EECE Department
ELC 303-B

Queues

Definition

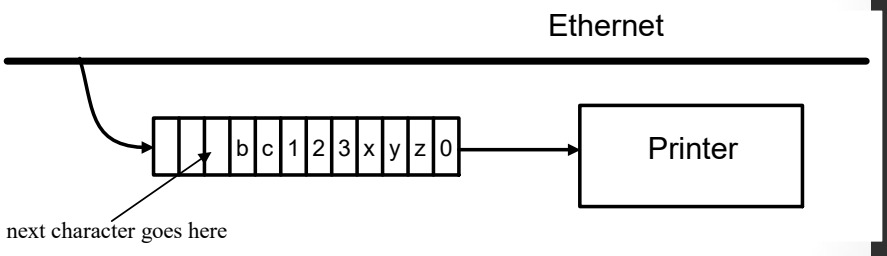
- A container data type that stores items in First-In-First-Out (FIFO) order.
-

Applications

- Computing
  - Buffering buffering buffering ……
    - Who can live without a buffer?
    - Speed matching
    - Communication links
  - Jobs waiting to be executed in a time-sharing multi-tasking OS
  - Jobs waiting to be printed
- Real world examples
  - People waiting in line in the bank
  - Cars in line at a toll booth

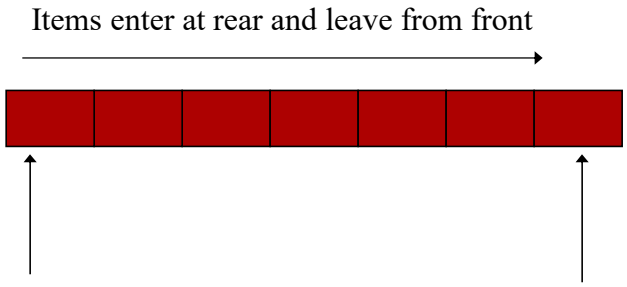
3

Printer Input Buffer Example



4

Dynamics



Rear item is most recent addition to queue

Front item has waited Longer than all other queue entries

5

Queues and lists

- A queue is a restricted form of a list.
- Additions to the queue must occur at the rear.
- Deletions from the queue must occur at the front

6

Queue ADT

Operations

bool Q.enqueue(queueElementType x)

Precondition: None

Postcondition: $Q_{post} = Q_{pre}$ with x added to the rear.

Returns: true if enqueue succeeds, false otherwise

queueElementType A.dequeue()

Precondition: !isEmpty()

Postcondition: $Q_{post} = Q_{pre}$ with front removed

Returns: The least-recently enqueued item (the front).

7

Queue ADT operations (continued)

queueElementType Q.front()

Precondition: !isEmpty()

Postcondition: $Q_{post} = Q_{pre}$

Returns: The least-recently enqueued item (the front).

bool Q.isEmpty()

Precondition: None

Postcondition: None

Returns: true if and only if Q is empty, i.e., contains no data items.

8

Code Example

```
int main()
{
  char c;
  Queue < char > q;
```

9

Code Example (continued)

```
// read characters until '.' found, adding each to Q and S.
  while (1) {
    cin  >>  c;
    if (c == '.') break;     // when '.' entered, leave the loop
    q.enqueue(c);
  }
  while (!q.isEmpty()) {
    cout << "Q: " << q.dequeue() << '\n';
  }
  return 0;
}
```

10

Dynamic Queues

- The advantages of linked list implementation are
  - The size is limited only by the pool of available nodes (the heap)
  - There is no need to wrap around anything compared with array implementation.

11

Header for Queue as Dynamic List

```
template < class queueElementType >
class Queue {
public:
  Queue();
  bool enqueue(const queueElementType &e);
  queueElementType dequeue();
  queueElementType front();
  bool isEmpty();
```

12

Private section

```
private:
  struct Node;
  typedef Node * Link;
  struct Node {
    queueElementType data;
    Link next;
  };
  Link qfront;
  Link qrear;
};
```

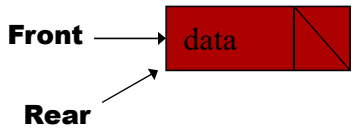
13

Implementation file, constructor

```
template < class queueElementType >
Queue < queueElementType >::Queue()
{
  // set both front and rear to null pointers
  qfront = NULL;
  qrear = NULL;
}
```

14

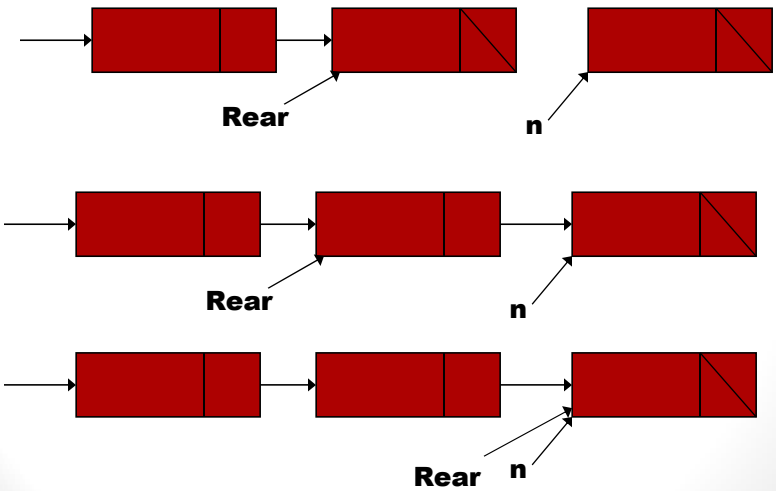
A Single-Element Queue



A single element queue
Front == rear

15

enqueueing



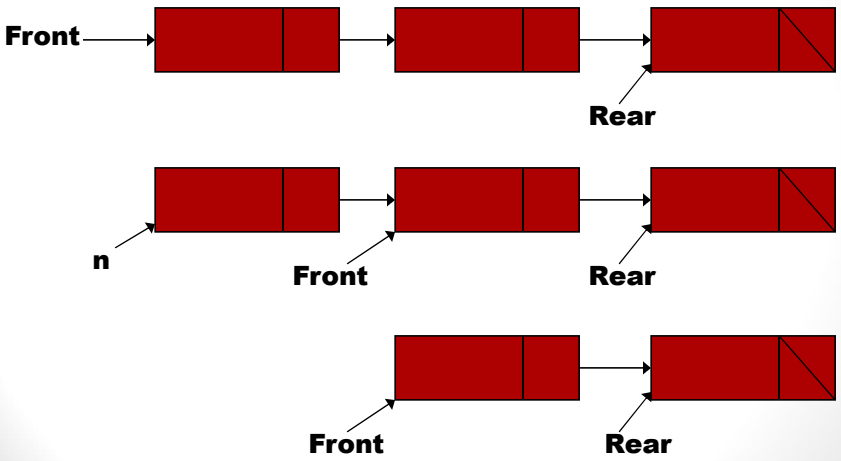
16

enqueue( )

```
template < class queueElementType >
bool Queue < queueElementType > ::enqueue(const queueElementType &e)
{// create a new node, insert it at the rear of the queue
  Link addedNode = new Node;
  if (addedNode == NULL) return false;
  addedNode->next = NULL;
  addedNode->data = e;
  if (!isEmpty()) { // existing queue is not empty
    qrear->next = addedNode; // add new element to end of list
} else {// adding first item in the queue
    qfront = addedNode; // so front, rear must be same node
}
qrear = addedNode;
return true;
}
```

17

dequeueing



18

dequeue( )

```
template < class queueElementType >
queueElemType
Queue < queueElementType >::dequeue()
{ assert(!isImpty());// make sure queue is not empty
  Link n = qfront;
  queueElementType frontElement = n->data);
  qfront = qfront->next;
  delete n;
if (qfront == NULL) // we're deleting last node
    qrear = NULL;
  return frontElement;
}
```

19

front ( )

```
template < class queueElementType >
queueElementType
Queue < queueElementType >::front()
{
  assert(qfront !=NULL);
  return qfront->data;
}
```

20

isEmpty( )

```
template < class queueElementType >
bool
Queue < queueElementType >::isEmpty()
{
  // true if the queue is empty -- when f is a null pointer
  return (qfront== NULL);
}
```

21

Priority Queues

- Priority queues are a special type of queues in which queue elements are processed in order of importance/priority
- The priority queues appears in different contexts
  - packets with different priority
  - patients at emergency section
- Implementation Approaches
  - Unsorted list
    - Adv: simple insert
    - Disadv: search before dequeue
  - Linked sorted list
    - Adv: simple dequeue (always get the first element)
    - Disadv: O(N) enqueue as we need to decide where to insert the received object

22