CMPS 12M

Introduction to Data Structures

Lab Assignment 6

Objective

The purpose of this assignment is to learn how to use a separate .h header file and .c implementation file to construct an abstract data type in C.

Overview

You are given an incomplete implementation of a queue in C. Your task is to implement the functions so the program works correctly. Files for this lab can be found in:

`/afs/cats.ucsc.edu/users/r/nwhitehe/cmps12/lab6/`

Separate Compilation

This lab will introduce you to header (.h) files, also known as #include files, which are C's equivalent of interfaces, and implementation files (.c) which are C's equivalent of implementation files. C has no keyword private, but its equivalent is to put hidden information only in implementation files.

For your program you will use separate compilation to implement a simple queue of lines in C.

cbox

Sample code has been provided in the cbox subdirectory. First, study the trivial cbox application which shows how to do separate compilation using header files and implementation files.

Inspect the following files :

cbox.h The interface to the cbox ADT showing the constructor, destructor, accessors, and mutators. In Java these would be the public functions.

cbox.c The implemention of the ADT, showing the private functions and fields. Note that the struct fields are thus hidden from the client.

main.c Shows how to create and access an ADT. The application itself is trivial.

Makefile Builds each object (.o) file and then links them into an executable image. Note how it creates dependencies by calling gcc with the -MM flag.

Try experimenting with "gcc -MM cbox.c" and "gcc -MM main.c". This should give you an idea of how the option works.

Queue

The queue subdirectory provides a simple queueing application, where each line of the input is read into a queue. The queue is then printed and the program stops. The idea of a stub is introduced. Incomplete code is just represented by a print statement which must eventually be replaced by actual code. The following command will tell you where all of the stub calls are:

```
grep STUB *.[hc]
```

You must implement queue_new, queue_insert, and queue_remove. Your implementation must be done via the several files, whose object modules are linked together into a single executable. Remove any complaints valgrind may have about memory access or memory leaks.

What to Turn In

All files you turn in for every assignment and lab should begin with a comment block that includes your name, CruzID, class, date, filename, short description of the file's role in the assignment, and any special instructions related to the file. Also create a file called README. The README file should have the normal comment block, then list all the files being submitted (including itself) along with any special notes to the graders. If you do pair programming, both names and CruzIDs should appear as author and in the README file.

For this lab, submit the following files:

```
README
MAKEFILE
Makefile
queue.c
queue.h
main.c
```

To submit, use the submit command.

```
submit cmps12b-nojw.f14 lab6 ...
```

Your makefile should have a default target that builds an executable called "queue", along with phony targets "clean" that removes compiled object files, "spotless" that cleans up all built files, and "test" which runs the executable in valgrind.