MP 4 Stacks and Queues

Extra credit: Friday, October 9 at 11:59 PM
Due: Friday, October 16 at 11:59 PM
Doxygen for MP 4

Goals and Overview

In this MP (machine problem) you will:

- learn about Stacks and Queues
- learn about depth-first-search (DFS) and breadth-first-search (BFS) traversal
- learn about some common implementation issues regarding code reuse

Checking Out the Code

To check out the provided code for MP4 from the class repository, run

```
svn up
```

TERMINAL

from your `cs225` directory.

This will update your working directory to contain a new directory called `mp4`. You can see a list of files you should be aware of here.

Doxygen

The Doxygen for MP4 is available here.

MP4.1: The `Stack` and `Queue` Classes

Template Compilation

Note that the `stack.h` and `queue.h` files we've given you includes the `stack.cpp` and `queue.cpp` files respectively at the end, so you should NOT use a `#include "stack.h"` (or `#include "queue.h"`) statement at the top of the `stack.cpp` (or `queue.cpp`) file. We have done it this way so that the template types are instantiated appropriately, as explained in MP3.

The `Stack` Class

You will write a class named `Stack` that works just like the stack you heard about in lecture, with the addition of the `peek()` function. It is declared in the given file `stack.h`. You will implement it in `stack.cpp`.

Your `Stack` class must implement all of the methods mentioned in the Doxygen for the Stack class. Please read the documentation (either in Doxygen or in the header file) to see what limitations we have placed on your `Stack` class and what the running times of each function should be.

The `Queue` Class

You will write a class named `Queue` that works just like the queue you heard about in lecture, with the addition of the `peek()` function. It is declared in the given file `queue.h`. You will implement it in `queue.cpp`.

Your `Queue` class must implement all of the methods mentioned in the Doxygen for the Queue class. Please read the documentation (either in the Doxygen or in the header file) to see what limitations we have placed on your `Queue` class and what the running times of each function should be.

Testing

We have provided a file `testStackQueue.cpp` which includes several (albeit trivial) test cases that your code should pass if it is correct. To compile this executable, type:

TERMINAL
```
make mp4.1
```

You can check your code's output with these tests with our solution output:

TERMINAL
```
./testStackQueue > testStackQueue.out
diff testStackQueue.out soln_testStackQueue.out
```

It's also a good idea to run your code under ASAN:

TERMINAL
```
./testStackQueue-asan
```

These test cases are deliberately insufficient. We encourage you to augment this file with additional test cases, using the provided ones as examples.

Extra Credit Submission

For extra credit, you can submit the code you have implemented and tested for MP4.1. You must submit your work before the extra credit deadline as listed at the top of this page. Follow the handin instructions given at the bottom of this page.

Grading Information (MP 4.1)

The following files are used to grade MP4.1:

- `stack.cpp`
- `queue.cpp`
- `partners.txt`

All other files (including any testing files you create) will not be used for grading.

MP4.2: Flood Fill

For this part of the assignment, you will be writing a number of functions that execute a “flood fill” on a `PNG` image. To flood fill a region of an image, you specify a point on the image and a fill color (or pattern) and all points similar in color and adjacent to the chosen point are changed to the fill color (or pattern). We will implement two different fill algorithms and two different fill patterns. The two fill algorithms are animated in the pictures above with the solid color fill pattern that you will implement.

Functors

Your first task in this part of the project is to implement the mechanism by which colors are selected for the fill pattern. To do this, you will create function objects (or “functors”) which will be passed into the fill algorithm and applied to pixels in a `PNG`. Specifically you will be making two new functors, each of which will be derived classes of the abstract base class called `colorPicker`, which is provided and described in the Doxygen for `colorPicker`. We have also given you an example functor so you can see how things are supposed to work. Our functor is called `gridColorPicker`. The `colorPicker` base class has one purely virtual function that you **must** implement in your derived classes, but you may add others if you wish. In particular, you should expect your derived class constructors to initialize the functor with situation-dependent variables.

The purpose of any `colorPicker` is to take an image coordinate as it’s `(x,y)` parameters to `operator()`, and return the color it would suggest the client recolor the source `PNG` to. So in the Pac-Man examples above, the `colorPicker` is returning blue, telling the flood fill to recolor each pixel of Pac-Man to a specific shade of blue.

Your task is to implement the `solidColorPicker` and `gradientColorPicker` function objects. Refer to the documentation for implementation details: they **must** override the abstract **operator()**!

The Fill Algorithms

You will be implementing two different basic kinds of flood fills: **depth-first-search** (DFS) and **breadth-first-search** (BFS), each with three different `patterns`. The first pattern will be the simple solid color flood fill you have seen in MSPAINT or Photoshop (`solidColorPicker`). The second will be filling in the area with a grid (`gridColorPicker`), and the third a gradient radiating out from the point at which the flood fill originated (`gradientColorPicker`). The animations above are breadth-first-search and depth-first-search, respectively, both with a solid fill.

You must implement **all** of the functions in the filler namespace, **including those in the nested dfs and bfs namespaces**. Refer to the documentation for implementation details.

Testing

ImageMagick

Note that you must have ImageMagick installed if you're developing on your personal machine. You can test for this requirement by running

```
TERMINAL
convert
```

in your terminal. Windows users should visit ImageMagick to download the latest binary. OSX users can use homebrew or MacPorts to install it if missing. Linux users likely already have it installed. If not, then you can install it via your system's package manager.

We have provided a file `testFills.cpp` which includes several test cases that your code should pass if it is correct. To compile this executable, type:

```
TERMINAL
make mp4.2
```

This will create two executables: `testFills` and `testFills-asan`, just like in MP3.

One important thing to note about this MP is that the `.gif` compression may take some time. In general, on remlnx via `ssh` (it might be slower over NX), the solution code runs in somewhere from 5-20 seconds when all the tests are enabled and completed and the server is under a light load. It is advisable to comment out the tests for things you have already completed, so that you don't have to wait as long (and use the CPU up) for things you already did. That said, the main point is that if you code doesn't finish immediately don't fret. Also note that as more traffic is happening on the server things will slow down. This may be an incentive to not wait until the last minute and have to try to run your code when everyone else is too!

To that extent, `testFills` takes a parameter specifying which tests you want to run. If you are debugging a specific image, we suggest you only run that image.

```
./testFills colorpickers   # tests your colorPicker classes
./testFills grid           # tests DFSfillGrid/BFSfillGrid
./testFills solid          # tests DFSfillSolid/BFSfillSolid
./testFills gradient       # tests DFSfillGradient/BFSfillGradient
./testFills pacman         # tests DFSfill/BFSfill with a rainbow colo
./testFills all            # runs all tests (SLOW!)
```

Alternatively you can run them with ASAN, like so:

```
./testFills-asan colorpickers
./testFills-asan grid
./testFills-asan solid
./testFills-asan gradient
./testFills-asan pacman
./testFills-asan all
```

The `testFills` program will generate images and animated gifs in the `images/` folder, which can be diffed with the solution images in `soln_images/`. For animated gifs, the frames making up the animation are placed in `frames/`, and can be diffed with their corresponding expected frames in `soln_frames/`.

You can check your color picker outputs by redirecting `testFills`' output to a file and `diff`ing that with our solution output:

```
./testFills all > testFills.out
diff testFills.out soln_testFills.out
```

You can diff each frame, which will probably be useful for debugging:

```
diff frames/bfsSolidTest00.png soln_frames/bfsSolidTest00.png
```

These images and the animations are also conveniently compiled into a single html page:

```
firefox testImages.html &
```

And all the provided tests can all be automatically diffed by running the following script:

```
sh ./diffAllFills.sh
```

Note that this script is much slower than running the individual tests, so it will not be much use early on, but it may help as a sanity check towards the end of your MP to make sure you've diffed all the provided images :-).

These test cases are deliberately insufficient. We encourage you to augment this file with additional test cases, using the provided ones as examples.

Handing in Your Code

To facilitate anonymous grading, **do not** include any personally-identifiable information (like your name, your UIN, or your NetID) in any of your source files. Instead, before you hand in this assignment, create a file called `partners.txt` that contains only the NetIDs of people in your collaboration group (if it exists), one per line. If you worked alone, include only your own NetID in this file. We will be automatically processing this information, so do not include anything else in the file. If we must manually correct your submission, you may lose points. As always, if you're working in a group, each group member must hand in the assignment. Failure to cite collaborators violates our academic integrity policy; we will be aggressively pursuing such violators.

To commit your changes to the repository type:

```
TERMINAL
svn ci -m "mp4 submission"
```

Grading Information

The following files are used to grade MP 4:

- `stack.cpp`
- `queue.cpp`
- `solidColorPicker.cpp` (only MP 4.2)
- `solidColorPicker.h` (only MP 4.2)
- `gradientColorPicker.cpp` (only MP 4.2)
- `gradientColorPicker.h` (only MP 4.2)
- `filler.h` (only MP 4.2)
- `filler.cpp` (only MP 4.2)
- `partners.txt`

All other files will not be used for grading.

Good luck!