**Assignment 9 – Putting it All Together**

Submit to MUOnline as a compressed (.zip) file containing your code project.

The purpose of this final assignment is to test all you have learned of programming in C++ by making a common and useful tool: an activity planner.

**Requirements**

Create a program that allows a user to store information on up to 10 events in an array. It should use an int ‘numEvents’ to keep track of how many events have been entered in the array.

An event should be represented by a struct with several member variables. An event should include:

Strings for a name, description, and notes.

A date and time represented by their own structs. The Date struct should contain an int day, string month, and int year, and the Time struct should contain an int hour, int minute, int second, and bool am.

A boolean ‘isAppointment’ to tell whether it is an appointment or an activity. An appointment would be something like ‘Go to Doctor’ and the date/time are when it occurs, and an activity would be something like ‘Finish 163 Final’ and the date/time would be when it is due. The purpose of this boolean is to format your output appropriately.

A boolean ‘active’ to mark an event as being a current active event entered by the user. If active is false, it can be considered removed from the list of events and it can be overwritten when a new event is added. When the program begins, all the events in the array should have ‘active’ set to false.

The program should display any ‘active’ events currently in memory as a list, showing their name, date and time (so it would show nothing at first).

Then it should provide the user with the following menu of options (at least) and allow them to continue choosing options until they choose to quit. Each time they choose an option and complete its operation, it should show them the updated list of events and the menu again.

1. View Event – Asks the user for the number of an event to view. It then shows the user the details including the name, date, time, notes, and description.
2. Add Event – Makes sure that there are not already 10 events stored in the array, then allows the user to enter information for a new event by finding the first inactive event in the array and setting its data. It should set the event’s ‘active’ boolean to true.
3. Edit Event – Asks the user for the number of an event to edit. It then allows the user to input new data for the event, overwriting the existing data.
4. Delete Event – Asks the user for the number of an event to delete. It then marks that event as having been deleted by setting its ‘active’ variable to false and decreases the number of events by 1. That way the next time Add Event is called, the deleted event can be overwritten with new data.
5. Exit – Ends the program.

Your program should also be neatly divided into functions. For instance, you should have a DisplayMenu function, and each menu option (except Exit) should call a function which performs that task.

The program **should** **validate user input** such as invalid menu choices, text when numbers are expected, etc. Also note that certain menu choices cannot be used if no events have been added to the list.

Finally, sorting the events is not required.

**Hint:** Probably the trickiest part to handle is this situation:

You have 3 events, event A, B, and C.

You delete event B

You add event D.

Where does it go? It should overwrite the event where event B used to be; otherwise if you only add them to the end of the array of Events, you will run out of space even if you deleted some and have fewer than 10 events. Try to work out how you will handle this (drawing it on paper helps a lot!) before you get too far programming…

**For up to 10% extra credit**, add 2 more options: Save Planner and Load Planner. Save Planner should save the events the user has created to a text file, and load planner should load events in from a file.

**For up to 10% MORE extra credit**, instead of making Event a struct, make it a class that properly hides the variables (as ‘private’ variables), and uses public getter/setter functions to access and validate them. This is extra credit for a few reasons: it’s the very last chapter we cover in 163 and we cover it a lot more in 236, sometimes we don’t get to it depending on snow/sick days and other circumstances, and finally you all could use some extra credit anyway…

**Example of the program running:**

(assume 2 events have already been entered)

You have 2 events coming up:

1. Lunch with Bob on April 25 2019 12:00:00PM
2. Assignment 7 due April 26 2019 11:59:59PM

Would you like to:

1. View an Event
2. Add an Event
3. Edit an Event
4. Delete an Event
5. Exit

2

Add an Event – Enter the name of the new event:

Pickup Alice at airport

Enter a description:

Alice needs picking up at Yeager International.

Enter any extra notes about the event:

Use Rt. 60 – construction on the interstate.

Enter a time in the form Hour Minute Second AM/PM

1 00 00 PM

Enter a date in the form mm dd yyyy

May 4 2019

Is this event an appointment? (Y/N)

y

Event added. Returning to Menu.

You have 3 events coming up:

1. Lunch with Bob on April 25 2019 12:00:00PM
2. Assignment 7 due April 26 2019 11:59:59PM
3. Pickup Alice at airport on May 4 2019 1:00:00PM

Would you like to:

1. View an Event
2. Add an Event
3. Edit an Event
4. Delete an Event
5. Exit

5

Press any key to continue…