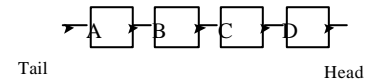


# Queue

## Queue

- New nodes can be added only to the rear
- Existing nodes can be removed only at the front
- First-in, first-out (FIFO)
- End of queue indicated by a link member to NULL
- Constrained version of a linked list

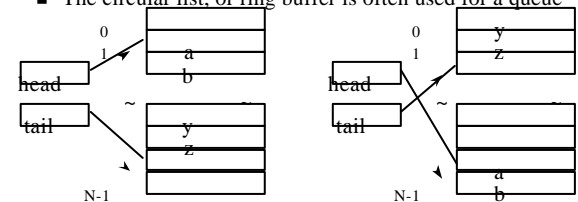


## Enqueue & Dequeue

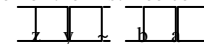
- **enqueue**
  - ◆ Adds a new node to the rear of the queue
- **dequeue**
  - ◆ Removes a node from the front of the queue
  - ◆ Stores and returns the dequeued value

## Queues Using Arrays

- The circular list, or ring buffer is often used for a queue



tail is the index of the first free cell



## Queues Using Arrays

- Insert at tail, if not full:

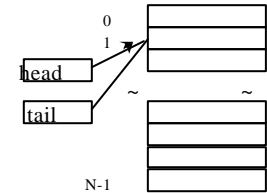
```
put data in cell tail;
tail = (tail+1) % N;
```

- Delete from head, if not empty:

```
remove data from cell head;
head = (head+1) % N;
```

## Queues Using Arrays

- When the queue is either full or empty, the 2 indices have the same value, so we cannot distinguish these two cases.
- A better way is to use an index for the head, and a count of the number of cells occupied.



## Queues Using Arrays

- Code when we use an index for head and a count:

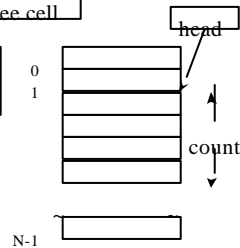
```
(head+count)%N gives the first free cell
```

- Insert at tail, if not full:

```
put data in cell (head+count)%N;
count++;
```

- Delete from head, if not empty:

```
remove data from cell head;
head = (head+1) % N;
count--;
```

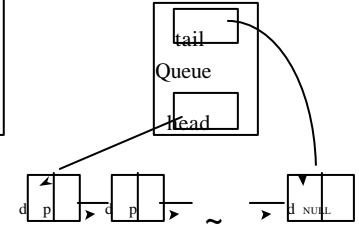


## Queues Using Linked Lists

- In a queue, data is added at one end (the tail) and removed from the other end (the head). A linked list is adequate for this purpose with the addition of a pointer to the tail:

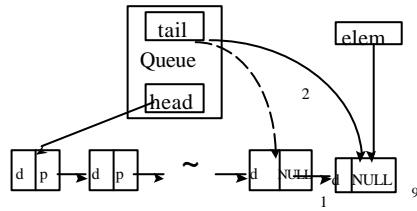
```
class Queue_node {
char * name;
int age;
~
Queue_node * next;
}
```

```
class Queue {
Queue_node * head;
Queue_node * tail;
}
```



## Inserting at the Queue Tail

```
void add_to_tail(Queue_node * elem) {
    if(q.tail == NULL) /* queue is empty */
        q.head = q.tail = elem;
    else {
        q.tail->next = elem;
        q.tail = elem;
    }
}
```



CS215 ©Peter Lo 2004

9

## Example 11A

- /\* 11A
- Operating and maintaining a queue \*/
- #include <stdio.h>
- #include <stdlib.h>
- #include <conio.h>
- struct queueNode { /\* self-referential structure \*/
- char data;
- struct queueNode \*nextPtr;
- };
- typedef struct queueNode QueueNode;
- typedef QueueNode \*QueueNodePtr;
- /\* function prototypes \*/
- void printQueue( QueueNodePtr );
- int isEmpty( QueueNodePtr );
- char dequeue( QueueNodePtr \*, QueueNodePtr \* );
- void enqueue( QueueNodePtr \*, QueueNodePtr \*, char );
- void instructions( void );

CS215 ©Peter Lo 2004

10

## Example 11A (cont')

- int main()
- {
- QueueNodePtr headPtr = NULL, tailPtr = NULL;
- int choice;
- char item;
- instructions();
- printf( "? " );
- scanf( "%d", &choice );
- while ( choice != 3 ) {
- switch( choice ) {
- case 1:
- printf( "Enter a character: " );
- scanf( "%c", &item );
- enqueue( &headPtr, &tailPtr, item );
- printQueue( headPtr );
- break;

CS215 ©Peter Lo 2004

11

## Example 11A (cont')

- case 2:
- if ( !isEmpty( headPtr ) ) {
- item = dequeue( &headPtr, &tailPtr );
- printf( "%c has been dequeued\n", item );
- }
- printQueue( headPtr );
- break;
- default:
- printf( "Invalid choice. \n\n" );
- instructions();
- break;
- }
- }
- printf( "? " );
- scanf( "%d", &choice );
- }
- printf( "End of run.\n" );
- getch();
- return 0;
- }

CS215 ©Peter Lo 2004

12

## Example 11A (cont')

```

▪ void instructions( void )
▪ { printf( "Enter your choice:\n"
▪     " 1 to add an item to the queue\n"
▪     " 2 to remove an item from the queue\n"
▪     " 3 to exit\n");
▪ }
▪ void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr, char value )
▪ { QueueNodePtr newPtr;
▪   (void *) newPtr = malloc( sizeof( QueueNode) );
▪   if ( newPtr != NULL ) {
▪     newPtr->data = value;
▪     newPtr->nextPtr = NULL;
▪     if ( isEmpty( *headPtr ) )
▪       *headPtr = newPtr;
▪     else
▪       (*tailPtr)->nextPtr = newPtr;
▪     *tailPtr = newPtr;
▪   }
▪   else
▪     printf( "%c not inserted. No memory available!\n", value );
▪ }

```

CS215 ©Peter Lo 2004

13

## Example 11A (cont')

```

▪ char dequeue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr )
▪ {
▪   char value;
▪   QueueNodePtr tempPtr;
▪
▪   value = ( *headPtr )->data;
▪   tempPtr = *headPtr;
▪   *headPtr = ( *headPtr )->nextPtr;
▪
▪   if ( *headPtr == NULL )
▪     *tailPtr = NULL;
▪
▪   free( tempPtr );
▪   return value;
▪ }
▪ int isEmpty( QueueNodePtr headPtr )
▪ {
▪   return headPtr == NULL;
▪ }

```

CS215 ©Peter Lo 2004

14

## Example 11A (cont')

```

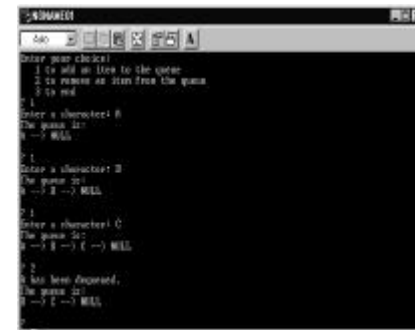
▪ void printQueue( QueueNodePtr currentPtr )
▪ {
▪   if ( currentPtr == NULL )
▪     printf( "Queue is empty.\n\n" );
▪   else {
▪     printf( "The queue is:\n" );
▪
▪     while ( currentPtr != NULL ) {
▪       printf( "%c -> ", currentPtr->data );
▪       currentPtr = currentPtr->nextPtr;
▪     }
▪
▪     printf( "NULL\n\n" );
▪   }
▪ }

```

CS215 ©Peter Lo 2004

15

## Output Result



```

-: ~$ ./queue11
Enter your choice:
 1 to add an item to the queue
 2 to remove an item from the queue
 3 to exit
1
Enter a character: A
The queue is:
A -> NULL

2
Enter a character: B
The queue is:
A -> B -> NULL

1
Enter a character: C
The queue is:
A -> B -> C -> NULL

2
A has been dequeued.
The queue is:
B -> C -> NULL

3

```

CS215 ©Peter Lo 2004

16