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 to undo and redo their actions, and save their tasks to a file in order to load them during subsequent application launches.

Tasks

All **generic tasks** have a **description** and a **deadline**. 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.

Commands

Your application accepts the following set of commands:

COMMAND	DESCRIPTION

ADD	This command creates a new task. All newly created tasks are considered "outstanding" to start. The user specifies the task's type, deadline, and description. If a specialized task requires additional information, the user provides that as well. For the sake of simplification, assume that a task's deadline is simple integer representing the number of days into the future when the task is due and assume that an event's time is a simple, unvalidated string.
	
	Examples
	`> ADD`
	`What type of task is this? [G: Generic, S: Shopping, E: Event, H: Homework]`
	`> G`
	`In how many days is this task due?`
	`> 1`
	`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`
	`In how many days is this task due?`
	`> 7`
	`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 In how many days is this task due? > 2 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 In how many days is this task due? > 4 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 deadline. 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 deadline, 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. (1 day from now) Take out the trash 2. (2 days from now) [Event] Halloween Party 3. (4 days from now) [Homework] Assignment #2 4. (7 days from now) [Shopping] Staten Island Mall Day

DETAILED	This command does everything that the PRINT command does and also displays any specialized task information.
	
	Examples
	`> DETAILED`
	`You have no outstanding tasks!`
	
	`> DETAILED`
	`1. (1 day from now) Take out the trash`
	`2. (2 days from now) [Event] Halloween Party`
	`WHERE: Mike's House`
	`WHEN: 10:00PM`
	`3. (4 days from now) [Homework] Assignment #2`
	`SUBJECT: CSCI 235`
	`4. (7 days from now) [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.
	
	Examples
	`> 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. (1 day from now) Take out the trash`
	`2. (4 days from now) [Homework] Assignment #2`

UNDO	This command undoes the last ADD, REMOVE, or COMPLETE action. The user can undo actions multiple times, so long as there are actions left to undo. If there is nothing left to undo, indicate so to the user.
	
	Examples
	`> UNDO`
	`You have no commands to undo!`
	
	`> UNDO`
	`ADD command undone successfully.`
	
	`> UNDO`
	`REMOVE command undone successfully.`
	
	`> UNDO`
	`COMPLETE command undone successfully.`

REDO	This command redoes an action that was previously undone. Note that if an action is undone, but then the user subsequently performs an ADD/COMPLETE/REMOVE action, the user will no longer be able to redo the undone action. If there is nothing to redo, indicate so to the user.
	
	Examples
	`> REDO`
	`You have no commands to redo!`
	
	`> REDO`
	`ADD command redone successfully.`
	
	`> REDO`
	`ADD command redone successfully.`
	
	`> REDO`
	`ADD command redone successfully.`

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.
	
	Examples
	`> LOAD`
	`What file would you like to load outstanding tasks from?`
	`> ./my_tasks.data`
	`Tasks loaded successfully!`

EXIT	This command will exit the program.
	
	Example
	`> EXIT`
	`Thank you for using To-Do List!`

HELP	This command displays all of the available commands and their descriptions.
	
	Example
	`> HELP`
	`ADD – Adds a new task to your To-Do list`
	`PRINT – Displays your outstanding tasks succinctly`
	`DETAILED – Displays your outstanding tasks with specialized task details`
	`DELETE – Deletes an outstanding task`
	`COMPLETE – Marks an outstanding task as complete`
	`COMPLETED – Displays your completed tasks`
	`UNDO – Undoes the last ADD, DELETE, or COMPLETE command`
	`REDO – Redoes the last undone command`
	`SAVE – Saves your outstanding tasks to a file`
	`LOAD – Loads a file of outstanding tasks`
	`EXIT – Exits the application immediately`

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, deadline, description, and any additional specialized task information. Generic tasks, shopping tasks, events, and homework tasks follow the formats, respectively:

```
G|4|Put away fans, bring out space heaters
S|3|Shopping Day @ Manhattan Mall|Socks|3 Sweaters (Blue, Green & Red)|Dessert Boots
E|5|Networking Event|Hudson Terrace|8:00PM
H|6|Homework|AFPRL 100
```

Error Handling

For this project, you may assume that any user input provided to your application is valid input. For example, a user will not try to complete or remove a task that does not exist. A user will not give an invalid file for loading. Your program will never experience "duplicate" tasks (same task type and description). 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. You are not required to implement code that handles these input error conditions; handle them however you'd like. For example, your program can intentionally abort on invalid input. Doing so will not affect your grade. Nevertheless, you may find it helpful to implement some error handling to make testing/debugging less time consuming and more manageable. If your program aborts every time you mistype, it can be painful when your testing depends on specific program state. Note that aborting the program is an intentional action in the code. Crashing is not the same thing; it means that the code itself has a bug.

Program Completion Phases

This assignment will be completed in two phases.

Program Design. You are responsible for looking through this specification and creating a design proposal outlining the classes you expect to have, the precise functionality that these classes provide, the data members/structures that these classes require, and how your classes interact with one another. Be sure to note whether any classes have inheritance relationships. Your design explanation should feature little to no code. It should instead focus on the macro-elements of your proposed program. You may include UML diagrams. After I create your personal assignment repository, I will open up a **GitHub Issue** in the repository asking you to define the design of your project. You will give your design proposal there. See the Grading section to understand how this will be graded.

Program Implementation. After you submit your design proposal, I will give you feedback on your design choices. Furthermore, I will give you a specific design to implement. You will not be implementing your proposed design. Keep in mind that the design given to you is perhaps not the "perfect" solution. Your design proposal may have been equally valid. Nevertheless, you should be able to implement any design given to you. You are encouraged to ask questions about why particular design decisions were made and how they compare to the decisions you made in your design proposal.

Provided Files

For this assignment, you are not given any starter code. Based on the design given to you after you submit your design proposal, you will be expected to implement all of the required code yourself. However, you will be given an executable file that is a working solution based on this specification. Because the executable's solution code was compiled on the Linux lab machines, the executable can only be run on the Linux machines with confidence. Running it on any other system may yield undefined behavior.

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. When coming up with a program design proposal, think about the following.

- 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 data structures do you need to make use of? Do you need a vector or a linked list to store tasks? A sorted vector or sorted linked list? Perhaps this project needs a stack or a queue?
- 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.
- How do you implement your undo functionality? How does redo differ? What do you need to keep track of in order to undo or redo the commands?

When implementing your final solution, think about the following.

- Remember that your output format and style should match the given output format and style EXACTLY in order for you to receive full credit. Be sure to test your running program alongside the provided executable to make sure that yours conforms.
- Memory management can become quite challenging in this project. When can you say that you are "done" with a task? When should you create and delete tasks to prevent memory leaks?
- Remember that the design you are given is very minimal. What helper functions will you need to help you write some of those functions?
- How do you implement the save and load functionality?

Submission Details

Your submission must include all of the header/source files (*.h/*.cpp) required for your program to properly compile and run. You must include a makefile that includes the sources to be compiled. The name of the generated executable must be **ToDoList**. Feel free to adapt the makefile from Assignment #1. Do not submit any generated executables or object files (*.o). You must also update the README file with any guidance that someone looking at your project needs to and might like to know; this could include compilation instructions, interface specifications, interactions between classes, problems overcome, etc. Confirm that your code successfully compiles and runs on the Linux lab machines. See

the **Assignment Guide Using Git & GitHub** for step-by-step information about how to submit your assignment via git and GitHub. Be sure to ask questions on the Q&A board if you have any issues.

Due Dates

There are two due dates for this assignment. The due date for the program design proposal is Wednesday, October 19th, before the start of our class (5:35pm). The later you submit your proposal, the later you will receive feedback from me. If you start coding before I give you feedback, your implementation may suffer as a result. The due date for the implementation portion of the assignment is Wednesday, November 2nd, before the start of our class (5:35pm). These are hard deadlines, meaning that there will be little leniency with regard to lateness. For every day that you miss the deadline for the implementation portion, I will deduct ten points from the final grade. A sample solution will be provided some time after the deadline.

Grading

Grades are comprised of the following:

Components	Percentage	Relevant Questions
Correctness	60%	Do each of your commands provide output as expected based on the input?
Design	15%	Does your program design proposal separate out logical units into classes? Does it employ data structures that make sense and/or solve your problems efficiently?
Documentation	15%	Does your README document provide users with adequate information about your program? Do you adequately comment your class interface files and class implementation files (when necessary)?
Style	10%	Is your coding style readable and consistent? Do you follow (to the best of your ability) the modified Google Style Guide?

If your program does not compile on the Linux machines, you will receive no points for the **Correctness** component. The **Design** component is based on the robustness and clarity of your program design proposal. If you submit your proposal after October 19th, you will receive at most half credit for the **Design** component.

Final Words

This assignment might seem overwhelming! It may feel like the whole Republic of CSCI 235 is conspiring against you. Don't feel alone in this endeavor, my padawans. Feel free to discuss design with your fellow students. But please don't be a Rebel; plagiarism is not acceptable. Start early. Good luck, and may the Force be with you.