CS2011 (Intro To Computer Systems)
Lab0 - C Programming

This is an **individual** project. All hand-ins are electronic via Canvas.

1 Environment Setup

For this lab, you may use any linux or windows environment of your choosing to compile, run, and test your program. However, your submission will only be tested on an **Ubuntu 20.04** or **Linux Mint 20 OS** (**O**perating **S**ystem) and compiler configurations similar to those that come pre-installed or installed later via the default repository on those OSes. We advise you to implement and test your code on such machines/VMs and such versions of OS. *If you run into compilation errors or runtime errors when you compile and run the given code in the handout without any modification, most likely you have installed the wrong version of OS*. Although, you may be able to fix such errors and move on with your code implementation, remember that if you make any changes to files that are not to be submitted as part of your hand-in, then your code will not compile or run correctly on the test environment when grading your submission.

You may find the ISO images for those two OSes here:

Ubuntu 20.04: https://releases.ubuntu.com/20.04/

Linux Mint 20: https://linuxmint.com/edition.php?id=281

If you are new to installing Virtual Machines (VMs) on your **host OS** (the majority of you have a Windows or a Mac **physical** machine/laptop). Please do an Internet search and find out how to install a Virtualization software on your own machine or laptop. If you want a cross-platform virtualization software/ solution that works for both Mac and Windows (or even a Linux machine if you are using a Linux machine as your physical machine but do not use Ubuntu or Linux Mint), I recommend downloading and installing VirtualBox.
Note that if you own one of the new *M1 or M2 MacBooks*, VirtualBox may

NOT run on your Macbook; please consult with your instructor or TA if you own a MacBook M1/M2 and they will help you with alternatives.

A simple guide/tutorial on how to install VirtualBox and use it to run a Ubuntu 20.04 VM is posted on Canvas under this lab module, but you can also do a direct Internet search and learn how to setup virtualization using VirtualBox.

IMPORTANT: Regardless if you decide to use the tutorial posted on canvas or some online resource to install the VM, please make sure you provide **adequate disk space** (30GB) and **CPU cores** (2 cores) for your VM to function all the time, otherwise your VM may crash anytime during the semester while you are working on other labs and you may need to reinstall it.

If you are having trouble setting up your environment, please seek help from one of your TAs. TA tutoring hours will be posted on Canvas. Please do **NOT** wait until close to the lab deadline and start seeking help with environment setup.

2 Objective

This lab will give you practice in the style of programming you will need to be able to do proficiently, especially for the later assignments in the class. The material covered *should* all be review for you. Some of the skills tested are:

- Explicit memory management, as required in C.
- Creating and manipulating pointer-based data structures.
- Working with strings.
- Enhancing the performance of key operations by storing redundant information in data structures.
- Implementing robust code that operates correctly with invalid arguments, including NULL pointers.

The lab involves implementing a queue, supporting both last-in, first-out (LIFO) and first-in-first-out (FIFO) queueing disciplines. The underlying data structure is a singly-linked list, enhanced to make some of the operations more efficient.

3 Resources

Here are some sources of material you may find useful:

1. *C programming*. A recommended textbook is Kernighan and Ritchie, *The C Programming Language, second edition*. You may be able to find a copy on reserve in the Langsam library. For this assignment, Chapters 5 and 6 of that book are especially important. There are good online resources as well, including: https: //en.wikibooks.org/wiki/C_Programming

2. *Linked lists*. There are a number of resources online that explain the **concept** of a singly linked list if you are not familiar with it already. Please do NOT attempt to copy code from those resources into your program. You must implement everything from scratch on your own.

3. *Asymptotic (big-Oh) notation*. If you are unsure what "$O(n)$" means, there are also a number of resources online that explain the **concept** of a Big-O notation which will help you with some of the performance tasks in this assignment.

4. *Linux Man pages*. The authoritative documentation on a library function *FUN* can be retrieved via the command "man *FUN*." Some useful functions for this lab include:

   Memory management: Functions malloc and free.
   String operations: Functions strlen, strcpy, and strncpy. (Beware of the behavior of strncpy when truncating strings!)

If you have any questions about the assignment or need some clarifications, please be sure to reach out to one of your TAs for this course. TA availability will be posted on Canvas. Also, consistently check Canvas for any updates regarding this lab assignment.

As the Academic Integrity Policy states, please remember that you should not search the web or ask others for solutions. That means that search queries such as “linked-list implementation in C” are off limits.

4 Downloading the assignment

For all assignments in this class, you should download the handout from Canvas.

Your lab materials are contained in a tar archive file called lab0_handout.tar, which you can download from Canvas. Start by copying this file to a directory on your VM in which you plan to do your work “e.g., ~/Downloads”. Then open a terminal in your VM and enter the following commands:

```
linux>cd ~/Downloads

linux> tar xvf lab0_handout.tar
```

This will create a directory called lab0_handout that contains a number of files. Consult the file **README** for descriptions of the files. You will modify the files queue.h and queue.c.

5 Overview

The file queue.h contains declarations of the following structures:

```
/* Linked list element */
typedef struct list_ele {
   char *value;
   struct list_ele *next;
} list_ele_t;
/* Queue structure */
typedef struct {
   list_ele_t *head;  /* First element in the queue */
} queue_t;
```

These are combined to implement a queue of strings, as illustrated in Figure 1. The top-level representation of a queue is a structure of type queue_t. In the starter code, this structure contains only a single field head, but you will want to add other fields. The queue contents are represented as a singly-linked list, with each element represented by a structure of type list_ele_t, having fields value and next, storing a pointer to a string and a pointer to the next list element, respectively. The final list element has its next pointer set to NULL. You may add other fields to the structure list_ele, although you need not do so.

Recall that a string is represented in C as an array of values of type char. In most machines, data type char is represented as a single byte. To store a string of length l, the array has $l + 1$ elements, with the first l storing the codes (typically ASCII format) for the characters and the final one being set to 0. The value field of the list element is a pointer to the array of characters. The figure indicates the representation of the list [“cab”, “bead”, “cab”], with characters a–e represented in hexadecimal as ASCII codes 61–65. Observe how the two instances of the string “cab” are represented by separate arrays—each list element should have a separate copy of its string.

In our C code, a queue is a pointer of type queue_t *. We distinguish two special cases: a *NULL* queue is one for which the pointer has value NULL. An *empty* queue is one pointing to a valid structure, but the head field has value NULL. Your code will need to deal properly with both of these cases, as well as queues containing one or more elements.

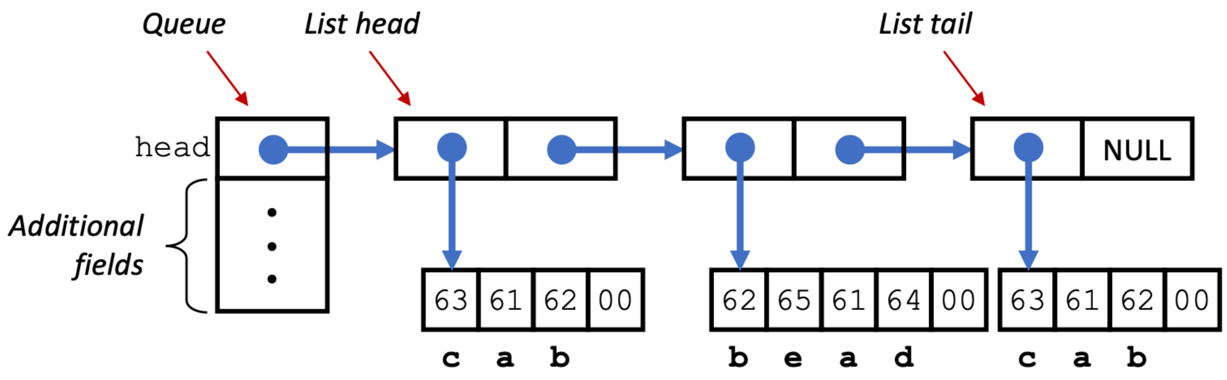

Figure 1: Linked-list implementation of a queue. Each list element has a value field, pointing to an array of characters (C’s representation of strings), and a next field pointing to the next list element. Characters are encoded according to the ASCII encoding (shown in hexadecimal.)

6 Programming Task

Your task is to modify the code in queue.h and queue.c to fully implement the following functions.

queue_new: Create a new, empty queue.

queue_free: Free all storage used by a queue.

queue_insert_head: Attempt to insert a new element at the head of the queue (LIFO discipline).

queue_insert_tail: Attempt to insert a new element at the tail of the queue (FIFO discipline).

queue_remove_head: Attempt to remove the element at the head of the queue.

queue_size: Compute the number of elements in the queue.

queue_reverse: Reorder the list so that the queue elements are reversed in order. This function should not allocate or free any list elements (either directly or via calls to other functions that allocate or free list elements.) Instead, it should rearrange the existing elements.

More details can be found in the comments in these two files, including how to handle invalid operations (e.g., removing from an empty or NULL queue), and what side effects and return values the functions should have.

For functions that provide strings as arguments, you must create and store a copy of the string by calling malloc to allocate space (remember to include space for the terminating character) and then copying from the source to the newly allocated space. When it comes time to free a list element, you must also free the space used by the string. ***You cannot assume any fixed upper bound on the length of a string—you must allocate space for each string based on its length***. Note: malloc, calloc, and realloc are the only supported functions in this lab for memory allocation, any other functions that allocate memory on the heap may cause you to lose points.

Two of the functions, *queue_insert_tail* and *queue_size*, will require some effort on your part to meet the required performance standards. Naive implementations would require $O(n)$ steps for a queue with n elements. ***We require that your implementations operate in time O(1), i.e., that the operation will require only a fixed number of steps, regardless of the queue size***. You can do this by including other fields in the queue_t data structure and managing these values properly as list elements are inserted, removed and reversed. Please work on finding a solution better than the $O(n^2)$ solution for all of the functions.

Your program will be tested on queues with over 1,000,000 elements. You will find that you cannot operate on such long lists using recursive functions, since that would require too much stack space. Instead, you need to use a loop to traverse the elements in a list.

7 Testing

You can compile your code using the command:

```
linux> make
```

If there are no errors, the compiler will generate an executable program qtest, providing a command interface with which you can create, modify, and examine queues. Documentation on the available commands can be found by starting this program and running the help command:

```
   linux> ./qtest
   cmd> help
```

The following file (traces/trace-eg.cmd) illustrates an example command sequence, which you can type into the qtest program:

```
# Demonstration of queue testing framework
# Initial queue is NULL.
show
# Create empty queue
new
# Fill it with some values.  First at the head
ih dolphin
ih bear
ih gerbil
# Now at the tail
it meerkat
it bear
# Reverse it
reverse
# See how long it is
size
# Delete queue.  Goes back to a NULL queue.
free
# Exit program
quit
5
```

You can also run qtest on an entire trace file all at once, as follows:

```
linux> ./qtest -f traces/trace-eg.cmd # This is the example trace.
linux> ./qtest -f traces/trace-01-ops.cmd  # This is the first real trace.
```

If you try to test the starter code, you will see that it does not implement many of the operations properly.

The traces directory contains 15 trace files, with names of the form trace-*k*-*cat*.txt, where *k* is the trace number, and *cat* specifies the category of properties being tested. Each trace consists of a sequence of commands, similar to those shown above. They test different aspects of the correctness, robustness, and performance of your program. You can use these, your own trace files, and direct interactions with qtest to test and debug your program.

You may also decide to debug your code at some point. I highly encourage you to use **GDB** (**G**nu **Deb**ugger) for this. GDB may seem difficult to use because it is a command line debugger, but it is in fact very powerful and does not require much learning on your side if you have ever debugged a program in the past. A good resource that will help you start using GDB to debug your program can be found here.

8 Evaluation

8.1 Autograder

Your program will be evaluated using the fifteen traces described above. You will be given credit (either 6 or 7 points, depending on the trace) for each one that executes correctly, summing to a maximum score of 100. This will be your score for the assignment—the grading is automated/semi-automated.

The driver program driver.py runs qtest on the traces and computes the score. This is the same program that will be used to grade your submission. You can invoke the driver directly with the following command (use Python v2.7):

```
linux> sudo apt upgrade
linux> sudo apt install python
linux> python driver.py
```

or with the command:

```
linux> make test
```

8.2 Style/Readability

Typically, the score you receive when running the auto-grader yourself will be your final grade for this assignment. However, your code must be formatted nicely and should include any additional code comments. Your grader may take those into consideration when grading your submission.

9 Handin

To receive a score, you should upload your submission to Canvas. You may handin as often as you like until the due date but please keep the number of submissions as MINIMAL as possible. The last submission is what will be graded.

Submit a tar file containing your code to Canvas. Running the make command will generate the tar file, lab0_handin.tar. You can upload this file to Canvas.

IMPORTANT: Do not assume your submission will succeed! You should ALWAYS check to see if your submission was successful on Canvas. IMPORTANT: Do not upload files in other archive formats, such as those with extensions .zip, .gzip, or .tgz.

10 Reflection

This should not be a difficult programming assignment for students who are fully prepared to take this course. If you found you had trouble writing this code or getting it to work properly, this may be an indication that you need to upgrade your programming skills over the next month (especially in the C language).

A good place to start is to carefully study Kernighan and Ritchie. This book documents the features of the language and also includes a number of examples illustrating good programming style. The book is a bit dated, and so it doesn't contain some more modern features of the language, such as the bool data type, but it is still considered one of the best books on how to program in C. OR, grab a good resource online.