Practice Problem

1. Data Structure Vector : Complete the following program implementing appropriate method and properties for Class Vector. (Find Codes in moodle)

```
#include<iostream>

using namespace std;

class Vector{

    private:
        int *vector;
        int size;
        int maximumCapacity;
        int isUnboundedVector;

    public:

        /*
            if capacity is not specified
            create initial vector of capacity 2
            marked is as unbounded
            default value is '\0' or 0
        */
        Vector(){

        }

        /*
            if capacity specified then create a vector
            with that maximum size. Mark bounded.
        */
        Vector(int capacity){


        }

        /*
            add to last
            if vector is bounded and size reached the capacity
            show maximum capacity reached
            for unbounded expaned the memory by 2 times the current capacity.
            to do this malloc a new location with 2 times the capacity
            copy existing values the point vector to that location
```

```
*/

int add(int value){

}

/*
    add to a specific position replacing the value if exists
    if beyond capacity for unbounded expand
    for bounded show error

*/

int add(int position,int value){

}

/*
    remove last entry
    decrement size
*/

int remove(){

}
/*
    remove specific positionvalues
    mark it as null or '\0'
*/
int remove(int position){

}
/*
    print valid values from start to size
    with valid (index,value) pair
*/
void printVector(){

}

/*
    print valid values between these position
*/
void printVector(int startPosition, int endPosition){

}
//free the allocated memory
~Vector(){

}
```

```
}


int main(){

    Vector bounded(5);
    Vector unbounded;

    bounded.add(4);
    bounded.add(5);
    bounded.add(1);
    bounded.add(3);
    bounded.add(2);
    bounded.add(10); //This will fail to insert.

    unbounded.add(1);
    unbounded.add(2);
    unbounded.add(3);
    unbounded.add(4);

    return 0;
}
```

2. Geometry Vector: Complete the following program by implementing Vector class.

```
#include<iostream>

using namespace std;

class Vector{

    private:
        int i;
        int j;
        int k;

    public:

        Vector(){
            //print constructing with default values

        }
```

```
Vector(int a,int b,int c){
    //print constructing with a, b, c values
}

int getI(){

}

int getJ(){

}

int getK(){

}

void setI(){

}

void setJ(){

}

void setK(){

}

double getMagnitude(){

}

Vector getDirectionVector(){


}

///return a new vector after adding current vector + vect.
///no change to this vector object

Vector addVector(Vector &vect){

}


Vector getNormalVector(){

}

///return cross product of this vector and the passed vector
```

```
        ///no change to this vector

        Vector crossProduct(Vector &vect){

        }

        ///return dot product of this vector and passed vector as a new vector

        Vector dotProduct(Vectory &vect){

        }

        ///multiply is vector component by this value

        Vector scaling(int multValue){

        }

        ///print component

        void printVector(){

        }



        ~Vector(){
            cout<<"Destructing ("<<i<<","<<j<<","<<k<<")"<<endl;
        }

}


int main(){

    ///you must be able to explain construct and destructing output sequence


    return 0;
}
```

3. Linked list: if you are already familiar with linked list then practice this otherwise ignore it.

```
#include<iostream>

using namespace std;

///Do not change this class

class Element{
    private:
        int value;
        Element *next;
    public:

        Element(){
            this->next=NULL;
        }
        Element(int value){
            this->value=value;
            this->next=NULL;
        }

        void setValue(int value){
            this->value=value;
        }

        int getValue(){
            return this->value;
        }

        void setNext(Element* aNext){
            next = aNext;
        }

        Element* getNext()
        {
            return next;
        }

};

///Implement here

class LinkedList{
    private:
```

```
    Element *head;
    int size;

public:
    LinkedList(){
        head=new Element();
        size=0;
    }

    ///add element to last element exists.
    void add(Element element){

        Element *temp=head;

        while(temp->getNext()!=NULL){
             temp=temp->getNext();
        }

        Element *newElement=new Element(element.getValue());
        temp->setNext(newElement);

        size++;
    }

    ///return the size
    int getSize(){

    }

    ///return the element of the specified position
    ///return NULL if position greater than size
    Element find(int position){

    }

    ///check if the element exists
    ///check by value
    bool find(Element element){

    }

    ///remove element from the specified position
    ///return true if can be successfully returned
    bool remove(int position){

    }

    ///remove the first element that matches the value
    bool remove(Element Element){
        }

        ///printLinkedListValue
        void printList(){

            Element *temp=head;

            for(int i=0;i<size;i++){
                temp=temp->getNext();
                cout<<temp->getValue()<<" -> ";
            }

            cout<<endl;
        }

};


int main(){

    LinkedList list;

    Element e1(1);
    Element e2(2);
    Element e3(3);

    list.add(e1);
    list.add(e2);
    list.add(e3);

    list.printList();


    return 0;
}
```

5. University/Departments/Student

Class Student:

- Write a class student which contains, public attribute
  - Char array of name
  - Int roll

- Private attribute
  - Double cgpa

- Write constructor for initializing the properties of class

- Write appropriate methods for setting and getting this properties

- Print Function display student information

Class Department:

- Write class Department which contains public attribute
  - Department name
- Private attribute,
  - Current Student No
  - Array of Students of size 100

- Constructor for setting name and initializing private attribute

- Write appropriate methods for

  - Adding a new student

    - Removing a student
    - Print Department Information

Class University:

- Write class University which contains public attribute
    - University name

- Private attribute
    - Array of Departments of size 10
    - Current Department No

- Write appropriate constructor for initializing attributes

- Write methods for
    - Add Department
    - Remove Department
    - Add a new student in a particular department

        Eg. university.addStudent(studentObj, departmentObj)
    - Print University Info
        - Print University information in tree format

```
BUET:
    CSE
          1. Anika
          2. Anik
          3. Ishtiyaque
    EEE
          1. Sadman
          2. Aminul
AUST:
    CSE
          1. Rahim
          2. Karim
          4. Babul
    EEE
          3. Rakib
          4. Jahid
```