Jack Thake

Karla Fant

CS163

10.18.20

# Program Two Design

This program will hold two classes used to store and keep track of a list of alert notifications. The client will communicate with the class *CS_alert_stack,* which itself manages a flexible array (a linear linked list of arrays). Each node in the list is represented by the data structure *node,* each node holds an array of alerts. Each array item will contain the *CS_alert* class which stores and manages the information regarding a single alert. *CS_alert_stack* itself will act like a stack, only allowing the client to push to the top, and only pop or peak the top as well. Components of this program will be rigorously tested for memory leaks, or errors using GNU's Valgrind terminal application.

## Error / Success Reporting

Similar to my last program, error and success in this ADT will be represented by one enum, CS_error. This enum will contain the following values to denote success/error states:

- **SUCCESS** - Operation was successful.
- **FAILURE** - Operation failed.
- **MEM_ALLOC_FAIL** - Memory failed to be allocated.

## The CS_alert_stack Class

The *CS_alert_stack* class will contain a flexible array as the core of the class, this will hold all *CS_alert* class objects. Its functionality will solely revolve around interacting with, and managing this stack. It will contain the following data members:

- **Top -** This will hold the next index in the flexible array to add to, this value can be decremented by one to get to the most recent value.
- **Stack -** This will contain the stack itself, it will point to the active array in the stack if the pointer points to NULL then the stack is empty.

The class will also contain the following functionality to aid in working with these members:

- **Constructor** - Set the stack pointer to NULL.
- **Destructor** - Will handle deallocating the stack and each alert.
- **Push** - When passed a valid alert, it will push said alert to the top of the stack, adding at top's value. If the value of *top* is greater than the length of the current array, a new node is added to the stack, and top is changed accordingly.
- **Pop -** Will remove an item from the top of the stack. This will then decrement top, if *top* becomes negative then the current node is removed from the stack, and *top* is changed accordingly.
- **Peak** - Will return the alert at the top of the stack, not allowing the client to modify the stack.
- **Display** - Will display the entire stack to the user, making use of *CS_alert* class's display function.

## The node structure

The *node* structure represents one item in the flexible array, and will contain the following data members:

- **Next** - This will be a pointer to the next item in the list, if the pointer is NULL then the node is the last in the list.
- **Data** - This will hold the array of alerts.

## The CS_alert Class

The *CS_alert* class will simply just allocate its memory, manage it and then destroy it at an appropriate time. It will contain the following private data members:

- **Organization** - The organization the alert originates from (string)
- **Date** - The date of the alert in "day/month/year" format (string)
- **Time** - The time of the alert in "hour:minute AM/PM" format (string)
- **Message** - The actual alert (string)

The class will also contain the following functionality to aid in working with these members:

- **Constructor** - Gets passed all info about the alert, allocates accordingly.
- **Destructor** - Safely deallocates all of the strings.
- **Display** - Ease of use function to display the alert in a formatted fashion.
- **Overloaded Operated =** - Used to make assigning *CS_alert* objects to each other much easier, must be implemented to make memory leaks or errors impossible.
- **Overloaded Operator ==** - Used to make checking if to *CS_alert* objects are equal easier.