# Assignment 02 – Memory Management & Operator Overloading

**Average expected estimated time to complete this assignment is 12 hours**

- [1 HOUR] software requirement understanding
  - reading of the assignment document
  - understanding of problem statement
  - understanding of software inputs and outputs
- [2 HOURS] software design
  - identification of user-software interaction sequences
    - software menus
    - user inputs and
    - software flows
  - identification of required variables and data structures to be used
    - for input, processing, and output
    - identification of storage class specifications of required variables
  - modularization of software into required functions
    - proper naming of functions
    - identification of tasks to be performed into functions
    - identification of parameters and return types of functions
  - identification of user defined header files
    - names and functions to be placed in
  - flow sequence of main function and call of various user defined functions from it
- [6 HOURS] software coding
  - coding of user defined functions
  - placement of user defined functions into header file/s
  - coding of main function
- [2 HOURS] software testing
  - running software on various inputs
  - identification of software error and bugs
  - removal of software error and bugs
- [1 HOUR] software documentation
  - proper indentation of code
  - use of proper naming conventions
  - commenting of code

## Objectives:

- This assignment will provide experience with Class, Constructors, Destructor, Member Functions, Operator Overloading

## Prerequisite Skills:

- Understanding of Class design
- Understanding of Constructor / Destructor
- Understanding of Operator Overloading
- Understanding Memory Management

## Before you start:

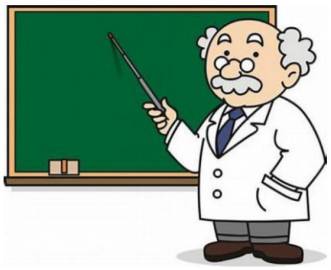
Please consider the following guidelines before you start the Assignment:

- Make sure that you understand the complete assignment before you actually start implementing the code.
- Your code must be in clean and easy to read form.
- Please handle all the corner cases and input validations of parameters inside functions. Functions without the exception handling wont be graded full marks.
- For any kind of understanding issues, please consult your TAs.
- After the assignment deadline, we will have a Viva for this assignment.
- ***Attention: Any form of cheating detected will result in 0 marks in your OOP sessional activities. So avoid cheating from each other and using AI tools.***

# Our very Own vector

Hi! Class, meet ***Mr. Clark***. Mr. Clark has heard that you guys are going incredible in OOP ☺. So he is here to have a little test of all of you. Over to Mr. Clark now. With him is Mr. Phil. While Mr.Clark is about to test you, Mr. Phil is here to guide you along the way. He often shares a word of advice whenever he feels need to. So keep a close ear (I mean eye) to Mr. Phil words. With him is Detective Oscar. He is very suspicious and often thinks out of the box about the corner cases in programs which can throw exceptions. He will let you know whenever he thinks an exception might occur here. Pay attention to him to make your vector robust and error free.

**Mr Clark:** Hi! class, I am Prof. Clark. I am here to have a little OOP programming exercise with you. So lets get to business.



Students, you guys have used vector STL in your Labs. Isn't it great? Well, lets just create a vector template header file of our own. So buckle up, get exited and lets go:

Your vector class will be in a separate header file named as vector.h, so that it can be included in your other programs and thus can be used.

Vector class should be template based, meaning it should work with Data of any kind.

It private members, vector class will have template type D-array, which obviously will have no memory to point to.

### Data member:

- ***data_array:*** this D-array will hold the Data for our vector. It will be template type.
- **max_size:** this will be the max number of elements vector can hold at time. We will get its value from parameterized constructor.

***Mr Phil:***

*What about a variable to track the current size of our D-array?*

## *Required methods of vector class:*

- ***Default constructor:*** This constructor will initialize D-array with the nullptr.
- ***Copy Constructor :*** There will be Copy constructor which will copy all the elements of other object using Deep copy.
- ***Parameterized construct*** D-array and initialize all indexes to Default values of Data type.

**Detective Oscar:**

*It's seems like a good idea to make sure that size is greater than zero. Lets throw exception If it is, in try catch block.*

- ***int size():*** This function will return the current size of vector. It should be const.
- ***int getMaxSize():*** This function will return the Max size, the vector can have. It should be const as well.
- ***void resize():*** this function will be private and will be called whenever the D-array reaches its max_size. This function will double the size of vector.

Lets say max size of vector was 5. current size also reaches 5. Then this function will be called to double the size of D- array. Double size will be 10. Thus now in D-array inside vector class, 5 indexes will hold the 5 previous stored values and now will have 5 free indexes to store 5 more values.

Also the same function will be called when after removing an element from vector class, the size of vector gets 1 /4$^{th}$ or less of the max size. Lets say we have max size 10. And after deleting an element from D-array in class, the current size becomes 2. Now its lets than 1/4$^{th}$ of the max size 10. In this case, when this resize function will be called, you will reduce the D-array max size in vector to class to half. In this scenario, it will get 5.

- ***void push_back(T element):*** When this function will be called, the provided element in parameters will be placed at the first empty index of D-array inside vector class. If the current size is 0 then place the parameter at first index. If current size is 1 then push_back will place the next element at index 1.

***If array inside vector class gets full, You should use resize function to double up the size***

- ***T pop_back():*** This function will clear the element from the last filled index of array in vector class and return it. After poping out the element, place the default value of the Data type at that last index from where the element is poped out. Lets say we have vector of element : 1 2 3 4 5. When pop_back( ) function will be called, the vector will be: 1 2 3 4 0. We placed zero at the place of 5 and returned 5 from this function. Here 0 is placed because vector here is int type. If vector was of string type, we would have placed empty string here.

***If array inside vector class becomes 1/4th of its max size, you should use resize function to half down the size***

***Attention! If user calls pop_back function when array is empty then simply return the default value.***

- ***bool is_empty()*:** return whether the the array in vector is empty or not.
- ***bool is_full():*** return whether the the array in vector is full or not.
- ***T at(int index):*** return the element at that index.
- ***T erase(index):*** remove the element at that index and returns that element.

***i.e If array was (3,4,5,6,7) and we are removing element at index 2. the updated (3,4,6,7). And it will return 5.***

***Attention! Handle out of the bound exception***

- ***Swap :*** Exchanges the contents of the vector with those of other.
- ***Insert:*** Insert a element at specific Index. i.e if array was (3,4,5,6,7) and we insert an element 9 at index 3 then the updated array will be (3,4,5,9,6,7).
- ***Sort***: Sort the entire D-array in vector using bubble sort.
- **Move (in num):** This function will move the elements of D-array (forward or backwards) in vector num number of times. For example, if the num value is 3 and array content is: ***1 2 3 4 5***, then after move function, content will be: ***3 4 5 1 2***.
  Also, num can be negative. In that case, we will move the content of array in backward direction. Let say num is -1 then content of array from 1 2 3 4 5 will be: 2 3 4 5 1.

- ***Union :*** This function will take union of both vectors and return a new vector.
- ***Intersection :*** This function will take intersection of both vectors and stores in new vectors and returns it.
- ***Unique :*** This function will store all the unique element into new vector and returns that vector.

***Elements in v1 are***
***1,2,2,2,3,4,4,4,5***
***So new vector will have***
***1,2,3,4,5***

- ***Int Count(T element) :*** This will return the count of occurrence of desired element in a vector.

***Elements of v1 are***
***0,1,2,3,4,5,6,5,5,5***
***Then count (5) will return 4***

- ***Bool Subset :*** This function will check whether the second vector is subset of first vector.

- ***Bool Search :*** This will search the desired key in the vector and return a bool.

**Overload the following Operators:**

- ***+ :*** This operator will add elements of first vector into new vector then elements of second vector. It will return the new vector .

***If vector1 has elements 0,1,2,4,5***
***and vector2 has elements 1,2,3,4,5,6,7,8***
***Then vector 3 will have 0,1,2,4,5,1,2,3,4,5,6,7,8***

- ***[ index ] :*** This operator ( [ ] ) will return reference to the element at desired index.

***Index must be greater than zero***
***and less than current size.***

- ***- :*** This operator will only store elements of first vector into new vector which are not present in second vector. This will return a new vector.
- ***<< :*** This will display all the elements in the D-Array.
- ***>> :*** This operator will be used to take input D-Array.
- ***== :*** This operator will return true if both vectors have equal size and each element is same in both D-Arrays.

- **>** : This operator will return true if size of first vector is greater than second.

***Suppose if v1 has elements 0,1,2,3 and v2 has elements 0,2,3,4,5,6,7 then size of v1 is is 4 and v2 is 7 And 4<7 so return false;***

- **<** : This operator will return true if size of first vector is less than second .
- **=** : This operator will assign the contents of one second object to first vector object, making a deep copy of the elements.

## *Main Function:*

After defining all the functions, implement a menu-driven main function to comprehensively test each function's and operative's functionality.

……

***Congratulations! You have reached the end of your assignment.Well done.Keep up the good work.***