Lecture 10.2

Queue structure

Queues

- A queue is much like a check-out line queue where the first element comes out first and the last element comes out last.
- Nodes are removed only from the front of the queue.
- Nodes are inserted only at the tail of the queue.

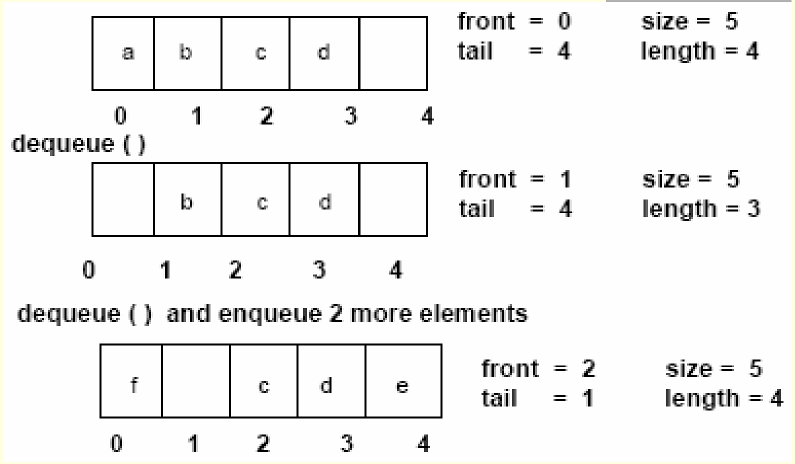
Sample Queue Applications

- Processor queue:
  - Entries of the user are placed on a queue and executed in the order they were entered.
- Keyboard queue:
  - Characters are placed on a queue and displayed in the order they were entered.
- Printer queue
- Router queue:
  - Sends packets in the order they arrive

Dequeue facts

- When using an array to implement the queue, every time we remove an element, there will be a space left free in the array.
- We will want to use those spaces when we reach the end of the array.

Queue implementation



The enqueue() function

```
bool enqueue (int val)
{
   // Check queue is not full, store then increment tail
   if ( length != size )
   {
        qPtr [ tail ] = val;
        tail = ( tail+1 ) % size;
        length++;
        return true;
   }
   return false;
}
```

The dequeue( ) function

```
bool dequeue ( int & data )
{
   //Retrieve then increment front
   if (length != 0)
   {
        data= qPtr [ front ];
        front=( front+1 )%size;
        length--;
        return true;
   }
   return false;
}
```

The main ( )

```
#include <iostream.h>
#include <stdlib.h>

bool enqueue (int val);
bool dequeue (int &data);
//declare these as global variables so they are visible to the
// functions
int* qPtr;
int front = 0, tail = 0;
int size = 0, length = 0;
```

The main () (cont)

```
int main()
{
   int size = 100;
   qPtr = new int [size];
   int num;
   int val;
```

The main() (cont 2)

```
for (int i = 0; i < size ; i++)
{
   cout << "Enter num: ";
   cin >>num;
   if ( enqueue ( num ) )
    …
```

The main ( ) (cont 3)

```
   while( dequeue(val)) {
   cout << "dequeued :" << val << endl;
   . . .
   system (“pause);
   return 0;
}
```