Queues

LECTURE 10

Objectives

- At the end of this lecture the learner is able to:
  - Explain the purpose the structure of queue
  - Understand the main operations on queue
  - Build and implement queue operations

outlines

- Introduction
- Applications of queues
- Main operations on queues
- Building queue

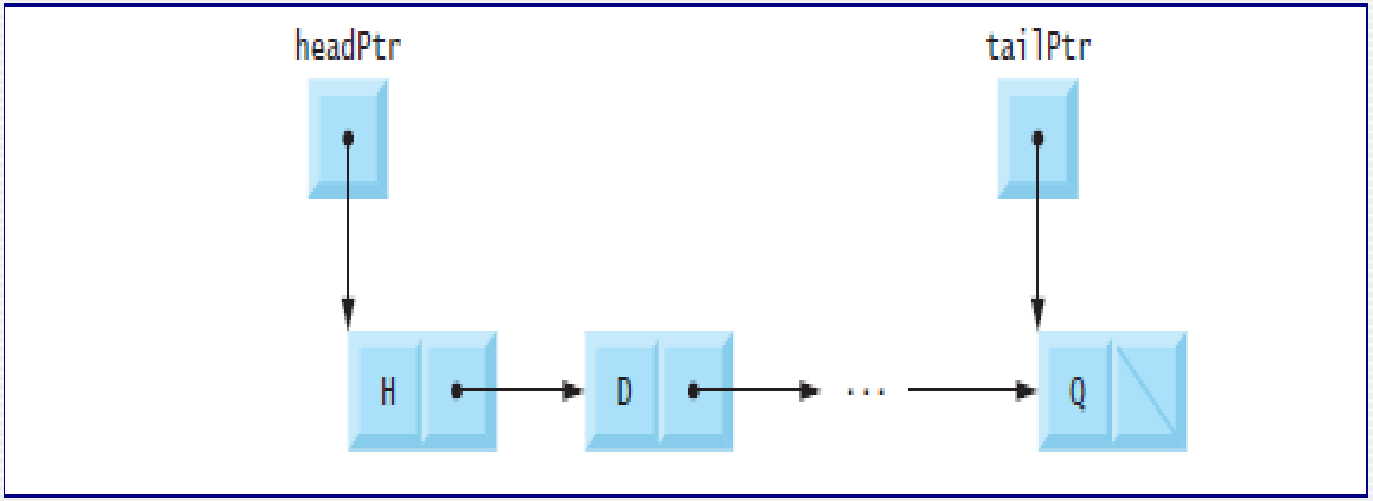
Introduction

- A queue is similar to a checkout line in a grocery store—the *first person in line is serviced first, and other customers enter the line* only at the *end and wait to be serviced.*
- Queue nodes are removed *only from the* head of **the queue and are inserted *only at the*** tail of ***the queue.***
- A queue is referred to as a **first-in, first-out (FIFO) data structure**

Applications of queues

- Support *print spooling*.
- Information packets also wait in queues in computer networks.
- Communications systems
- …etc

Main operations



Main operations

- *insert a node in the queue (function **enqueue**)*
- remove ***a node*** from the queue (function **dequeue**)
- **terminate the program.**

Main operations

```
1  // Fig. 12.13: fig12_13.c
2  // Operating and maintaining a queue
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  // self-referential structure
7  struct queueNode {
8     char data; // define data as a char
9     struct queueNode *nextPtr; // queueNode pointer
10 }; // end structure queueNode
11
12 typedef struct queueNode QueueNode;
13 typedef QueueNode *QueueNodePtr;
14
15 // function prototypes
16 void printQueue( QueueNodePtr currentPtr );
17 int isEmpty( QueueNodePtr headPtr );
18 char dequeue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr );
19 void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr,
20    char value );
21 void instructions( void );
22
```

Main operations

```
23  // function main begins program execution
24  int main( void )
25  {
26     QueueNodePtr headPtr = NULL; // initialize headPtr
27     QueueNodePtr tailPtr = NULL; // initialize tailPtr
28     unsigned int choice; // user's menu choice
29     char item; // char input by user
30
31     instructions(); // display the menu
32     printf( "%s", "? " );
33     scanf( "%u", &choice );
34
35     // while user does not enter 3
36     while ( choice != 3 ) {
37
38        switch( choice ) {
39           // enqueue value
40           case 1:
41              printf( "%s", "Enter a character: " );
42              scanf( "\n%c", &item );
43              enqueue( &headPtr, &tailPtr, item );
44              printQueue( headPtr );
45              break;
46           // dequeue value
47           case 2:
48              // if queue is not empty
49              if ( !isEmpty( headPtr ) ) {
50                 item = dequeue( &headPtr, &tailPtr );
51                 printf( "%c has been dequeued.\n", item );
52              } // end if
```

Main operations

```
53
54                printQueue( headPtr );
55                break;
56             default:
57                puts( "Invalid choice.\n" );
58                instructions();
59                break;
60          } // end switch
61
62          printf( "%s", "? " );
63          scanf( "%u", &choice );
64       } // end while
65
66       puts( "End of run." );
67    } // end main
68
69    // display program instructions to user
70    void instructions( void )
71    {
72       printf ( "Enter your choice:\n"
73                "   1 to add an item to the queue\n"
74                "   2 to remove an item from the queue\n"
75                "   3 to end\n" );
76    } // end function instructions
77
```

Main operations

```
78  // insert a node in at queue tail
79  void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr,
80     char value )
81  {
82     QueueNodePtr newPtr; // pointer to new node
83
84     newPtr = malloc( sizeof( QueueNode ) );
85
86     if ( newPtr != NULL ) { // is space available
87        newPtr->data = value;
88        newPtr->nextPtr = NULL;
89
90        // if empty, insert node at head
91        if ( isEmpty( *headPtr ) ) {
92           *headPtr = newPtr;
93        } // end if
94        else {
95           ( *tailPtr )->nextPtr = newPtr;
96        } // end else
97
98        *tailPtr = newPtr;
99     } // end if
100    else {
101       printf( "%c not inserted. No memory available.\n", value );
102    } // end else
103 } // end function enqueue
104
```

Main operations

```
105 // remove node from queue head
106 char dequeue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr )
107 {
108    char value; // node value
109    QueueNodePtr tempPtr; // temporary node pointer
110
111    value = ( *headPtr )->data;
112    tempPtr = *headPtr;
113    *headPtr = ( *headPtr )->nextPtr;
114
115    // if queue is empty
116    if ( *headPtr == NULL ) {
117       *tailPtr = NULL;
118    } // end if
119
120    free( tempPtr );
121    return value;
122 } // end function dequeue
123
124 // return 1 if the queue is empty, 0 otherwise
125 int isEmpty( QueueNodePtr headPtr )
126 {
127    return headPtr == NULL;
128 } // end function isEmpty
129
```

Main operations

```
130  // print the queue
131  void printQueue( QueueNodePtr currentPtr )
132  {
133     // if queue is empty
134     if ( currentPtr == NULL ) {
135        puts( "Queue is empty.\n" );
136     } // end if
137     else {
138        puts( "The queue is:" );
139
140        // while not end of queue
141        while ( currentPtr != NULL ) {
142           printf( "%c --> ", currentPtr->data );
143           currentPtr = currentPtr->nextPtr;
144        } // end while
145
146        puts( "NULL\n" );
147     } // end else
148  } // end function printQueue
```

```
? 1
Enter a character: A
The queue is:
A --> NULL

? 1
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

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

? 2
C has been dequeued.
Queue is empty.

? 2
Queue is empty.

? 4
Invalid choice.

Enter your choice:
   1 to add an item to the queue
   2 to remove an item from the queue
   3 to end
? 3
End of run.
```

Function dequeue

- Function dequeue (lines 106–122) receives the *address of the pointer to the head of the queue*
- and the *address of the pointer to the tail of the queue as arguments and removes the first node*
- from the queue. The dequeue operation consists of six steps:
- **1. Assign (*headPtr)->data to value to save the data (line 111).**
- **2. Assign *headPtr to tempPtr (line 112), which will be used to free the unneeded** memory.
- **3. Assign (*headPtr)->nextPtr to *headPtr (line 113) so that *headPtr now**
- points to the new first node in the queue.
- **4. If *headPtr is NULL (line 116), assign NULL to *tailPtr (line 117) because the** queue is now empty.
- **5. Free the memory pointed to by tempPtr (line 120).**
- **6. Return value to the caller (line 121).**

Function enqueue

- The function consists of three steps:
  - **1. To create a new node: Call malloc, assign the allocated memory location to** newPtr (line 84), assign the value to be inserted in the queue to newPtr->data (line 87) and assign NULL to newPtr->nextPtr (line 88).
  - **2. If the queue is empty (line 91), assign newPtr to *headPtr (line 92), because the** new node will be both the head and tail of the queue; otherwise, assign pointer

Function enqueue

- newPtr to (*tailPtr)->nextPtr (line 95), because the new node will be placed after the previous tail node.

- **3. Assign newPtr to *tailPtr (line 98), because the new node is the queue's tail.**

End