Lab 4 – Pre-processor, header files

Version : 2014-2015

Exercise: *typedef, functions, multiple files, conditional compilation*

THE SOLUTION OF THIS EXERCISE NEEDS TO BE UPLOADED ON TOLEDO BEFORE THE NEXT LAB.

Your solution is only accepted if the following criteria are satisfied:

1. *Compilation with '-Wall -Werror' option generates no warnings or errors*
2. *Running 'cppcheck –-enable=all --suppress=missingIncludeSystem' on the source code results in no warnings or errors*
3. *files are named as : main.c, myqueue.c, myqueue.h*

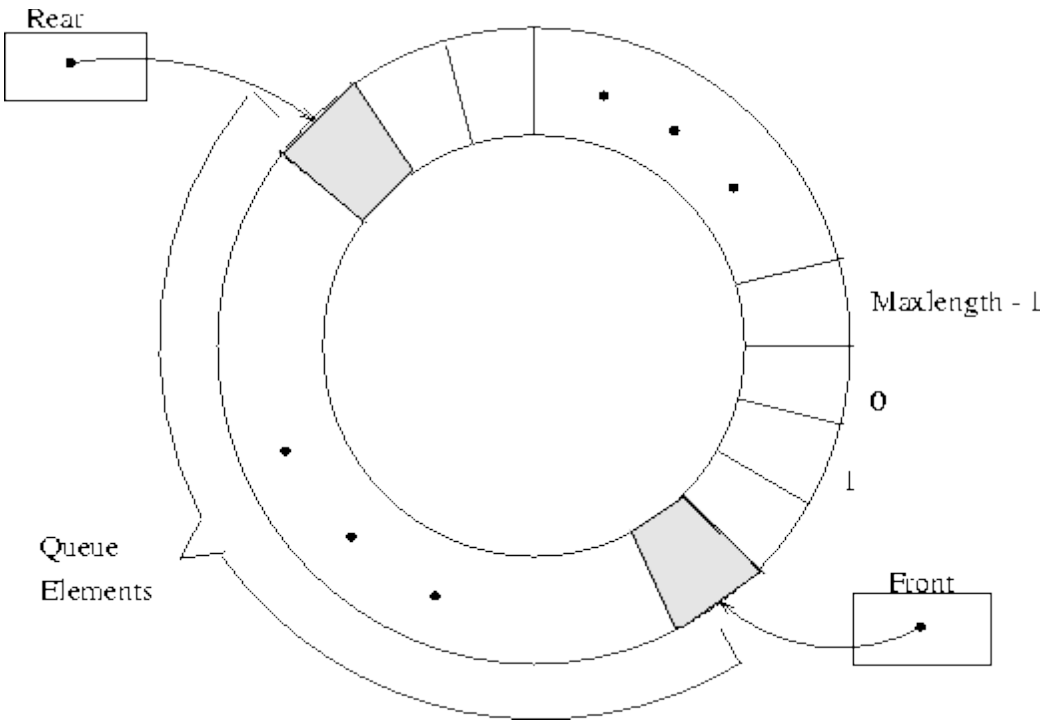
Write a small test application (e.g. as mentioned in the main_template.c file) to test your queue implementation.

DO NOT upload code that doesn't satisfy these criteria.

A queue is a data structure that implements the 'first come, first serve' principle. A queue is also called a FIFO (First in, first out). A queue is an important data structure in many Operating Systems tasks, e.g. in multi-tasking management, print queues, buffering queues (e.g. socket buffers, disk buffers,…), priority queues (see next lab), etc. Write a C program that implements and tests a queue. Use at least 3 files: main.c, queue.c, and queue.h. The queue.c file contains the queue variable to access the queue. Use a **circular array** as underlying data structure to store the queue elements. The array should be defined with **dynamic memory.** At least the following operators should be supported:

- queue_create: creates (e.g. allocatest memory) and initializes the queue and prepares it for usage
- queue_free: deletes the queue from memory such that there are no memory leaks; the queue can no longer be used unless queue_create is called again
- queue_size: return the number of elements in the queue
- queque_top: return the top element in the queue
- queue_enqueue: add an element to the queue
- queue_dequeue: remove the top element from the queue

- queue_print: prints all elements in the queue to stdout starting from the front element



Rear is the position in the queue where new elements are added (enqueue). Front is the position in the queue where elements are removed (top, dequeue).

The type of elements stored in the queue should not matter (int, float, structs, …). Functions to print, copy and free an element should be implemented by the caller of the queue (e.g. in main.c) such that the queue indeed doesn't need to worry about the type of the elements. Later you will learn how this can be implemented with function pointers.

On Toledo, you find already template files for main.c, queue.c and queue.h to start from.

Use **storage class** specifiers (extern, static, etc.) where needed. For example, to protect access to the queue variable and local functions in queue.c from other files like main.c.

Use **conditional compilation** to allow a user to run the code in ‘debug mode’.

Use **Valgrind** to check for potential memory leaks.

Exercise: *GDB*
Compile your code with debug information (How? Check man-pages of gcc). Run gdb to inspect variables, set breakpoints and use the step-by-step execution mode.

Exercise: *preprocessor, conditional compilation, gcc toolchain*
Run the pre-processor (How? Use the man-pages of gcc!) on the code of the previous exercise and save the result in a text file. Open this text file and find out what the pre-processor has done with (i) #include, (ii) #define, and (iii) #ifdef statements.

Change the queue implementation of the previous exercise such that at compile time the user can set 'debug mode' on/off and the maximum size of the queue. If the user doesn’t define a size at compile time, a default size should be taken.