeToAddNewStudents(address add) public onlyAdmin {

  require(colleges[add].isAllowedToAddStudents, "College is already blocked to add new students");

  colleges[add].isAllowedToAddStudents=false;

}



function unblockCollegeToAddNewStudents(address add) public onlyAdmin {

  require(!colleges[add].isAllowedToAddStudents, "College is already unblocked to add new students");

  colleges[add].isAllowedToAddStudents=true;

}
```

```
    function addNewStudentToCollege(address add,string memory sName, uint phoneNo, string memory
courseName ) public {
        require(colleges[add].isAllowedToAddStudents, "This College is blocked to add new students");
        require(colleges[add].cAdmin == msg.sender, "Only College admin can add the new student");
        students[sName] = Student(sName,phoneNo,courseName);
        colleges[add].totalNoOfStudents += 1;
        totalNoOfStudents++;
    }


    function getNumberOfStudentForCollege(address add) public view returns(uint){
        return (colleges[add].totalNoOfStudents);
    }


function viewStudentDetails(string memory sName) public view returns (string memory, uint, string
memory) {
   return (students[sName].sName,students[sName].phoneNo, students[sName].courseEnrolled);
}


function changeStudentCourse(address add, string memory sName, string memory newCourse) public {
      require(!areBothStringSame(students[sName].courseEnrolled,newCourse), "Student already
enrolled to same course");
      require(colleges[add].cAdmin == msg.sender, "Only College admin can change the student course");
      students[sName].courseEnrolled=newCourse;
}
```

```
    function areBothStringSame(string memory a, string memory b) private pure returns (bool) {
        if(bytes(a).length != bytes(b).length) {
            return false;
        } else {
            return keccak256(bytes(a)) == keccak256(bytes(b));
        }
    }

}
```