In this assignment you will create a program to read command line arguments to configure and create an image. This program will have all of the functionality of the previous program, but no menu system or prompts to the user, once the program begins.

Potential Session

```
# This command will create a Julia set image, without any menus or prompts
$ ./ppm_command -T 302 -C set:300,255,255,255 -C set:301,12,99,166 -C gradient:0,2,46,80,39,2,46,80 -C gradient:130,104,228,1,59,67,148,0 -C gradient:130,104,228,1,129,67,148,0 -C gradient:130,104,228,1,299,5,63,107 -J -a 0.279047,-0.481944 -p 1000,1000 -m 300 -r 0.412096:0.797398,0.624066:1.00937 -g calculate -g apply-color -w sample-color-julia.ppm
$ ls -l *.ppm
-rw-r--r-- 1 cgl cgl 3000017 Apr 12 12:05 sample-color-julia.ppm

# This shell script has one command that generates many images
$ ./ppm_command_11.bash
$ ls -l *.ppm
-rw-r--r-- 1 cgl cgl 180015 Apr 12 12:06 sample-added-face-bright.ppm
-rw-r--r-- 1 cgl cgl 180015 Apr 12 12:06 sample-added-face.ppm
-rw-r--r-- 1 cgl cgl 180015 Apr 12 12:06 sample-added-head.ppm
-rw-r--r-- 1 cgl cgl 3000017 Apr 12 12:06 sample-color-julia.ppm
-rw-r--r-- 1 cgl cgl 750015 Apr 12 12:06 sample-color-mandelbrot.ppm
-rw-r--r-- 1 cgl cgl 180015 Apr 12 12:06 sample-face-bluegray.ppm
-rw-r--r-- 1 cgl cgl 180015 Apr 12 12:06 sample-face-greengray.ppm
-rw-r--r-- 1 cgl cgl 180015 Apr 12 12:06 sample-face-lineargray.ppm
-rw-r--r-- 1 cgl cgl 180015 Apr 12 12:06 sample-face.ppm
-rw-r--r-- 1 cgl cgl 180015 Apr 12 12:06 sample-face-redgray.ppm
-rw-r--r-- 1 cgl cgl 180015 Apr 12 12:06 sample-head.ppm
-rw-r--r-- 1 cgl cgl 3000016 Apr 12 12:06 sample-joined-julia.ppm
-rw-r--r-- 1 cgl cgl 180015 Apr 12 12:06 sample-left-eye.ppm
-rw-r--r-- 1 cgl cgl 180015 Apr 12 12:06 sample-mouth.ppm
-rw-r--r-- 1 cgl cgl 180015 Apr 12 12:06 sample-nose.ppm
-rw-r--r-- 1 cgl cgl 3000016 Apr 12 12:06 sample-rainbow-julia.ppm
-rw-r--r-- 1 cgl cgl 750014 Apr 12 12:06 sample-rainbow-mandelbrot.ppm
-rw-r--r-- 1 cgl cgl 180015 Apr 12 12:06 sample-right-eye.ppm
# This will show all of the command line arguments available.
$ ./ppm_command -h
```

Options:

PPM options:
- 1 filename ◆ Read file into input image 1.
- 2 filename ◆ Read file into input image 2.
- w filename ◆ Write output image to file.

PPM filter options:
- f copy ◆ Copy input image 1 to output image.
- f red-gray ◆ Set output image from input image 1's grayscale from red.
- f green-gray ◆ Set output image from input image 1's grayscale from green.
- f blue-gray ◆ Set output image from input image 1's grayscale from blue.
- f linear-gray ◆ Set output image from input image 1's grayscale from linear colorimetric.
- f + ◆ Set output image from sum of input image 1 and input image 2.
- f += ◆ Set input image 1 by adding in input image 2.
- f - ◆ Set output image from difference of input image 1 and input image 2.
- f -= ◆ Set input image 1 by subtracting input image 2.
- f *:number ◆ Set output image from input image 1 multiplied by number.
- f *=:number ◆ Set input image 1 by multiplying by number.
- f /:number ◆ Set output image from input image 1 divided by number.
- f /=:number ◆ Set input image 1 by dividing by number.

Draw options:
- d size:height,width ◆ Set the size of input image 1.
Programming Requirements

Create `image_command.h` and `image_command.cpp`

These files will declare and define many functions, which will be used to complete the required actions. All of the functions use an additional type, `CommandData`, that must be created. It should be created as a `struct`, which is the same as a class, where everything is `public`. Additionally, you’ll create function pointers, using the `ActionFunctionType` type, which is declared using a `typedef`. Since `ActionFunctionType` and `CommandData` introduce a circular dependency (each needs the other to be declared), we use a forward declaration of `CommandData`.

Here’s the declaration of the `typedef` and `struct`:

```cpp
// Forward Declaration
struct CommandData;

typedef int (*ActionFunctionType)( CommandData& data );

// this structure just wraps a group of data members together
struct CommandData {
    CommandData( int n, std::istream& i, std::ostream& o );
    PPM input_image1;
    PPM input_image2;
    PPM output_image;
    NumberGrid* grid;
    ColorTable table;
    std::istream& is;
    std::ostream& os;
    std::string argument;
    std::map< std::string, ActionFunctionType > filter_actions;
    std::map< std::string, ActionFunctionType > draw_actions;
    std::map< std::string, ActionFunctionType > grid_actions;
    std::map< std::string, ActionFunctionType > color_actions;
};
```
This is the function list. Most of these functions carry out the same operation as a similar menu option in the menu program. Please refer to that documentation where appropriate. You can also see the expected argument syntax in the sample session above.

Most functions with `int` return type will return `0` if their operation went as expected, and a `1` if not as expected. For example, if the command line argument isn't a correct command or correctly formatted, command, the function would send an error message to `data.os` and return 1.

**Input parsing**

- void `readUntilChar(std::stringstream& ss, std::string& dest, char delimiter)`; This function reads characters from the input stream until the delimiter character is encountered. It stores all of the characters, except the delimiter, into the destination string.

**PPM options**

- int `readImage1(CommandData& data)`; Installed in the option map.
- int `readImage2(CommandData& data)`; Installed in the option map.
- int `writeImage(CommandData& data)`; Installed in the option map.

**PPM filter options**

- int `filterCopy(CommandData& data)`;
- int `filterRedGray(CommandData& data)`;
- int `filterGreenGray(CommandData& data)`;
- int `filterBlueGray(CommandData& data)`;
- int `filterLinearGray(CommandData& data)`;
- int `filterPlus(CommandData& data)`;
- int `filterPlusEquals(CommandData& data)`;
- int `filterMinus(CommandData& data)`;
- int `filterMinusEquals(CommandData& data)`;
- int `filterTimes(CommandData& data)`;
- int `filterTimesEquals(CommandData& data)`;
- int `filterDivide(CommandData& data)`;
- int `filterDivideEquals(CommandData& data)`;
- int `filterAction(CommandData& data)`; Extracts the action word from `data.argument`, using `readUntilChar`. Uses the action word and `data.filter_actions` to call the correct action function. For example, `r = data.filter_actions[action](data);`. Returns the value returned by the function. Unless the action is not in `data.filter_actions`, in which case a 1 is returned. This function itself is installed in the option map.

- void `makeFilterActionMap(std::map<std::string, ActionFunctionType>& actions)`; Loads `actions` with entries for each of the commands as the key and the handler function as the value. For example `actions["+="] = filterPlusEquals;`.

**Draw options**

- int `drawSize(CommandData& data)`;
- int `drawMax(CommandData& data)`;
- int `drawChannel(CommandData& data)`;
- int `drawPixel(CommandData& data)`;
- int `drawClear(CommandData& data)`;
- int `drawDiamond(CommandData& data)`;
- int `drawCircle(CommandData& data)`;
- int `drawBox(CommandData& data)`;
- int `drawAction(CommandData& data)`; Similar to `filterAction()` above.
- void `makeDrawActionMap(std::map<std::string, ActionFunctionType>& actions)`; Similar to `makeFilterActionMap()` above.

**NumberGrid functions**

- int `gridSetSize(CommandData& data)`; Installed in the option map.
- int `gridSetMaxNumber(CommandData& data)`; Installed in the option map.
- int `gridSetNumber(CommandData& data)`;
- int `gridCalculate(CommandData& data)`;
- int `gridApply(CommandData& data)`;
- int `gridApplyColor(CommandData& data)`;
Complex Fractal functions

- `int setPlaneSize(CommandData& data);` Installed in the option map.

Julia functions

- `int setJuliaFractal(CommandData& data);` Installed in the option map.
- `int setJuliaParameters(CommandData& data);` Installed in the option map.

Mandelbrot functions

- `int setMandelbrotFractal(CommandData& data);` Installed in the option map.

Color Table functions

- `int setColorTableSize(CommandData& data);` Installed in the option map.
- `int setColor(CommandData& data);`
- `int setRandomColor(CommandData& data);`
- `int setColorGradient(CommandData& data);`
- `int colorAction(CommandData& data);` Similar to `filterAction()` above.
- `void makeColorActionMap(std::map<std::string, ActionFunctionType>& actions);` Similar to `makeFilterActionMap()` above.

Meta functions

- `int helpFunction(CommandData& data);` Installed in the option map.
- `int errorFunction(CommandData& data);` Installed in the option map, under `?`. This function uses the global variable, `optopt`, to display an message, then calls `helpFunction()` and returns 1.
- `void makeOptionMap(std::map<int, ActionFunctionType>& actions);` Creates a map from command line option flags (such as `1`, `[w]`, `[d]`, etc.) to the action commands.
- `int imageCommand(int argc, char *argv[], std::istream& is, std::ostream& os);` Creates and initializes a `CommandData` object, including filling in all of the action maps, using the functions above. Also builds the master option map. Loops over the command line arguments, using `getopt()`. Sets `data.argument` to the global, `optarg`, unless `optarg` is `0`, in this case sets `data.argument = ""`. For each option, if the option is in the option map, call the action function from the option map. If the action function returns anything other than `0`, break from the loop, and return the value returned. If the option given is not in the option map, give a message about the unrecognized option, then call the `helpFunction()`, break from the loop and return `1`. If all options are processed correctly, and no action returns anything but `0`, return `0`.

Create `ppm_command.cpp`

- `int main(int argc, char *argv[])` Create this function, which calls `imageCommand()`, and returns whatever is returned by `imageCommand`.

Update `Makefile`

- This file must include the rules to build the program `ppm_menu` and `ppm_command`. See the modifications in the class example for 2 target programs.

Additional Documentation

- [C++ Reference]
- [Examples from class]
- [Sample Script]

Sample PPM Images

Here are some of the images created by the shell script in the sample session.

- [Bright face]
Show Off Your Work

To receive credit for this assignment, you must

- zip the source code (.cpp and .h files) and the Makefile and upload to the Canvas submission system
- 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)

- Create additional command line options for other features of the program that you add.