Objective:

- Defining an ADT that supports the creation of ND-Arrays in order to consolidating the concept of ND-Arrays.
- Learning the advantage of Array Representation Schemes in JAVA/C# and in newer languages.

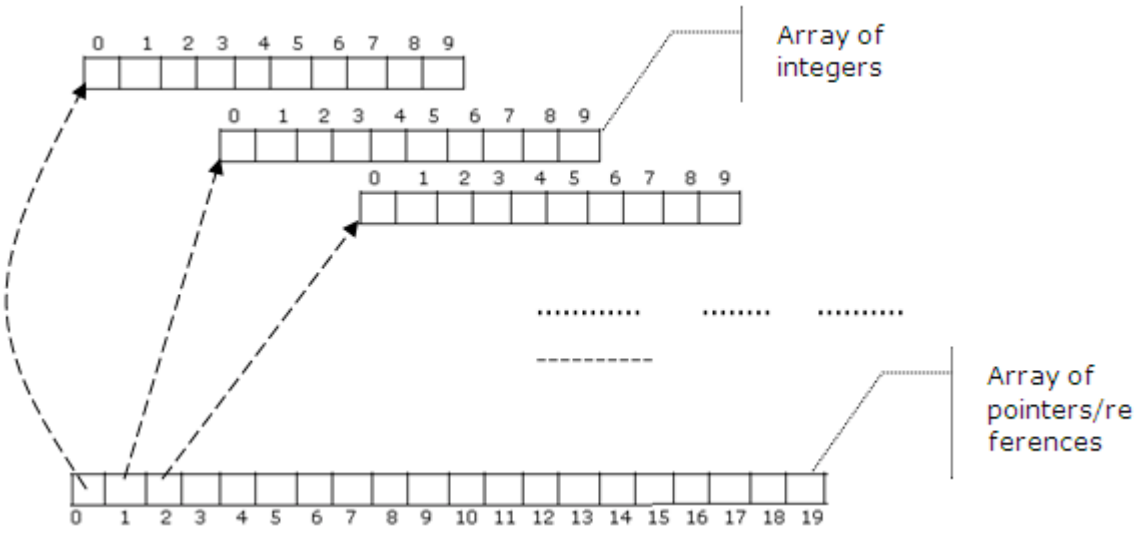
Task-1:

Ali is a very good Java Developer and he wants to create an array of integers of 200 locations, so technically speaking he must get a contiguous chunk of 800 bytes in RAM (in Java integer takes 4 bytes), but System reports him that in RAM there is thousands of bytes available with many smaller chunks but do not have any contiguous chunk of 800 bytes. **OOPS!**

But Ali is a genius guy and he comes up with a scheme, a data structure that can help him to resolve this problem.

Now, let's have a look on this scheme!

Ali says that there is high probability that I will get many smaller chunks as the System reported so he come up with a structure i.e. he makes many smaller size arrays of integers in RAM and will save their references in another array, See the diagram below:



For array of 200 locations he makes 20 smaller integer arrays, each of size 10 locations. So he needs to make another array of 20 locations to store the references of these 20 integer arrays.

- After developing this scheme Ali is claiming that he is achieving the same read/write access cost that could be in an array i.e. access to particular index is at O(1) time. As an example if he wants to access index number 47 in the array then this index is represented by index 7 of the 5th chunk in his scheme.
- Now you have to give just a formula that tells any index 'K' in an array of 200 locations is represented by which chunk-number and index in the chunk.

Task-2:

Define a function which prints the Row-Major based ND-array formula against a given number of dimensions.

For Example;

If the function is called as 'printND(3)' then the following should be displayed on console:

$I_1U_2U_3 + I_2U_3 + I_3$

for 'printND(1)'

I_1

for 'printND(2)'

$I_1U_2 + I_2$

Where I_1 , I_2 , I_3 , …, I_N represents the index set and

U_1 , U_2 , U_3 , …, U_N represents the dimension set.

Task-3:

Requirements:

- Even though the multidimensional array is provided as a standard data object in C++, it is often useful to define your own class for multidimensional arrays. This gives a more robust class that:
  - Does require the index set in each dimension to start at any number (Є Set of Integers).
  - Selects range of each dimension of the array during runtime.
  - Provide mechanism to determine the size of array.

No specific data structure is being told to you for this task so intelligently do a proper structure selection yourself and when feels confident then also discuss your structure with me before start coding.

Sample Program Run

```
int dim-size[3]={5,3,10};

NDArray arr(3, dim-size); // first parameter = number of dimension
                          // second parameter = size of each dimension
int index-set[3]={4,2,8};

int val;
cin>>val

arr.setValue( index-set , val);
cout<<arr.getValue(index-set)
.
.
.
.
.
```