STUDENT OUTLINE

Lesson 37 – Queues

INTRODUCTION: Queues are another restricted-access data structure, much like stacks. A queue is like a one-way line where data enters the end of the line and exits from the front of the line. Implementation of the `Queue` interface will support operations similar to, but different from, stacks.

The key topics for this lesson are:

A. Queues
B. Operations on Queues
C. Implementing Queues as a Linked List

VOCABULARY: QUEUE ENQUEUE
DEQUEUE

DISCUSSION:

A. Queues

1. A queue is a linear data structure that simulates waiting in line. A queue has two ends, a front and a (rear) end.

head tail

2. Data must always enter the queue at the end and leave from the front of the line. This type of action can be summarized as FIFO (first-in, first-out).

3. A queue is the appropriate data structure when simulating waiting in line. A printer that is part of a multi-user network usually processes print commands on a FIFO basis. A queue would be used to maintain the order of the print jobs.

B. Operations on Queues

1. The following operations are supported by the `Queue` interface:

```
public interface Queue
{
  boolean isEmpty();
  void enqueue(Object obj);
  Object dequeue();
  Object peekFront();
}
```

REMOVE FROM HEAD OF QUEUE

ADD TO TAIL OF QUEUE

2. Using a queue implemented through the `Queue` interface is very similar to using a stack. Here is a sample program that uses some of the key operations provided by a `Queue` implementation.

Program 37-1

```
public static void main(String[] args)
{
  ListQueue queue;

  for (int k = 1; k <= 5; k++)
    queue.enqueue(new Integer(k));

  while (!queue.isEmpty())
  {
    System.out.println(queue.dequeue());
  }
}
```

Run output:

```
1 2 3 4 5
```

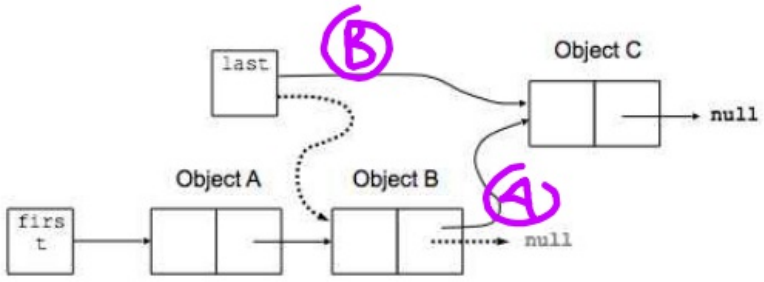
See Handout H.A.37.1, *Queue.*

3. See Handout H.A.37.1, *Queue Interface* for the full specifications of the `Queue` interface.

C. Implementing Queues as a Linked List

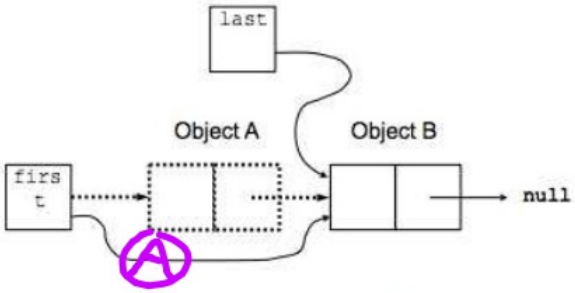
1. A queue can be implemented as an array or a linked list. If we use a linked list to implement a queue we must deal with the following issues.

2. Two external pointers must be used to keep track of the front and end of the queue. When discussing a queue it is traditional to add new data at the end of the queue, as in "get in at the end of the line." The term *enqueue* is used to describe this operation.



An *Enqueue* Operation – Inserting at the End

3. Data will be removed from the "front" of the queue. This operation is called *dequeue.*



A *Dequeue* Operation – Deletion from the Front

4. A queue can be implemented as a singly linked list if the two external pointers are appropriately placed.

5. When a new piece of data is added to the tail of the queue (enqueue), the external pointer `last` must be changed to point to the new node added to the list.

6. When an old piece of data is extracted from the queue (dequeue), the external pointer `first` must be changed to point to the appropriate node after the data has been removed.

SUMMARY/ REVIEW: Queues serve a very useful function whenever a program needs to simulate waiting in line. One of the lab exercises will require queues to solve an intriguing problem regarding binary trees.

ASSIGNMENT: Lab Exercise L.A.37.2, *RPN*