To-Do List

Background Information

These days, it seems as if there are more "listing" applications to help you keep track of your daily tasks than seems necessary. All of them operate on the same premise, even if they implement that premise differently. For this assignment, you'll be making yet another "listing" application. Your goal is to write an interactive console program that allows users to organize the tasks on their to-do list in one easy to use application. Your users will be able to do simple actions like add, remove, and complete tasks. They will also be able save their tasks to a file in order to load them during subsequent application launches.

Tasks

All **generic tasks** have a **description** and a **due date**. There are also more specialized kinds of tasks. A **shopping task** contains a **list of items** that needs to be purchased. An **event** has a **location** and a **time**. A **homework task** has a **class subject** associated with it.

Start-up & Commands

Your application is a command based program in the console. This means when the user starts your application, your main function will give a welcome message and wait for the user to provide one of the commands below (the commands should be case insensitive). If a user enters an invalid command you should let them know and wait for the next command. A "HELP" command which would print all commands would certainly be useful but not a requirement for this project.

Your application will not take in any command line arguments and will start off with an empty list of tasks.

COMMAND	DESCRIPTION
ADD	This command creates a new task. All newly created tasks are considered "outstanding" (not completed) to start. The user specifies the task's type, due date, and description. If a specialized task requires additional information, the user provides that as well. For the sake of simplification, assume that an event task's time is a simple, un-validated string. Examples `> ADD` `What type of task is this? [G: Generic, S: Shopping, E: Event, H: Homework]` `> G` `When is this task due?` `> 10/23/2016` `How would you describe this task?` `> Take out the trash` `Task added successfully.`

	`> ADD` `What type of task is this? [G: Generic, S: Shopping, E: Event, H: Homework]` `> S` `When is this task due?` `> 10/30/2016` `How would you describe this task?` `> Staten Island Mall Day` `What items do you need to buy? [Type your item and press ENTER to add another item. Type DONE to complete the list.]` `> Tank Tops for the Beach` `> Boat Shoes` `> 5 iPhone Cases` `> DONE` `Task added successfully.` `> ADD` `What type of task is this? [G: Generic, S: Shopping, E: Event, H: Homework]` `> E` `When is this task due?` `> 10/24/2016` `How would you describe this task?` `> Halloween Party` `Where is this event taking place?` `> Mike's House` `When is this event taking place?` `> 10:00PM` `Task added successfully.` `> ADD` `What type of task is this? [G: Generic, S: Shopping, E: Event, H: Homework]` `> H` `When is this task due?` `> 10/27/2016` `How would you describe this task?` `> Project Due` `What subject is this homework task for?` `> CSCI 235` `Task added successfully.`
PRINT	This command displays all of the outstanding tasks *in order of due date*. If multiple tasks are due the same number of days into the future, they are displayed in alphabetical order of description. Each task is printed on its own line, beginning with its position in the overall list, followed by its due date, type, and description. If there are no outstanding tasks, indicate so to the user. <u>Examples</u> `> PRINT` `You have no outstanding tasks!`

	`` > PRINT 1. 10/23/2016 - Take out the trash 2. 10/24/2016 - [Event] Halloween Party 3. 10/27/2016 - [Homework] Assignment #2 4. 10/30/2016 - [Shopping] Staten Island Mall Day ```
DETAILED	This command does everything that the PRINT command does and also displays any specialized task information. <u>Examples</u> ``` > DETAILED You have no outstanding tasks! > DETAILED 1. 10/23/2016 - Take out the trash 2. 10/24/2016 – [Event] Halloween Party WHERE: Mike's House WHEN: 10:00PM 3. 10/27/2016 - [Homework] Assignment #2 SUBJECT: CSCI 235 4. 10/30/2016 - [Shopping] Staten Island Mall Day ITEMS TO PURCHASE: Tank Tops for the Beach Boat Shoes 5 iPhone Cases ```
REMOVE	This command deletes an existing outstanding task. The user specifies the task number to remove, as that task appears in the lists of the PRINT/DETAILED commands. If there are no outstanding tasks, indicate so to the user. <u>Examples</u> ``` > REMOVE You have no outstanding tasks! > REMOVE Which task would you like to remove? > 2 Task removed successfully. ```
COMPLETE	This command marks a specific outstanding task as complete. The user specifies the task number to mark as complete, as that task appears in the lists of the PRINT/DETAILED commands. If there are no outstanding tasks, indicate so to the user.

	Examples `> COMPLETE` `You have no outstanding tasks!` `> COMPLETE` `Which task would you like to complete?` `> 1` `Task marked complete successfully.`
COMPLETED	This command displays all of the completed tasks. It follows the same format as the PRINT command. If there are no completed tasks, indicate so to the user. Examples `> COMPLETED` `You have no completed tasks!` `> COMPLETED` `1. 10/23/2016 - Take out the trash` `2. 10/27/2016 - [Homework] Assignment #2`
SAVE	This command saves all of the outstanding tasks to a file. The user specifies the name of the file to create. If a file with that name already exists, overwrite it. Note that the contents of the saved file must match the format described later in this specification. Example `> SAVE` `You have no outstanding tasks!` `> SAVE` `Where would you like to save your outstanding tasks?` `> ./my_tasks.data` `Tasks saved successfully!`
LOAD	This command loads all of the tasks from a file as outstanding tasks. Note that the contents of the loaded file must match the format described later in this specification. If there are already tasks in the list, the list will be cleared before the load takes place. Examples `> LOAD` `What file would you like to load outstanding tasks from? (Note: All existing tasks will be deleted)` `> ./my_tasks.data` `Tasks loaded successfully!`
EXIT	This command will exit the program.

	Example `> EXIT` `Thank you for using To-Do List!`

Tasks File Format

A tasks file contains a single task on every line. Each individual component of a task's data is separated by a pipe character ("|") and is ordered as follows—type, due date, description, and any additional specialized task information. Generic tasks, shopping tasks, events, and homework tasks follow the formats, respectively:

```
G|04/10/2017|Put away fans, bring out space heaters
S|04/11/2017|Shopping Day @ Manhattan Mall|Socks|3 Sweaters (Blue, Green &
Red)|Dessert Boots
E|03/12/2017|Networking Event|Hudson Terrace|8:00PM
H|03/01/2017|Homework|AFPRL 100
```

Note that the *ordering* of the file is up to you, try to think about which ordering (if any) would be most efficient for your implementation. Although your ordering may be different, the format of the file should look the same.

Error Handling

For this project, you may NOT assume that any user input provided to your application is valid input (unless specifically mentioned otherwise). For example, a user may try to complete or remove a task that does not exist or may give an invalid file for loading. Note that these aforementioned examples are not all-inclusive; if there is an error situation that you are unsure about, feel free to ask about it on Piazza. All of these error conditions should print a message to the user and allow the user to continue using the application without any side effects. (e.g – if the user tries to remove a task that doesn't exist it should notify the user and leave the rest of the application as if the user never tried to remove that task.) Remember: Crashing is not the same thing and purposely aborting your application (in the event of a fatal error); it means that the code itself has a bug.

Design Requirements

For this assignment, you are not given any starter code you are expected to implement all of the required code yourself. Below are a few things you must adhere to while making your applications:

1. You must create a SortedLinkedList class which you use to store Tasks, you may not use a regular list or a vector.
2. You must create a struct or class to represent a date, this structure will need a few operators overloaded in order for you to do the project. *You may assume that any dates you are given are in the proper MM/DD/YYYY format and are valid dates.*

Things to Think About

From an architectural perspective, there is no one perfect way to do this assignment. Nevertheless, good design choices can make your solution modular, extensible, efficient, and understandable. Endeavor to strike a balance that makes sense.

- What classes do you need to implement all of the required functionality? Is there any inheritance? Which classes interact with the others, and how? Understand has-a vs. is-a relationships.
- Which classes act merely as basic "data containers" and which classes are responsible for the logic of the To-Do List's maintenance? Remember that a single class should handle a single logical set of responsibilities. Don't make any one class do too much heavy lifting. Be modular.
- Remember that your output format and style should match the given output format and style EXACTLY in order for you to receive full credit.
- Don't forget about memory management, you should always have a "delete" for every time you call "new" to allocate memory.

Grading

The assignment grade is broken down as per the project grading rubric which is on Piazza. Please take a look at that for submission instructions and expectations.

Due Date

The due date for this project is April 4th, 2017 at 11:59PM. Remember, no late submissions are accepted.