BITS Pilani, Pilani Campus
2nd Sem. 2018-19
CS F211 Data Structures & Algorithms
==========
==========

Topic: Abstract Data Type, Linked Lists, Measuring Run Time

Definition: **ADT Queue** (a.k.a. FIFO List a.k.a. FIFO Queue)

- Queue newQ() // returns an empty Queue
- Boolean isEmptyQ(Queue q) // tests whether q is empty
- Queue delQ(Queue q) // deletes the element from the front of the Queue; returns the modified Queue
- Element front(Queue q) // returns the element from the front of the Queue;
- Queue addQ(Queue q , Element e) // adds e to the rear of the Queue; returns the modified Queue
- Queue lengthQ(Queue q) // returns the length of the Queue

Exercise 1. (a) Define the ADT Queue interface in file que.h.
(b) Implement ADT Queue using a linked list in file que.c. Note that efficient implementation of a Queue in a linked list requires a head (pointing to the first node of the linked list) and a tail (pointing to the last node of the linked list).
==========

Consider a task scheduling system where tasks are queued according to priority i.e. each task is added to a FIFO queue according to its priority. (e.g. an Operating System where processes arrive to be executed and each process has a priority that is not necessarily unique; e.g. a machine shop where requests for machining jobs come with a priority).
All these scenarios can be modelled using the **ADT MultiQ defined** with the following behaviour:

- MulitQ createMQ(int num) // creates a list of num Queues, each of which is empty.
- MultiQ addMQ(MultiQ mq, Task t) // adds t to the Queue corresponding to priority p in mq. Task includes a TaskID tid and a priority p.
- Task nextMQ(MultiQ mq) // returns the front of the non-empty Queue in mq with the highest priority.
- Task delNextMQ(MultiQ mq) // deletes the front of the non-empty Queue in mq with the highest priority; returns the modified MultiQ
- Task isEmptyMQ(MultiQ mq) // tests whether all the Queues in mq are empty
- int sizeMQ(MultiQ mq) // returns the total number of items in the MultiQ
- int sizeMQbyPriority(MultiQ mq, Priority p) returns the number of items in mq with the priority p.
- Queue getQueueFromMQ(MultiQ, Priority p) returns the Queue with priority p.

Exercise 2: (a) Define the MultiQ interface in file multiq.h
(b) Implement the operations on MultiQ in file muliq.c. Note that createMQ should dynamically allocate an array of size num (the value that is passed to it) and return it. The implementation should use ADT Queue to perform operations on Queue.

Exercise 3: (a) Write a main function in testMultiq.c that tests the MultiQ ADT using the following functions:

i. MultiQ loadData(File f) - reads from f a list of pairs (task_id, priority) and adds each item to a MultiQ appropriately.
ii. MultiQ testDel(int num) – performs num of delNextMQ operations.

You can use the attached files – input10.txt, input100.txt and input10000.txt each having 10, 100 & 10,000 <task_id,priority> pairs in them respectively.
[**Note**- there are a total of **10** different priorities each task can take in the given input files. Both task_id and priority are positive integer values.]

(b) Measure the time taken for adding the elements to MultiQ within the loadData operation. Also measure the time taken for performing the testDel function and compute the average cost per delete operation.
Measurements are to be done in two ways: (i) using functions in the library (see header file time.h and use man to find details) (ii) using the gnu profiler (*gprof*) that comes with gcc.

Given below is a sample program that measures the time taken by a program using - "sys/time.h"

```
#include <sys/time.h>                // for gettimeofday()

int main()
{
    struct timeval t1, t2;
    double elapsedTime;

    // start timer
    gettimeofday(&t1, NULL);

    // do something or call a function
    // ...

    // stop timer
    gettimeofday(&t2, NULL);

    // compute and print the elapsed time in millisec
    elapsedTime = (t2.tv_sec - t1.tv_sec) * 1000.0;      // sec to ms
    elapsedTime += (t2.tv_usec - t1.tv_usec) / 1000.0;   // us to ms
    printf("Total time is" + elapsedTime + "ms.\n";

    return 0;
}
```

To measure time using gprof, you don't need to add any instrumentation in the file like we did before. Rather we should compile our files and link them using the flag "`-pg`", execute your program and then generate the profiling output. For example you need to do the following for profiling a program writing in "file1.c" :

```
gcc –pg file1.c –o test        // compiles file1.c and generates an executable with
                                  name test

./test                         // execute the program

gprof test gmon.out > prof_output      // creates a file prof_output which contains
                                          the time taken for execution of each
                                          function in file1.c
```

(c) Repeat steps (a) and (b) for different sizes of the list in the input file and for different num values for deletion.