|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Part** | **1** | **2** | **3** | **4** | **Total** |
| *maximum* | **25** points | **25** points | **25** points | **25** points | **100**G101010 pointsG |
| ***Your Score*** |  |  |  |  |  |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Queues**

Reading Assignment: Thoroughly read Chapter 8 in the course textbook.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 1 Glossary Terms**

Define, in detail, each of these glossary terms from the realm of computer programming logic and design and computer topics, in general. If applicable, use examples to support your definitions. Consult your notes or course textbook(s) as references or by visiting Web sites such as: [**http://www.ask.com**](http://www.ask.com),[**http://www.bing.com**](http://www.bing.com), [**http://www.webopedia.com**](http://www.webopedia.com)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(a) Dequeue**

|  |
| --- |
| To “pop” an item from the front of a queue is to “dequeue” it. To “enqueue” is to “add” an item to the rear of a queue (or, in a priority queue, in the correct position in the queue). |

**(b) FIFO order**

|  |
| --- |
| “First In, First Out” refers to the defining protocol of a queue, namely, that items that are inserted first into a queue (or, items that are inserted first and with the highest priority in a queue) will always be in the front of a queue. Items inserted last (and/or with the lowest priority) will always be in the rear of a queue, etc. |

**(c) Linked Queue**

|  |
| --- |
| A linked queue is a linked based implementation of queue wherein the front of the queue is represented by a “front” node, whose “next” pointer references the next item (node) in a queue, and so forth until the final node, which has a “rear” pointer.  When accessing the linked queue via a “peek” operation, the data in the front node is always returned, and when mutating the linked queue via a “pop” operation (or, a ‘dequeue`), the front node data is returned and the reference (self.front) is set to it’s “next” pointer (self.front = self.front.next), which removes the front item from the queue.  “Adding” a new node to a linked queue involves creating a new node and setting the `self.rear.next` pointer to the new node, essentially adding the new item to the end of the queue. |

**(d) Priority Queues**

|  |
| --- |
| A priority queue follows all of the protocols of a regular queue, with the introduction of a “comparable” class of items to be added to the queue. These “comparable” items have a comparable “priority” property (defined by the \_\_lt\_\_ and \_\_le\_\_ special methods), and their position in a queue is determined by their level of priority.  Items of equal priority are added according to the FIFO principle, while items of unequal priority are inserted after items of greater priority, etc. |

**(e) Round - Robin CPU Scheduling**

|  |
| --- |
| Round Robin CPU scheduling is a process that makes use of a priority queue to allocate slices of CPU to certain process based on their priority and order of arrival. Some processes take higher priority than others (e.g., processes that take very little time to compute or are expected by the user to execute quickly, e.g., keyboard and mouse inputs) while long-lasting, background processes get smaller, more infrequent slices of CPU. |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 2 True / False Exercises**

For each of these exercises, enter True or False in the spaces provided.

**TRUE** **(1)** Queues are linear collections.

**??? (2)** The two fundamental operations supported by queues are pop and

**TYPO, missing second operation (I assume it is “add”)**

**FALSE** **(3)** The peek operation on a queue returns the item at the back of the queue without removing it.  
 **Returns item from front**

**TRUE** **(4)** The array implementation of a queue must access items at the logical beginning and the logical end.

**FALSE** **(5)** When using a circular array implementation for a queue, during the add operation, the front pointer moves ahead of the rear pointer.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 3 Multiple Choice Exercises**

Select the correct response or responses.

**(1)** What type of collection is a queue?

**(a) linear**  (b) parallel

(c) random (d) geometric

**(2)** Which protocol is supported by queues?

(a) last - in first - out (b) last - in last - out

**(c) first - in first - out** (d) first - in last - out

**(3)** Which of the following is true about a standard queue?

(a) the item that has been waiting the shortest is popped next

(b) the items are popped in LIFO order

**(c) the item that has been waiting the longest is popped next**

(d) removals are done from the rear of the queue

**(4)** Which of the following is NOT true of a priority queue?

(a) higher - priority items are popped before those of lower priority

**(b) items of equal priority are popped in LIFO order**

(c) items of equal priority are popped according to which item arrived first

(d) higher - priority items can jump ahead of lower - priority items

**(5)** How would you use a Python list method to remove and return an item at the front of the queue?

(a) peek(len(queue)-1) (b) peek(1)

(c) pop(len(queue)-1) **(d) pop(0)**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 4 Programming Exercises**

**(1)** What is the returned value and the state of the queue after the operation is executed?

Current queue state: **x a z b**

Operation: **q.peek()**

**RETURNS: x**

**STATE: x a z b**

**(2)** In the following code for the add method for a linked queue implementation, what is the missing code?

**def add(self, newItem) :**

**newNode = Node(newItem, None)**

**if self.isEmpty() :**

**self.front = newNode**

**else :**

**self.rear.next = newNode**

**# <missing code>**

**self.size += 1**

**COPY / PASTE from ADD method from my implementation of a LinkedQueue class, following Lambert’s example (the snippet above is missing the “self.rear = newNode” line):**

def add(*self*, *newItem*):

newNode = Node(*newItem*, None)

*if* *self*.isEmpty():

*self*.front = newNode

*else*:

*self*.rear.next = newNode

*self*.rear = newNode

*self*.size += 1

**(3)** A stack s , a queue q , and a max value priority queue p each have a single value of 3 in them. Then, the operations s.push(4) , q.push(4) and p.push(4) are executed. What would be the triple ( s.pop() , q.pop() , p.pop() ) ?

**s.pop() would return: 4**

**q.pop() would return: 3**

**p.pop() would return: 4 (assuming that “max value priority queue” assigns priority in descending, rather than ascending, order)**

**(4)** Consider the following operations performed on a **stack** of size 5 .

Push(1)

Pop()

Push(2)

Push(3)

Pop()

Push(4)

Pop()

Pop()

Push(5)

After the completion of all operations, the number of elements present on stack is **1**.

**(5)** Consider the sample program below which uses a list to demonstrate a queue implementation. Using this sample program, design a simple simulation of a Supermarket Checkout System with the following actions.

Set the size of the list to 6 .

Enqueue customer Jake

Enqueue customer Jill

Dequeue a customer

Enqueue customer Iris

Dequeue a customer

Enqueue customer David

Verify the size of the queue

Determine the customer at the front of the queue

|  |
| --- |
| **# program to demonstrate a queue implementation using a list**    **# initialize the queue**  **queue = []**    **# enqueue elements to the queue**  **queue.append("a")**  **queue.append("b")**  **queue.append("c")**    **print ("The initial queue")**  **print (queue)**    **# remove elements from the queue**  **print ("\nElements dequeued from the queue")**  **print (queue.pop(0))**  **print (queue.pop(0))**  **print (queue.pop(0))**    **print ("\nThe Queue after elements have been dequeued")**  **print (queue)**    **# print (queue.pop(0))**  **# oops - this will raise an IndexError**  **# since the queue is now empty** |

*# initialize the queue*

queue = []

*# Enqueue customer Jake*

queue.append("Jake")

*# Enqueue customer Jill*

queue.append("Jill")

*# Dequeue a customer*

queue.pop(0)

*# Enqueue customer Iris*

queue.append("Iris")

*# Dequeue a customer*

queue.pop(0)

*# Enqueue customer David*

queue.append("David")

*# Verify the size of the queue, returns True*

print(len(queue) == 2)

*# Determine the customer at the front of the queue, returns True*

print(queue[0] == "Iris")