Cmp Sci 187: Programming with Data Structures
Queues

1. Queues and Stacks

How can you implement a Queue using Stacks?

And how can you implement a Stack using Queues?

2. Web Browser Cache

Many web browsers like Netscape, Internet Explorer, Firefox and Safari maintain a cache of web pages that you have visited. If you were implementing the browser, how would you maintain this cache?

Browsers also allow the user to restrict the size of the cache. How does this change your implementation. Hint: Does Queue seem to be an appropriate mechanism ?

```
public class Browser {
  public void visit (String url) {}
}
public class Page {
  public Page (String url) {}
}
public class Cache {
  public Cache () {}
  public boolean isCached (String url) {}
  public Page updateCache (Page p) {}
}
```

Cmp Sci 187: Programming with Data Structures
Queues

3. Web Server Request Buffer

A web server responds to page requests from clients that arrive independently of each other. The pages requested by clients can take different times to service. What are the possible ways in which you can organize the incoming requests ?

```
public class Server {
  public  void receive (Request r) {}
  private void reply () {}
}
```

4. DeQue: Doubly Ended Queue

A **`deque`** is a doubly ended queue, meaning that data can be queued and removed both at the end as well as at the beginning. A printer queue implemented using this data structure can then allow high priority jobs at the beginning of the queue rather than at the end. How can you implement a deque using a queue and using an auxiliary queue.

5. Virtual Memory Manager

Many operating systems like Linux using a virtual memory manager to give an illusion of a large memory when the actual physical memory is relatively small. On a 32 bit Intel processor architecture, an operating system can access 4GB (2^{32}) of memory but a notebook may only have 512 Mb of physical memory. This illusion of virtual memory is created by dividing the addressable memory into chunks of memory regions called pages (usually 4kb). Some of the pages are mapped to physical memory, up to a maximum of what can fit into the available physical memory size. The rest of the pages are kept on disk and swapped in based on usage. One strategy used to decide which pages are sent to disk depends on the usage of the pages, the page that was least recently used (LRU) is evicted to disk. What kind of data structure can be used to implement this strategy?