CSE225L – Data Structures and Algorithms Lab
Lab 03
Classes & Objects

dynarr.h

```
#ifndef DYNARR_H_INCLUDED
#define DYNARR_H_INCLUDED
class dynArr
{
    private:
          int *data;
          int size;
    public:
          dynArr();
          dynArr(int);
          ~dynArr();
          void setValue(int, int);
          int getValue(int);
};
#endif // DYNARR_H_INCLUDED
```

dynarr.cpp

```
#include "dynarr.h"
#include <iostream>
using namespace std;

dynArr::dynArr()
{
    data = NULL;
    size = 0;
}
dynArr::dynArr(int s)
{
    data = new int[s];
    size = s;
}
dynArr::~dynArr()
{
    delete [] data;
}
int dynArr::getValue(int index)
{
    //write the code
}
void dynArr::setValue(int index, int value)
{
    //write the code
}
```

Tasks:

Task 1: In the driver file (main.cpp), perform the following sub-tasks.

1. Create two objects of this class, one with no constructor argument and one with the argument 5.
2. Take five input values from the user and store them in the array inside the second object using the set method.
3. For the second object, print all the values you just stored.

Note that, you cannot assign anything in the first object since the array inside it has size 0. Neither can you change the size of this array to some other size.

Task 2: Modify the header and the source files. Add a member function **`void allocate(int s)`** which allows you to change the size of the array. Make sure that memory is not leaked.

Task 3: Modify the header file and the source files again, so that it works for two dimensional array where all the rows are the same size. The user will specify the number of rows and columns as well as the content of the array, which you will take as input from user in the main function.

What is 'Template Class' in C++:

"Template Class" is an important feature of C++ which enables the coder to write **generic** functions or classes. In a **generic function or class**, the type of data (i.e: int, float, double, etc.) upon which the function or class operates is specified as a parameter.

Why 'Template Class'?

By creating a templated class/ function, you can define the nature of your algorithm to be independent of any kind of data types.

Once you have written a templated code, your compiler will automatically generate the correct code for the type of data that is actually used when you execute the function.

Format for writing a 'Template Class' in C++

Remember the simple **DynamicArray** class we discussed in our **class**. If we convert that simple class into a templated class, then that class object will be able to hold any valid type of numeric values (int, float, double). Now, the format for writing a template function in C++ (in the source .cpp file) is as follows:

```
template <class ItemType>
return-type Class_Name<ItemType>::functionName(parameters)
{
        // your code goes here
}
```

Task 4: Modify the header file and the source file given below so that they now work as template class (the array elements in the dynamically allocated memory can be any type as the user defines).