# ASSIGNMENT #4

## Purpose

The purpose of this assignment is to practice using pointers, manage memory and work substantially with arrays in an application.

## Instructions

In this lab you will continue creating a Todo List application using the **`TodoItem`** class created in the last assignment. You will add a **`TodoList`** class that will create dynamic instances of **`TodoItem`**s.

## Class - TodoList

This is the data manager class. Contains a *dynamic* array of *dynamic* **`TodoItem`**s

**Private Data Members:**
A dynamic array of **`TodoItem*`**
An integer to hold the maximum capacity of your list
An integer to hold the current size of your list

**Constructor:**
Creates a dynamic array of 25 elements and initializes the elements to **`NULL`**

**Destructor:**
Frees the memory for all **`TodoItem`**s
Frees the memory for the dynamic **`TodoItem*`** array

**Member Function 1:**
Named **`AddItem`**. Has one parameter, a dynamic instance of **`TodoItem`**. If there is room in the array add the new dynamic instance to the first available spot (i.e. the current size). If the array is full, increase capacity by 10 and then add the item.

**Member Function 2:**
Named **`DeleteItem`**. Has one parameter, an integer of the location to delete. Please note the location is in human-readable form, i.e. location 1 is really array index 0. After you delete the item you will need to pack your array (shift all items "down" so there are no empty slots between items).

**Member Function 3:**
Named **`GetItem`**. Has one parameter, an integer of the location to retrieve the **`TodoItem`**. Please note the location is in human-readable form, i.e. location 1 is really array index 0. This function will return a pointer to the **`TodoItem`** requested.

| | |
|---|---|
| **Member Function 4:** | Named **`GetSize`**. Returns an integer containing the current size of the list (number of items present). |
| **Member Function 5:** | Named **`GetCapacity`**. Returns an integer containing the current maximum capacity of the list (number of slots). |
| **Member Function 6:** | Named **`Sort`**. Sorts the array by the priorities of the items. (1 is highest priority, 5 is lowest). |
| **Member Function 7:** | Named **`ToFile`**. Returns a string containing all **`TodoItem`**s in the list. Uses the **`TodoItem`**s **`ToFile`** function to create. Each item should be on its own line. |
| **Overloaded Friend operator<<** | Outputs a numbered list of all **`TodoItem`** present in the list. Will use **`TodoItem overloaded<<`** |
| **Private Member Function 1:** | Increases the capacity of the array by 10. Should be called by **`AddItem`** at the appropriate time. |
| **Private Member Function 2:** | Compacts the array to get rid of an empty spot in the array. Should be called by **`DeleteItem`** at the appropriate time. |

## Objectives

- Properly manage memory
- Use pointers and their related syntax
- Create an properly manage a dynamic array and object
- Write proper destructors and use them to manage dynamically allocated arrays

## Requirements

Your code must follow the styling and documenting guidelines presented in class. Please note that I do not give points for style and documentation. You can only lose points. Please make sure your source code is documented correctly and is neatly and consistently formatted using guidelines provided in class.

Your program must provide the features described above:

**[100 pts]** - Fully complete **`TodoList`**

***IF YOUR PROGRAM DOES NOT COMPILE YOU WILL RECEIVE A ZERO!!!***

***Warnings are treated as non-compile. Use g++ flag –Werror***

## Deliverables (via Blackboard)

Your files need to be uploaded/attached to the Assignment Submission on Blackboard. You should upload the following four files:

- **`todo_item.h`**
- **`todo_item.cpp`**
- **`todo_list.h`**
- **`todo_list.cpp`**