# CS 3005: Programming in C++

## Action Data, Menu Data

In previous assignments the functions declared in `image_menu.h` needed to have an input stream and an output stream to interact with the user. `getString()` is an example of this. Other functions also needed an `Image` object or a `PPM` object to work on. `diagonalQuadPattern()` and `writeUserImage()` are examples of these cases. As we add to the complexity of the software project, some of these functions will require additional parameters, and new functions will requires these parameters, and more.

Soon, our project create a program that allows the user to choose the actions it takes, by using a menu system. This menu will allow the user to read PPM images from files, edit them with several image operations, and save the resulting images back to PPM files.

In this assignment, you will build a pair of classes to support the menu system, and the various parameters needed by the action functions in the system. The `ActionData` class will contain the input and output streams, some `PPM` objects, other data necessary to control user interaction, and application status. The `MenuData` class will keep a collection of all actions the user can apply, the functions that implement the actions, and a help message for each action.

There will not be a new executable program at the end of this assignment. But, there will be one soon.

## Assignment

In this assignment, you will add code to the work from the previous assignments. When you have completed the assignment the previous programs will all work the same as they did before. You will also have some unused functions that will make the next assignments easier.

## Programming Requirements

The following files must be updated or created and stored in the `src` directory of your repository.

## Create `ActionData.h` and `ActionData.cpp`

Declare the `ActionData` class in the header file, and implement its methods in the implementation file. Descriptions of the data members and methods follow.

Data Members:

- `std::istream&` The application's input stream. Notice that this data member should be declared as a reference (`&`). See the constructor for instructions to initialize it correctly.
- `std::ostream&` The application's output stream. Notice that this data member should be declared as a reference (`&`). See the constructor for instructions to initialize it correctly.
- `PPM` We will call this the "input image 1". This is the `PPM` object that the user will act on most of the time. For example, if the user wants to set the width of the image, this is the object that will be changed.
- `PPM` We will call this the "input image 2". This is the `PPM` object that the user will act on some of the time. For example, if the user wants to merge two images, this will be the secondary image to merge.
- `PPM` We will call this the "output image". This is the `PPM` object that will be used when the user writes an image to a file. The main way to change this object is for the user to ask for the input image 1 to be copied to it.
- `bool` This data member keeps track of whether the user has asked for the application to be done (quit).

Methods:

- `ActionData(std::istream& is, std::ostream& os);` The constructor initializes the input and output stream data members from the two parameters. The `:` initialization syntax *must* be used for this initialization. Also initializes the Boolean data member to `false`. The `PPM` data members do not need any initialization. Their default constructors will automatically initialize them.
- `std::istream& getIS();` Returns the input stream data member.
- `std::ostream& getOS();` Returns the output stream data member.
- `PPM& getInputImage1();` Returns the input image 1 data member.
- `PPM& getInputImage2();` Returns the input image 2 data member.
- `PPM& getOutputImage();` Returns the output image data member.
- `bool getDone() const;` Returns the Boolean data member.

- `void setDone();` Sets the Boolean data member to `true`.

## Create `MenuData.h` and `MenuData.cpp`

Declare the `MenuData` class in the header file, and implement its methods in the implementation file. Descriptions of the data members and methods follow. This class will be used to keep track of the command names that users can type, the function that will be called for each command name, and a text description of each command's actions.

In the header file, define the `ActionFunctionType` using this `typedef`.

```
typedef void (*ActionFunctionType)(ActionData& action_data);
```

Data Members:

- `std::vector<std::string>` This is a collection of the command names the user can type.
- `std::map<std::string,ActionFunctionType>` This is a map from command name to action function.
- `std::map<std::string,std::string>` This is a map from command name to command description.

Methods:

- `MenuData();` All data members have default constructors that do the right thing, so no initialization of the data members is necessary. The constructor must be implemented, it is just empty.
- `void addAction(const std::string& name, ActionFunctionType func, const std::string& description);` Append `name` to the collection of names, insert `func` in the action function map, with `name` as the key, and insert `description` into the description map with `name` as the key.
- `const std::vector<std::string>& getNames() const;` Return the name collection data member.
- `ActionFunctionType getFunction(const std::string& name);` If `name` is a key in the action function map, return the function associated with it. If `name` is not a key, return `0`.
- `const std::string& getDescription(const std::string& name);` If `name` is a key in the description map, return the description associated with it. If `name` is not a key, return the empty string. Use a `static std::string` variable that is initialized to the empty string.

## Additions to the `PPM` Class

Update the `PPM` header file and implementation file to add the following method used to read images from files.

- `void readStream(std::istream& is);` This method is the reverse of the `writeStream()` method. It will read information from the input stream, and change the contents of the `PPM` object. Every place that `writeStream()` uses `<<` to write an ASCII string, `readStream()` will use `>>` to read an ASCII string into a variable. Everywhere that `writeStream()` uses `.write()` to write a byte of data, `readStream()` will use `.read()` to read a byte. Where `writeStream()` used the `get*()` methods to fetch values from the object, `readStream()` will use `set*()` methods to set them.

## Update `Makefile`

Since `ActionData.{h,cpp}` and `MenuData.{h,cpp}` have been added, you'll need to add rules to build `ActionData.o`, and `MenuData.o`. Since these are not currently used in any programs, they do not need to be added to any dependency lists, or linker commands, yet.

### Build Requirements

- `make ActionData.o` - builds the ActionData.o
- `make MenuData.o` - builds the MenuData.o
- `make hello` - builds the hello program
- `make questions_3` - builds the questions_3 program
- `make ascii_image` - builds the ascii_image program
- `make image_file` - builds the image_file program
- `make` - builds all programs

## Additional Documentation

- C++ Reference
- Examples from class
- Windows PPM Viewer
- Linux PPM Viewer: `eog file.ppm`

# Show Off Your Work

To receive credit for this assignment, you must

- use git to add, commit and push your solution to your repository for this class.
- successfully pass all unit tests and acceptance tests

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