

Queue Data Structure

Programming Project 1

CS 415

Overview

Your task in this project is to write a generic FIFO (first-in, first-out) queue with enqueue, dequeue and other operations. We will be relying on this queue implementation throughout the rest of the semester, so it's important that the implementation be efficient.

Specifically, enqueue and should work in $O(1)$ time, as should any other operations for which this efficiency is reasonable. Throughout the semester it will be your job to weight the tradeoffs of code complexity versus efficiency. Never discount the importance of simplicity, but also do not naïvely use a linear time algorithm for a problem which can be solved by a simple constant time algorithm. Also remember that very often the data that will be available as input to operations is up to you—part of designing data structures is deciding what data components you need to keep around. Again, simplicity dictates that we store as little as possible while still maintaining all truly useful information.

The file `queue.h` is included with the project materials and will get you started. Keep in mind that this is an external include file and so should not contain information specific to your implementation. You should fill in your queue implementation in `queue.c`. This pattern will be used throughout the projects. With rare exception you will not be modifying the header files provided to you. Each API function declaration in `queue.h` is preceded by a comment which describes what is expected of the function. If the de-

scribed behavior is unclear to you, please get clarification on the exact meaning, as imprecise specification is often the source of bugs. At any rate, ensure that your implementation fulfills the given contract for the function.

The project and coding guidelines covered in the first lab section as well as on the 415 course web site also apply.

How to Test Your Code

It's crucial that systems code be correct and robust. Testing is especially important since you are writing a library rather than an application. You must test your code with reasonable and unreasonable test cases, and ensure that your code always either causes the correct effect or cleanly returns the error condition that prevents the request from completing. All API functions should be tested.

To facilitate testing, a test program (`test_queue.c`) is provided for you. This is in no way complete, and since we know nothing about your design or implementation, these tests are written from a black box perspective. While black box testing is good for ensuring that the interface is completely implemented and correct for normal values, exhaustive testing of input values is impractical, so it is more efficient to approximate exhaustive input testing by focusing on inputs that exercise boundaries in your implementation.

As you engineer your design, outline the corner cases of both the abstraction and

Editing

It is permissible to code in another editor and only use Visual Studio to build your project, and you can actually even build your project using `make` on the command line while developing. However, the standard grading caveat applies—at grading time, the Visual Studio project will be used to compile your project, and the Visual Studio editor will be used for code walkthroughs (make sure the line endings in your source files work as they should). In addition, while Visual Studio does have its problems, it offers many features which make dealing with code complexity and debugging in larger projects much more manageable, and as such it will be very helpful for you in later projects.

Running Custom Tests

To change the test application which is compiled into the `minisystem.exe` executable, open the project Makefile. Search for a variable named `MAIN`. This variable should be the base name of the `.c` file which contains the main function will serve as the entry point for your test program. As is noted in the file, do not include the `.c` extension in the name assigned to `MAIN`. Then just save the changes and rebuild the project.

Submission

As you create new source files, including testing applications that you write, add them to the `minisystem` directory as well as to correct folder in the Visual Studio Solution Explorer pane.

When you are ready to submit your project, the Visual Studio Solution file (`.sln`), the

Visual Studio Project file (`.proj`), the Makefile, and all `*.c` and `*.h` files should be copied from this directory into a new directory named `project1`. This directory should be archived and compressed into `project1.zip`. Submit this file via CMS by the due date.

If you want to test that you have included all of the necessary files, extract your archive into a directory, open the Visual Studio Solution, and check that it builds correctly.