CS 3005: Programming in C++

Menu System

Introduction

We want to have a text based menu system that allows the user to execute commands from a list of commands. Each command will manipulate the data stored in the program.

Assignment

In this assignment, you will add to the `ApplicationData` class, and create some new classes that support a menu system. You will also build a very simple program to test this menu system.

The commands this program will have are listed in this table.

Command	Prefixable?	Function	Description
`help`	no	`menuUI`	Display help message.
`menu`	no	`menuUI`	Display help message.
`#`	yes	`commentUI`	Skip to end of line (comment).
`comment`	no	`commentUI`	Skip to end of line (comment).
`echo`	no	`echoUI`	Echo back the arguments given.
`quit`	no	`quitUI`	Terminate the program.

- *Command* indicates the command the user types to request the command.
- *Prefixable?* indicates if the command should be searched for as a prefix (the first characters) of the command typed.
- *Function* indicates the function that the command will call when it is executed.
- *Description* indicates the description of the command.

Example Session

```
$ ./program-menu-test/menu_test
Choice? # this is a comment, it does nothing.
Choice? #this is a comment, where the command name is a prefix of the command typed.
Choice? comment this is a comment using the command name, not #
Choice? commentthis is a bad comment, because the "comment" command can not be a prefix.
Unknown action 'commentthis'. Use 'help' for a list of valid actions
Choice? echo Hello world!
 Hello world!
Choice? echo "Hello world!"
 "Hello world!"
Choice? echo copies everything after the echo command to the output.
 copies everything after the echo command to the output.
Choice? echo help will show the list of available commands
 help will show the list of available commands
Choice? help
Options are:
  # - Skip to end of line (comment).
  comment - Skip to end of line (comment).
  echo - Echo back the arguments given.
  help - Display help message.
  menu - Display help message.
  quit - Terminate the program.

Choice? echo menu does too
 menu does too
Choice? menu
Options are:
  # - Skip to end of line (comment).
  comment - Skip to end of line (comment).
  echo - Echo back the arguments given.
  help - Display help message.
  menu - Display help message.
  quit - Terminate the program.
```

```
Choice? quit
```

Programming Requirements

Create `library-application/ActionFunctionData.{h,cpp}`

`ActionFunctionData` Class

This class represents one command, or action, that the user can execute. Think of it as a row in the table of commands listed above.

`public` Typedefs:

- `typedef void (*ActionFunctionType)(ApplicationData&);` Creates the `ActionFunctionType`.

Data Members:

The following data members should be `private` or `protected`. `public` data members are not allowed.

- `std::string` name for the command. This is the command the user types to execute it.
- `ActionFunctionType` the function that is called when the user types the command.
- `std::string` the command description given to the user when they ask for help.
- `int` the length of the prefix. `0` if the command is not allowed to be a prefix.

`public` Methods:

- `ActionFunctionData();` Default constructor, initializes all data members to `""` and `0` as appropriate.
- `ActionFunctionData(const std::string& name, const ActionFunctionType& function, const std::string& description);` Initializes all data members to the given arguments, expect `prefix_length` which is set to `0`.
- `ActionFunctionData(const std::string& name, const ActionFunctionType& function, const std::string& description, const int prefix_length);` Initializes all data members to the given argument.
- `const std::string& getName() const;` Returns the data member.
- `ActionFunctionType getFunction() const;` Returns the data member.
- `const std::string& getDescription() const;` Returns the data member.
- `int getPrefixLength() const;` Returns the data member.
- `void setName(const std::string& name);` Sets the data member.
- `void setFunction(ActionFunctionType function);` Sets the data member.
- `void setDescription(const std::string& description);` Sets the data member.
- `void setPrefixLength(const int length);` Sets the data member.

Update `library-application/Makefile`

Add `ActionFunctionData.{h,cpp}` in the appropriate places to add them to the library and install the header file.

Create `library-application/MenuData.{h,cpp}`

`MenuData` Class

This class represents all of the commands the user can execute. Think of it as all of the rows in the table of commands listed above.

Data Members:

The following data members should be `private` or `protected`. `public` data members are not allowed.

- `std::map<std::string, ActionFunctionData>` A map of command names to the action function data for the command.

`protected` Methods:

These methods must be protected, not public, and not private.

- `bool actionExistsAux(const std::string& name) const;` If `name` is a key in the map, return true. Otherwise return false.
- `const std::string actionPrefix(const std::string& name) const;` If the key of a command is the first `prefix_length` of `name`, return the key. Otherwise, return the empty string.

`public` Methods:

- `MenuData();` Default constructor. Does not need to initialize any data members. The `std::map` default constructor will automatically build an empty map.
- `void addAction(const ActionFunctionData& function);` If the action already exists, do nothing. If the action does not exist, add it to the map.
- `bool actionExists(const std::string& name) const;` If `name` is a key in the map, or if `name`'s prefix is a key in the map, return true. Otherwise return false. You should be using the protected methods to do most of the work here.
- `const ActionFunctionData& getAction(const std::string& name);` If `name` is an action, return the `ActionFunctionData` associated with it. Otherwise, return a `static` default constructed `ActionFunctionData` object.
- `void printActionHelp(std::ostream& out) const;` Send the `help` message to `out`. Use the form shown in the example session above.

Update `library-application/Makefile`

Add `MenuData.{h,cpp}` in the appropriate places to add them to the library and install the header file.

Update `library-application/ApplicationData.{h,cpp}`

We will update the `ApplicationData` class to support a text menu based application.

`ApplicationData` Class

Data Members:

These additional data members should be in the `private` or `protected` section of the class. No `public` data members are allowed.

- `bool` a variable to track whether the application is done or not. Initialized to false in the constructor.
- `MenuData` a collection of the commands supported by the application. The default constructor for `MenuData` will be called automatically without any new code in the constructor.

`public` Methods:

- Constructor: Initializes the new data members.
- `void addAction(const ActionFunctionData& action);` Adds `action` to the `MenuData` data member.
- `void setDone(bool done);` Sets the new `bool` data member to the value of `done`.
- `void printActionHelp();` Causes the `MenuData` data member's `printActionHelp` method to be called. Uses the `ApplicationData` data member for output stream.
- `void clearToEOL();` Reads from the `ApplicationData` data member for input stream until the end of line (EOL) is found. The EOL can happen in one of two ways. 1) A new line character is read. 2) The end of the stream is reached.
- `void takeAction(const std::string& choice);` If `choice` is the name of an action in the `MenuData` data member, call the `ActionFunctionType` associated with the action. Otherwise, print an error message and clear to end of line. The error message format can be seen in the example session above.
- `void mainLoop();` While the input stream is `good()` and the data member tracking done is `false`, read the user's "Choice? " from the input stream and take the action indicated by the choice.

Create `library-commands/menu_test_aux.{h,cpp}`

Functions:

- `void menuUI(ApplicationData& app_data);` Uses `app_data` to display the action help and clears to EOL.
- `void commentUI(ApplicationData& app_data);` Clears the input to the EOL.
- `void echoUI(ApplicationData& app_data);` Collect all of the characters in the input stream up to the EOL.

Send them to the output stream, followed by a newline character.
- `void quitUI(ApplicationData& app_data);` Sets the `app_data`'s done to be true.
- `int register_menu_test_commands(ApplicationData& app_data);` Adds all of the actions in the table above to the `app_data`.
- `int menu_test(ApplicationData& app_data);` Uses `register_menu_test_commands` to create all of the commands, then calls `app_data.mainLoop()`. Returns 0.

Update `library-commands/Makefile`

Add `menu_test_aux.{h,cpp}` in the appropriate places to add them to the library and install the header file.

Create `program-menu-test/menu_test.cpp`

Functions:

- `int main();` Entry point to the menu test program. Should create an `ApplicationData` and pass it to the `menu_test` function found in `menu_test_aux` and return the result of that function call.

Create `program-menu-test/Makefile`

This file must contain rules such that any of the following commands will build the `menu_test` program:

- `make`
- `make all`
- `make menu_test`

Create `program-menu-test/.gitignore`

The file needs to store one line of text:

```
menu_test
```

This will prevent the executable program from being committed to the repository. It is a *derived file*.

Update `Makefile`

- Update the project-level Makefile so that `make` and `make all` in the project directory will call `make` in the `program-menu-test` directory.
- If necessary, make sure the order of make commands is correct to build prerequisite libraries in the correct order.

Additional Documentation

TBA

Grading Instructions

To receive credit for this assignment:

- your code must be pushed to your repository for this class on GitHub
- all unit tests must pass
- all acceptance tests must pass
- all programs must build, run, and execute as described in the assignment descriptions.

Extra Challenges (Not Required)

TBA