Queue :-

Queues are data structures that, like the stack, have restrictions on where you can add and remove elements. To understand a queue, think of a cafeteria line: the person at the front is served first, and people are added to the line at the back. Thus, the first person in line is served first, and the last person is served last. This can be abbreviated to First In, First Out (FIFO).

The cafeteria line is one type of queue. Queues are often used in programming networks, operating systems, and other situations in which many different processes must share resources such as CPU time.

One bit of terminology: the addition of an element to a queue is known as an enqueue, and removing an element from the queue is known as adequeue.

Although the concept may be simple, programming a queue is not as simple as programming a stack. Let's go back to the example of the cafeteria line. Let's say one person leaves the line. Then what? Everyone in line must step forward, right? Now, imagine if only one person could move at a time. So, the second person steps forward to fill the space left by the first person, and then the third person steps forwards to fill the space left by the second person, and so on. Now imagine that no one can leave or be added to the line until everyone has stepped forward. You can see the line will move very slowly. It is not difficult to program a queue that works, but it is *quite* a proposition to make a queue that is *fast*!!

There are a couple of basic ways to implement a queue. The first is to just make an array and shift all the elements to accommodate enqueues and dequeues. This, as we said above, is slow.

The slowness of the previous method is that with many elements, the shifting takes time. Thus, another method, instead of shifting

the elements, shifts the enqueue and dequeue points. Imagine that cafeteria line again. If the front of the line continually moves backwards as each person leaves the line, then the people don't need to step forward or backwards, which saves time.

Unfortunately, this method is much more complicated than the basic method. Instead of keeping track of just the enqueue point (the "end"), we also need to keep track of the dequeue point (the "front"). This all gets even more complicated when we realize that after a bunch of enqueues and dequeues, the line will need to wrap around the end of the array. Think of the cafeteria line. As people enter and leave the line, the line moves farther and farther backwards, and eventually it will circle the entire cafeteria and end up at its original location.

The best way to understand all of this is to read some source code. The example source code uses the second method, and is commented to help you understand. Try working through the code on paper.