This is another warm-up, ungraded, exercise. The goal of this exercise is to familiarize you with C programming and its associate tools, which are essential for three of the subsequent (graded) labs.

Platform

This exercise requires the Linux platform. You may use the Linux machines in the OS Lab. If you want to use your own Linux machine, make sure that you have `valgrind` installed besides `make`, `gcc`, and `gdb`.

If you use the Linux machines in the OS lab, please use the given personal account (the one prefixed with `user`) to login. Your account has been assigned to you, please read the blog for details. Please remember to (i) change you password with `yppasswd` the first time you login, (ii) disable Firefox cache, and (iii) save and backup your files often.

There is NO need to ssh into SunFire to do this exercise.

If you are using MS Windows from home and would like to use SunFire for this exercise, it is fine too, but you will not be able to test your program with `valgrind`.

Students have reported that they are able to complete the lab on Mac OS X and `valgrind` works fine on Mac OS X.

Objective

In this exercise, you will program in C using pointers. You will get a chance to practice using `gdb`. You will also learn about using `make` to automate the compilation process, using `assert` to check for correctness of your program, and using `valgrind` to check for memory-related errors in your code.

You will implement one of the most basic data structure, a queue, using a linked list, with five functions, `queue_create`, `queue_delete`, `queue_enq`, `queue_deq`, and `queue_is_empty`. Queue is a common data structure used in operating systems. We will use this structure when you program a CPU scheduler later this semester. A quick review of linked list and queue from CS1102 or CS1020 (or the equivalent course from polytechnic) might be useful for some of you.

Skeleton Code

A skeleton code has been given to you. You can download the skeleton code with the command:

```
wget http://www.comp.nus.edu.sg/~ooiwt/cs2106/1112s1/lab02.tar.gz
```

To unzip the file, run[1].

```
tar -zxvf lab02.tar.gz
```

You should see a directory called `lab02`. Under the directory `lab02`, you will see four files:

- `Makefile`: a file that contains compilation instruction.
- `queue.h`: a header file that declares the structure of queue and functions that operate on queue.
- `queue.c`: a partial and incorrect implementation of the queue data structure.
- `queue_test.c`: a test program that illustrates how the queue data structure is used as well as tests the correctness of the implementation.

[1] `man gzip` and `man tar` to find out what these options mean if you are interested

Compiling and Running

To compile, type

```
make
```

in your shell. The given skeleton code should compile, producing the executable file `queue_test`. Type

```
./queue_test
```

to run the test program. You should received a segmentation fault error due to bugs in `queue_enq`.

Makefile

`Makefile` is a text file that contains compilation instructions with information about dependencies among the files. It is the default input to the `make` command. The nice thing about `make` is that, it checks for dependencies for you and only modified files are re-compiled. For instance, if you change `queue_test.c`, then `queue.c` is not re-compiled when you run `make`. Interested students can google and learn how to write your own Makefile yourself as we will not cover this in CS2106.

Test Program: queue_test.c

A good place to start for this exercise is to look at the file `queue_test.c`. This shows how the different APIs for queue will be used.

You may modify this file as much as you like to test your queue implementation. The given file provides a good place to start.

assert

There are many calls to the `assert` macro in `queue_test.c` `assert` aborts the execution if the given statement is false, and is extremely handy in checking for invariants in a program.

Queue implementation: queue.c and queue.h

The next files you should read are `queue.h`, to understand the structure of a queue, and `queue.c`, to understand the functions `queue_create` and `queue_is_empty`.

You must NOT modify the structure and function declarations in anyway (e.g., add new parameters, change return type).

Your Task

In `queue.c`, a buggy `queue_enq` has been given to you. This function will cause a segmentation fault when you run `queue_test`. You should identify the bug and fix it. The `gdb` debugger might be helpful to you here.

An empty `queue_deq` and `queue_delete` has been given to you. You should complete these two functions according to the specification given in the code. Besides ensuring correct implementation of the two functions, you should use `free` to deallocate any allocated memory properly.

Using `valgrind`

`valgrind` is a tool that is useful to track down memory errors (invalid memory access, memory leaks etc.). To complete this exercise, not only should your solution pass all test defined in `queue_test.c`, it should run in `valgrind` without any memory leaks or other memory errors. To run your executable in `valgrind`, type:

```
valgrind --tool=memcheck --leak-check=yes ./queue_test
```

The following site would be useful to learn more about `valgrind`:

`http://www.cprogramming.com/debugging/valgrind.html`

Tips

- Use `gdb` to help you debug. The commands you learnt from Lab 1 will be useful.
- Have a clear idea about what each variable is (Is it an int? A pointer to int?) and what you want your program to do (e.g., what values to put into which box? what is the address of the box? etc.) before you code . Walk through the steps in your mind or on paper before you code. Do not use trial and error. If you find yourself doing trial and error (Mmm.. x = y does not work, let me try *x = &y) then you should step away from the keyboard and spend sometime thinking and reading.

THE END