Tutorial Problem Set #4

Due: Wednesday, October 9, 2024, 11:59 PM

Policy

- Piazza questions on tutorial problems will be ignored or deleted. Questions will only be answered in your assigned tutorial section.
- Sample executables can be found in your 1249 git repository directory (run `git pull`).
- Completing the problem set will reduce the weight of the final exam by 0.5%. To complete a problem set, you must pass at least 50% of the secret tests.
- You may assume all input is valid. Tutorial problems **NEVER** require checking for invalid inputs.

Question 1

In this exercise, you will write some of the methods of a class that manages tasks in a To-do list. The To-do list can have a maximum of 5 tasks and each task in the list can have a maximum of 5 sub-tasks. There is always a top-level `ToDo` task, representing the list. The list below is an example of the formatting printed out by the program.

```
ToDo
1. [] Do groceries
2. [] Do laundry
3. [] Finish final project
  4. [] Code Module1
  5. [] Code Module2
  6. [] Run tests
  7. [] Write report
8. [] Call Mom
```

We have provided the declaration and definition of classes `List` and `Task`, except for the definition of `Task's` copy constructor, copy assignment operator, move constructor, and move assignment operator in `task-impl.cc`. You must implement these methods so task objects can be copied, moved, and pasted in the list. **You may not change the contents of `task.cc` other than by adding private instance members and comments, i.e., the public interface must stay exactly the same. You should not modify any of the imports.**

```
class Task {
    std::string description;   // the description of the Task
    int id;                    // the id attribute of the Task
    bool done;                 // the done status of the Task
    Task *parent;              // pointer to the parent Task
                               // (or nullptr for the ToDo Task,
                               // which does not have a parent)
    Task **children;           // array of children Task pointers
    int childrenLength;        // current number of children Tasks
    // please check task.cc for details
};
```

Note that every task has a pointer to its `parent`, and an array of pointers to its `children` (subtasks). Remember to handle those when implementing the copy/move constructors and copy/move assignment operators.

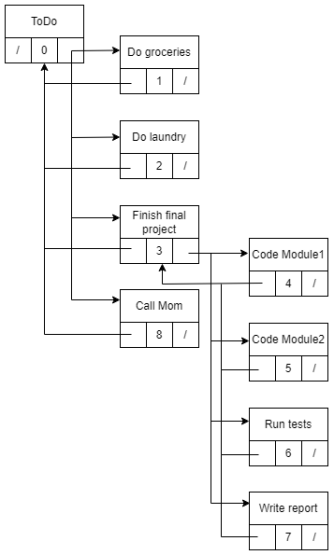

Figure 1:

The diagram in Figure 1 shows an overview of the tasks composing the example document and the pointers to their respective parent and children tasks.

A test harness is provided in `todo.cc` (which uses the module `list`). You can use it to test your `Task` class. **The test harness is not robust and you are not to devise tests for it, just for the `Task` class. Do not change these files.** We have also provided a sample executable (`todo`) that you can use for tests.

The test harness supports the following commands. Please read the contents of `todo.cc`, `list.cc`, and `list-impl.cc` if you need more details about them. Note that the harness keeps track of the "current" task at any moment and most of the commands operate on it. Also, note that the harness automatically creates a top-level `todo` task. You can add children to it. **However, you cannot delete, cut, or copy the top-level `todo` task.**

Command	**Description**
`print`	Prints the current task to the standard output.
`add` *desc*	Creates a new task with *description* specified by the argument, adds it as a child of the current task, and makes it the new current task.
`delete`	Deletes the current task, removes it from its parent, and makes its parent the new current task.
`parent`	Sets the parent of the current task as the new current task.
`up`	Sets the topmost task (`<document>`) as the current task.
`desc` *desc*	Sets the `description` attribute of the current task to *desc*.
`done`	Sets the `done` attribute of the current task to true.
`undone`	Sets the `done` attribute of the current task to false.
`find` *desc*	Finds the first task that is a child of the current task and has the specified description. If such a task is found, it is set as the current task. Otherwise, it prints an error message.
`findId` *id*	Finds the first task that is a child of the current task and has the specified id. If such a task is found, it is set as the current task. Otherwise, it prints an error message.
`cut`	Removes the current task from its parent and moves it to the program's internal clipboard. Sets its parent task as the current task.
`copy`	Copies the current task to the program's internal clipboard.
`paste`	Adds a new child to the current task, whose contents are a copy of the task currently in the clipboard, and updates the pointers of the child and current task accordingly.
`quit`	Quits the program.

Submission

Submit a zip file called `tutorial4.zip` containing `task.cc` and `task-impl.cc` to Marmoset.