CS 3005: Programming in C++
Image Menu Code Improvements
Assignment

In this assignment you will clean up the code in `image_menu.cpp` and `image_menu.h` to improve its structure.

The first change will be to add a new class `ActionData` to store all of the objects that functions commonly work on, such as the input and output streams, the PPM objects, the grid object, and the color table. This class will be used to replace the parameter lists of most of the functions in `image_menu.{h,cpp}`, especially those called by `takeAction()` or `imageMenu()`.

Potential Session

# There is no specific sample session for this assignment
# If you have completed the work correctly, all previous
# sample sessions will still work the same.

Programming Requirements

**ActionData** class

The `ActionData` class should be declared in `image_menu.h` and implemented in `image_menu.cpp`.

The `ActionData` class needs the following data members:

- `std::istream& mIS;` A reference to the input stream to read user input from.
- `std::ostream& mOS;` A reference to the output stream to display data to user.
- `PPM mInputImage1;` The “input image 1”.
- `PPM mInputImage2;` The “input image 2”.
- `PPM mOutputImage;` The “output image”.
- `NumberGrid *mGrid;` A pointer to the current grid-like object. Should be initialized to 0.
- `ColorTable mTable;` The color table object, should be initialized to the colors as described in the Color Table assignment.
- `bool mDone;` Is the user done or not.

The `ActionData` class needs the following methods:

- `ActionData(std::istream& is, std::ostream& os);` Initialize all data members. The color table should be size 16, and have the gradient from `Color(0,255,0)` to `Color(255,0,255)`. This is the same as was created in `imageMenu` previously. `mDone` should be initialized to `false`.
- `~ActionData();` If `mGrid` is not 0, delete it.
- `std::istream& getIS();` Return the data member.
- `std::ostream& getOS();` Return the data member.
- `PPM& getInputImage1();` Return the data member.
- `PPM& getInputImage2();` Return the data member.
- `PPM& getOutputImage();` Return the data member.
- `NumberGrid& getGrid();` Return `*mGrid`, a reference to the object pointed to by `mGrid`.
- `void setGrid(NumberGrid *grid);` If `mGrid` is not 0, delete it, then assign `mGrid` to `grid`.
- `ColorTable& getTable();` Return the data member.
- `bool getDone() const;` Return the data member.
- `void setDone();` Set `mDone` to `true`.

**ActionFunctionType**

Define `typedef void (*ActionFunctionType)(ActionData& action_data);` in `image_menu.h`.

**MenuData** class

The `MenuData` class should be declared in `image_menu.h` and implemented in `image_menu.cpp`.

The `MenuData` class needs the following data members:

- `std::vector<std::string> mActionNames;` A vector of action names users are allowed to use.
The `MenuData` class needs the following methods:

- **MenuData();** Constructor, no initialization required.
- void addAction(const std::string& name, ActionFunctionType func, const std::string& description); Add name to `mActionNames`, name to `mActionFunctions`, and name: description to `mActionDescriptions`. If name has already been added, do nothing.
- const std::vector<std::string>& getNames() const; Returns the data member.
- ActionFunctionType getFunction(const std::string& name); If name is registered, return the function pointer. Otherwise, return 0.
- const std::string& getDescription(const std::string& name); If name is registered, return the description. Otherwise, throw an exception. The exception should be a `std::string` with the value “Action NAME has no description.”, where NAME is the value of `name`.

### `image_menu` functions

Replace the parameter list for the functions of `image_menu` with `ActionData& action_data`. This will require replacing the previous parameters in the function body with a call to the getter method. For example, `input_image1` would be replaced with `action_data.getInputImage1()`.

A few of the functions also receive the `MenuData` object.

Here is a partial list of the functions to change. This includes some of my solutions to previous exam questions. You don’t have to add them if they aren’t already in your code. But, if they are, update them, don’t delete them.

- `std::string getString( ActionData& action_data, const std::string& prompt );`
- `int getInteger( ActionData& action_data, const std::string& prompt );`
- `double getDouble( ActionData& action_data, const std::string& prompt );`
- `int assignement1( ActionData& action_data );`
- `short getShort( ActionData& action_data, const std::string& prompt );`
- `int inquisitor( ActionData& action_data );`
- `void writeUserImage( ActionData& action_data );`
- `int assignment2( ActionData& action_data );`
- `int four_square( ActionData& action_data );`
- `std::string getChoice( ActionData& action_data );`
- `void commentLine( ActionData& action_data );`
- `void setSize( ActionData& action_data );`
- `voidsetMaxColorValue( ActionData& action_data );`
- `void setChannel( ActionData& action_data );`
- `void setPixel( ActionData& action_data );`
- `void clearAll( ActionData& action_data );`
- `void drawDiamond( ActionData& action_data );`
- `void drawCircle( ActionData& action_data );`
- `void drawBox( ActionData& action_data );`
- `void drawTestPattern( ActionData& action_data );`
- `void drawSquare( ActionData& action_data );`
- `void plus( ActionData& action_data );`
- `void plusEquals( ActionData& action_data );`
- `void minus( ActionData& action_data );`
- `void minusEquals( ActionData& action_data );`
- `void times( ActionData& action_data );`
- `void timesEquals( ActionData& action_data );`
- `void divide( ActionData& action_data );`
- `void divideEquals( ActionData& action_data );`
- `void plusEqualsInt( ActionData& action_data );`
- `void blendImages( ActionData& action_data );`
- `void timesEqualsPPM( ActionData& action_data );`
- `void configureGrid( ActionData& action_data );`
- `void setGrid( ActionData& action_data );`
- `void applyGrid( ActionData& action_data );`
- `void applyGridColorTable( ActionData& action_data );`
- `void setFractalPlaneSize( ActionData& action_data );`
- `void calculateFractal( ActionData& action_data );`
These functions have additional changes:

- `void setJuliaParameters( ActionData& action_data );` This is a copy of `readUserImage()` from before, but operates on `action_data.getInputImage1();`
- `void readUserImage1( ActionData& action_data );` This is a copy of `readUserImage()` from before, but operates on `action_data.getInputImage1();`
- `readUserImage1()` Remove
- `void readUserImage2( ActionData& action_data );` This is a copy of `readUserImage()` from before, but operates on `action_data.getInputImage2();`
- `readUserImage2()`, `setDone()`, and `getChoice()` need to be added to the menu in `configureMenu()`. This is because `setGrid` will delete the old pointer for us. All of these functions need to be added to the menu in `configureMenu()`. This function should be simplified now. For initialization it will complete these steps: It creates an `ActionData` object, using the `is` and `os` passed to it for initialization. It also creates a `MenuData` object, then calls `configureMenu()` on it. It also calls `setGrid` on the `ActionData` object with a `new JuliaSet()` as the parameter. Now the `while` loop will continue while the action data objects is not done and the action data object's input stream object is good (`action_data.getIS().good()`). Inside the loop, call `showMenu()`, `getChoice()` and `takeAction()`. That's it.

These functions are new. They are mostly created to operate on the data members inside `ActionData`. For example `copyImage` will do `action_data.setOutputImage1() = action_data.getInputImage1();` as its function body. These were mostly 1 line statements in the previous `takeAction()` if/else chain.

- `void quit( ActionData& action_data );` Calls `setDone()`
- `void copyImage( ActionData& action_data );` Copies input image 1 to output image.
- `void grayFromRed( ActionData& action_data );` Sets output image to the red gray filter of input image 1.
- `void grayFromGreen( ActionData& action_data );` Sets output image to the green gray filter of input image 1.
- `void grayFromBlue( ActionData& action_data );` Sets output image to the blue gray filter of input image 1.
- `void grayFromLinearColorimetric( ActionData& action_data );` Sets output image to the linear colorimetric gray filter of input image 1.
- `void sepiaFilter( ActionData& action_data );` Sets output image to the sepia filter of input image 1.
- `void tealFilter( ActionData& action_data );` Sets output image to the teal filter of input image 1.
- `void verticalEdgeFilter( ActionData& action_data );` Sets output image to the vertical edge filter of input image 1.
- `void horizontalEdgeFilter( ActionData& action_data );` Sets output image to the horizontal edge filter of input image 1.

All of the `set*Fractal` functions, such as `void setJuliaFractal( ActionData& action_data );` need to be updated. Their body is very simple now: `action_data.setGrid(new JuliaSet());`. This is because `setGrid` will update the old pointer for us. All of these functions need to be added to the menu in `configureMenu()`. `configureMenu()` will be changed internally, but not the parameter list.

`int imageMenu( std::istream& is, std::ostream& os );` This function should be simplified now. For initialization it will complete these steps: It creates an `ActionData` object, using the `is` and `os` passed to it for initialization. It also creates a `MenuData` object, then calls `configureMenu()` on it. It also calls `setGrid` on the `ActionData` object with a `new JuliaSet()` as the parameter. Now the `while` loop will continue while the action data objects is not done and the action data object's input stream object is good (`action_data.getIS().good()`). Inside the loop, call `showMenu()`, `getChoice()` and `takeAction()`. That's it.

**Additional Documentation**

- [C++ Reference](#)
- [Examples from class](#)
Show Off Your Work

To receive credit for this assignment, you must

- complete the unit tests available in CodeGrinder (if available)
- use git to add, commit and push your solution to your repository for this class.

Additionally, the program must build, run and give correct output.

Extra Challenges (Not Required)