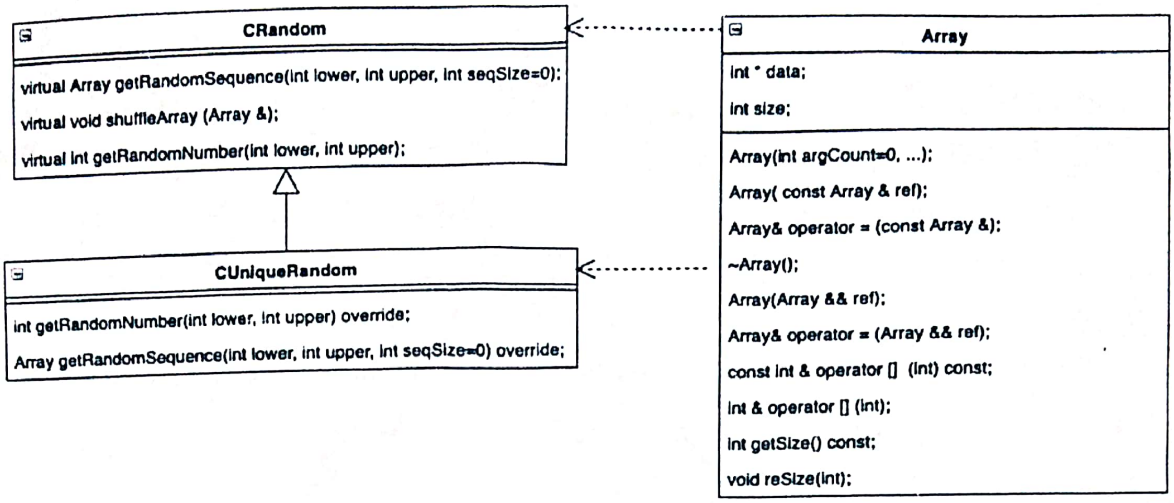
Objective:

- Exploring an Application of Polymorphism and How to write a generic function in the old 'C' era.

Challenge A: (21)

For this task, you are to develop following set of classes as shown in UML class diagram.



Here is the detailed explanation of each class:

Array

You already know about it.

CRandom

- `virtual Array getRandomSequence(int lower, int upper, int seqSize=0);`
  Returns array of integers of size seqSize populated with randomly generated integers in the range of lower to upper inclusive.
  If seqSize==0 then Array size will be equal to upper-lower+1.
  Note: randomly generated elements in array may repeat.

- `virtual void shuffleArray(Array &);`
  Shuffles the elements of the received array.
  To get full credit: your algorithm should be efficient and unbiased (means each element has equal probability to be placed anywhere). Hint: QuizApp.

- `virtual int getRandomNumber(int lower, int upper);`
  Returns a random integer in the range lower to upper inclusive

CUniqueRandom

- `int getRandomNumber(int lower, int upper) override;`
  Returns a random integer in the range lower to upper inclusive but it makes Sure that it doesn't generate/return a duplicate random number consecutively. Means, current call to *getRandomNumber* must not generate the same number which it generated on the immediately previous call.

- `Array getRandomSequence(int lower, int upper, int seqSize=0) override;`
  Returns array of integers of size seqSize populated with randomly generated integers in the range of lower to upper inclusive.
  Note: randomly generated elements in array must not repeat if possible.
  If seqSize <= upper-lower+1 Then elements in array must not be duplicate.
  If seqSize > upper-lower+1 Then elements in array may be duplicate but it guarantees that it must have all the range values at least once and for rest of the empty slots it must fill them randomly from the range with equal probability
  If seqSize==0 then Array size will be equal to upper-lower+1.
  //Note: While populating the sequence, your algorithm should make sure that each element in the range has equal probability to be part of sequence.

Note:

- In the above classes, the range lower ~ upper can be any of $\{(+,+),(-,-),(-,+)\}$.
  Assumption: lower<=upper.
- You are free to decide about adding any private **data member(s)** or any **ctor/dtor** needed for your classes (CRandom and CUniqueRandom only) but you got to justify your decision to the TA, failing to give some acceptable/reasonable justification will result in less or zero marks.

Sample Run	Sample Output
`int main()`	
`{`	
`CRandom cr;`	
`Array x = cr.getRandomSequence(5, 19);`	
`for (int i=0; i<x.getSize(); i++)`	
`cout<<x[i]<<":";`	10:5:15:5:19:15:12:16:19:8:10:6:17:19:12:
`CUniqueRandom cur;`	
`Array un = cur.getRandomSequence(5, 19);`	
`for (int i=0; i<un.getSize(); i++)`	
`cout<<un[i]<<":";`	10:16:15:8:19:7:9:5:11:6:17:14:13:12:18:
`for (int i=1; i<=10; i++)`	
`cout<<cr.getRandomNumber(25, 50)<<":";`	40:38:33:41:36:36:31:25:35:27:
`for (int i=1; i<=10; i++)`	
`cout<<cur.getRandomNumber(25, 50)<<":";`	25:34:46:48:41:29:28:48:40:32:
`return 0;`	
`}`	

(10)

Challenge X:
Give definition of following function which is capable of sorting any type of array received.

```
void myGenericSort (void * baseArr, int arrElementCount, int sizeOfEachElement,
                    bool (*comparator)(const void*, const void*));
```

See the sample run below for further details:

Sample Run	Sample Output
`int main()`	
`{`	
`double a[5] ={5.6745,1.3,30,40, 2.3};`	
`int b[8] = {10,2,-30,5, 4, 5, 9, 1};`	
`myGenericSort(a, 5, 8, myDoubleComparator);`	
`for (int i=0; i<5; i++)`	
`cout<<a[i]<<" : ";`	1.3 : 2.3 : 5.6745 : 30 : 40 :
`cout<<endl;`	
`myGenericSort(b, 8, 4, myIntComparator);`	
`for (int i=0; i<8; i++)`	
`cout<<b[i]<<" : ";`	10 : 9 : 5 : 5 : 4 : 2 : 1 : -30 :
`return 0;`	
`}`	

Note following points:

```
void * baseArr:-----------------------------: pointer to the array.
int arrElementCount:------------------------: count of elements in the array
int sizeOfEachElement:----------------------: size in bytes of an element of array
bool (*comparator)(const void*, const void*): Will receive the address of function which is
                                              responsible for comparing two elements of array.
```

I have shared the sample prototype of function (used in sample run) for your help to understand the purpose of it. It is supposed to return true if a > b otherwise false.

```
bool myIntComparator(const void * a, const void * b);
bool myDoubleComparator(const void * a, const void * b);
```

Soon, we shall see an elegant way of writing generics.

"We can't take any credit for our talents. It's how we use them that counts."

-- --

"I am not bound to win, but I am bound to be true. I am not bound to succeed, but I am bound to live up to what light I have."

-- Abraham Lincoln --