Queues

September 28, 2020

Overview

- What are queues?
- Where are queues used?
- Queue behavior
- Example (with code)

What are Queues?

- In English, a queue is defined as a waiting line, like a line of people waiting at a supermarket checkout stand where the first person in line is the first person to be served.
- In computer applications, a queue is defined as a list in which all additions are made at one end, and all deletions are made at the other.
- Queues are also called *first-in, first-out* lists, or FIFO data structures for short.

Standard Queue Operation Names

The standard names for Queue operations are:

- Enqueue – Adding an item to the queue.
- Dequeue – Removal of an item from the queue.

Naming Quiz

- Queues are also called *first-in, first-out* lists, or FIFO for short.
- What do we call a stack? (LIFO (*last-in, first-out*) lists)

Where are queues used?

- Process scheduling
- Simulation (shopping, banks, amusement parks, etc.)
- Computing
  - Instruction Queue on a processor
  - Operating System (commands/event processing)
  - Communication (I/O)
  - Printing
  - Multimedia

Queues are frequently used for *Resource Management* in many applications.

Queue Implementations

Queues (like stacks) may be implemented in several different ways. Common implementation methods include:

- array
- linear linked list
- circular array
- circular linked list

Queue Test Program


```
/*  testQueue.cpp
 */

#include <iostream>

using namespace std;

#include "queueL.h"

int main()
{
    Queue q1;
    int qVal;

      //  add some initial nodes
    q1.Insert( 3 );
    q1.Insert( 5 );

    cout << "Initial contents of q1:" << endl;
    q1.Print();
```

Queue Test Program (2)


```
    //  add a few more nodes
    q1.Insert( 1 );

    cout << "Contents of q1 after adding:" << endl;
    q1.Print();

    //  delete a few items
    cout << "\nDeleting two items from queue:" << endl;

    qVal = q1.Delete();
    cout << "Contents of q1 after deleting one item:" << endl;
    q1.Print();

    qVal = q1.Delete();
    cout << "Contents of q1 after deleting one item:" << endl;
    q1.Print();
}
```

Output from Queue Test Program


```
Initial contents of q1:
3
5
Contents of q1 after adding:
3
5
1

Deleting two items from queue:
Contents of q1 after deleting one item:
5
1
Contents of q1 after deleting one item:
1
```

Queue Interface


```
/*  queueL.h

    Interface file for the ADT queue.
    Implementation uses a list object.
 */

#ifndef QUEUE_L_H
#define QUEUE_L_H

#include "listQ.h"

typedef int QueueItemType;

class Queue
{
private:
    LinkedList  L;   //  list of queue items
```

Queue Interface (2)


```
public:
    Queue();
    Queue( const Queue& q );
    ~Queue();

    void Insert( QueueItemType newItem );
    int Delete();

    int  GetFront();
    int  GetEnd();

    bool IsEmpty();

    void Print();
};

#endif
```

Queue Class—Comments

Note:

- Nonstandard method names!
  - `Insert` should be `Enqueue`
  - `Delete` should be `Dequeue`

Queue Class—Implementation


```
/*  queueL.cpp

    Definition file for the ADT queue.
    Implementation uses a list object.
 */

#include "queueL.h"

Queue::Queue()
{
    //  default constructor
}

Queue::Queue( const Queue& q ) : L(q.L)
{
    //  copy constructor
}

Queue::~Queue()
{
    //  destructor
}
```

Queue Implementation (2)


```
void Queue::Insert( QueueItemType newItem )
{
    L.AddNode( newItem );
}

int Queue::Delete()
{
    int iVal;

    iVal = L.FirstNode();
    L.DeleteNode();

    return iVal;
}
```

Queue Implementation (3)


```
int Queue::GetFront()
{
    L.FirstNode();
}

int Queue::GetEnd()
{
    return  L.LastNode();
}
```

Queue Implementation (4)


```
bool Queue::IsEmpty()
{
    int  length = L.Size();
    return bool(length == 0 );
}

void Queue::Print()
{
    L.PrintNodes();
}
```

List Class—Interface


```
/*  listQ.h

    Class interface for a linked list of integers
    used in queues.
 */

#include <iostream.h>

class LinkedList
{
private:

  struct node {
     int info;
     node * next;
  };

  typedef node * NodePtr;

  NodePtr start;     //  pointer to front
  NodePtr end;       //  pointer to end
```

List Class—Interface (2)


```
public:
        // Constructor
   LinkedList()
   {
      start = NULL;
      end   = start;
      count = 0;
   }

       // Destructor
   ~LinkedList()
   {
      NodePtr p = start, n;

      while (p != NULL)
      {
         n = p;
         p = p->next;
         delete n;
      }
   }
```

List Class—Interface (3)


```
   // Put a node at the end of the linked list.
  void AddNode(int x);

   //  Delete the first node in the list.
  void DeleteNode();

   //  Return the first or last node.
  int  FirstNode();
  int  LastNode();

   //  Output the values in the nodes, one integer per line.
  void PrintNodes();

   //  Return true if a node with the value x is in the list.
  bool IsInList(int x);

   //  Return a count of the number of nodes in the list.
  int Size();
};
```

List Class—Implementation


```
/*  listQ.cpp

    Class for a sorted linked list of integers.
 */

#include "listQ.h"

void LinkedList::AddNode(int x)
{
    NodePtr n = new node;
    n->info = x;
    n->next = NULL;
    count++;

    if( start == NULL ) {
        start = n;
        end   = start;
    } else {
        end->next = n;
        end = end->next;
    }
}
```

List Class—Comments

Notes

- Comment says that it is a sorted linked list.
  - *Uses a double-ended list in actuality.*
- Why a *double-ended* list?
  - Simplifies code and more efficient.
- `start` used for `head` node name.

List Class—Implementation (2)


```
void LinkedList::DeleteNode()
{
    NodePtr curr;

    if( start != NULL )
    {
        curr = start;
        start = start->next;
        delete curr;
    }
}

int LinkedList::FirstNode()
{
    int  iVal;

    if( start != NULL )
        iVal = start->info;

    return iVal;
}
```

List Class—Implementation (3)


```
int LinkedList::LastNode()
{
    if( end != NULL )
        return end->info;
}

void LinkedList::PrintNodes()
{
    NodePtr p = start;

    while( p != NULL )
    {
        cout << p->info << endl;
        p = p->next;
    }
}
```

List Class—Implementation (4)


```
bool LinkedList::IsInList(int x)
{
    NodePtr p = start;

    while (p != NULL && x > p->info)
        p = p->next;

    return (x == p->info);
}

int LinkedList::Size()
{
    return count;
}
```