```
// Public Interface Header file for the ADT queue.
typedef desired-type-of-queue-item queueItemType;

class queueClass
{
public:
   queueClass();                          // default constructor
   queueClass(const queueClass& Q);       // copy constructor
   ~queueClass();                         // destructor

   bool QueueIsEmpty() const;
   // Determines whether a queue is empty.
   // Precondition: None.
   // Postcondition: Returns true if the queue is empty;
   //     otherwise returns false.

   void QueueInsert(queueItemType NewItem, bool& Success);
   // Inserts an item at the back of a queue.
   // Precondition: NewItem is the item to be inserted.
   // Postcondition: If insertion was successful, NewItem
   //     is at the back of the queue and Success is true;
   //     otherwise Success is false.

   void QueueDelete(bool& Success);
   // Deletes the front of a queue.
   // Precondition: None.
   // Postcondition: If the queue was not empty, the item
   //     that was added to the queue earliest is deleted and
   //     Success is true. However, if the queue was empty,
   //     deletion is impossible and Success is false.

   void QueueDelete(queueItemType& QueueFront, bool& Success);
   // Retrieves and deletes the front of a queue.
   // Precondition: None.
   // Postcondition: If the queue was not empty, QueueFront
   //     contains the item that was added to the queue
   //     earliest, the item is deleted, and Success is true.
   //     However, if the queue was empty, deletion is
   //     impossible and Success is false.

   void GetQueueFront(queueItemType& QueueFront,
                      bool& Success) const;
   // Retrieves the item at the front of a queue.
   // Precondition: None.
   // Postcondition: If the queue was not empty, QueueFront
   //     contains the item that was added to the queue earliest
   //     and Success is true. However, if the queue was empty,
   //     the operation fails, QueueFront is unchanged, and
   //     Success is false. The queue is unchanged.
private:
```

```
// ********************************************************
// Header file QueueL.h for the ADT queue.
// ADT list implementation.
// ********************************************************

#include "ListP.h"  //  ADT list operations

typedef listItemType queueItemType;

class queueClass
{
public:
// constructors and destructor:
   queueClass();                        // default constructor
   queueClass(const queueClass& Q);     // copy constructor
   ~queueClass();                       // destructor

// queue operations:
   bool QueueIsEmpty() const;

   void QueueInsert(queueItemType NewItem, bool& Success);

   void QueueDelete(bool& Success);

   void QueueDelete(queueItemType& QueueFront,
                    bool& Success);

   void GetQueueFront(queueItemType& QueueFront,
                      bool& Success) const;

private:

   // list of queue items
   listClass L;

};
```

```
// ********************************************************
// Implementation file QueueL.cpp for the ADT queue.
// ADT list implementation.
// ********************************************************

#include "QueueL.h"  // header file

queueClass::queueClass()
{
}  // end default constructor

queueClass::queueClass(const queueClass& Q): L(Q.L)
{
}  // end copy constructor

queueClass::~queueClass()
{
}  // end destructor

bool queueClass::QueueIsEmpty() const
{
   return bool(L.ListLength() == 0);
}

void queueClass::QueueInsert(queueItemType NewItem,
                             bool& Success)
{
   L.ListInsert(L.ListLength()+1, NewItem, Success);
}

void queueClass::QueueDelete(bool& Success)
{
   L.ListDelete(1, Success);
}

void queueClass::QueueDelete(queueItemType& QueueFront,
                             bool& Success)
{
   L.ListRetrieve(1, QueueFront, Success);
   if (Success)
      L.ListDelete(1, Success);
}

void queueClass::GetQueueFront(queueItemType& QueueFront,
                               bool& Success) const
{
   L.ListRetrieve(1, QueueFront, Success);
}
```

```
// ********************************************************
// Header file QueueP.h for the ADT queue.
// Pointer-based implementation.
// ********************************************************

typedef desired-type-of-queue-item queueItemType;

struct queueNode;              // defined in implementation file
typedef queueNode* ptrNode;    // pointer to node

class queueClass
{
public:
// constructors and destructor:
   queueClass();                       // default constructor
   queueClass(const queueClass& Q);    // copy constructor
   ~queueClass();                      // destructor

// queue operations:
   bool QueueIsEmpty() const;

   void QueueInsert(queueItemType NewItem, bool& Success);

   void QueueDelete(bool& Success);

   void QueueDelete(queueItemType& QueueFront,
                    bool& Success);

   void GetQueueFront(queueItemType& QueueFront,
                      bool& Success) const;

private:

   // points to last item in the queue
   ptrNode BackPtr;

};
```

```
// ********************************************************
// Implementation file QueueP.cpp for the ADT queue.
// Pointer-based implementation.
// ********************************************************
#include <stddef.h>  // for NULL

#include "QueueP.h"  // header file

// The queue is implemented as a circular linked list
// with one external pointer to the back of the queue.

struct queueNode
{
   queueItemType Item;
   ptrNode       Next;
};

queueClass::queueClass() : BackPtr(NULL)     // default constr
{
}

queueClass::queueClass(const queueClass& Q)  // copy constructor
{
   // Implementation left as an exercise (Exercise 7.4).
   // Similar to StackP example
}

queueClass::~queueClass()                    // destructor
{
   bool Success;

   while (!QueueIsEmpty())
      QueueDelete(Success);
}

bool queueClass::QueueIsEmpty() const
{
   return bool(BackPtr == NULL);
}

void queueClass::GetQueueFront(queueItemType& QueueFront,
                               bool& Success) const
{
   Success = bool(!QueueIsEmpty());

   if (Success)
   {
      QueueFront = (BackPtr->Next)->Item;
   }
}
```

```
void queueClass::QueueInsert(queueItemType NewItem,
                             bool& Success)
{
   // create a new node
   ptrNode NewPtr = new queueNode;

   Success = bool(NewPtr != NULL);  // check allocation
   if (Success)
   {
      // set data portion of new node
      NewPtr->Item = NewItem;

      // insert the new node
      if (QueueIsEmpty())     // insertion into empty queue
      {
         NewPtr->Next = NewPtr;              // link to itself
      }
      else                    // insertion into nonempty queue
      {
         NewPtr->Next = BackPtr->Next;       // link to back
         BackPtr->Next = NewPtr;
      }

      BackPtr = NewPtr;       // new node is at back
   }
}
```

```
void queueClass::QueueDelete(bool& Success)
{
   Success = bool(!QueueIsEmpty());

   if (Success)
   {
      // queue is not empty; remove front
      ptrNode FrontPtr = BackPtr->Next;

      if (FrontPtr == BackPtr)    // special case?
         BackPtr = NULL;          // yes, one node in queue
      else
         BackPtr->Next = FrontPtr->Next;

      FrontPtr->Next = NULL;  // defensive strategy
      delete FrontPtr;
   }
}


void queueClass::QueueDelete(queueItemType& QueueFront,
                             bool& Success)
{
   Success = bool(!QueueIsEmpty());

   if (Success)
   {
      // queue is not empty; retrieve front
      ptrNode FrontPtr = BackPtr->Next;
      QueueFront = FrontPtr->Item;

      QueueDelete(Success);  // delete front
   }
}
```

```
// ********************************************************
// Header file QueueA.h for the ADT queue.
// Array-based implementation.
// ********************************************************

const int MAX_QUEUE = maximum-size-of-queue;

typedef desired-type-of-queue-item queueItemType;

class queueClass
{
public:
// constructors and destructor:

   queueClass();  // default constructor

   // copy constructor and destructor are
   // supplied by the compiler

// queue operations:
   bool QueueIsEmpty() const;

   void QueueInsert(queueItemType NewItem, bool& Success);

   void QueueDelete(bool& Success);

   void QueueDelete(queueItemType& QueueFront,
                    bool& Success);

   void GetQueueFront(queueItemType& QueueFront,
                      bool& Success) const;

private:

   queueItemType Items[MAX_QUEUE];

   int           Front;
   int           Back;

   int           Count;

};
```

```
// ********************************************************
// Implementation file QueueA.cpp for the ADT queue.
//
// Circular array-based implementation.
//
// The array has indexes to the front and back of the queue.
// A counter tracks the number of items currently in the queue.
// ********************************************************

#include "QueueA.h"  // header file


queueClass::queueClass():           // default constructor
    Front(0), Back(MAX_QUEUE-1), Count(0)
{
}


bool queueClass::QueueIsEmpty() const
{
   return bool(Count == 0);
}


void queueClass::GetQueueFront(queueItemType& QueueFront,
                               bool& Success) const
{
   Success = bool(!QueueIsEmpty());

   if (Success)
      QueueFront = Items[Front];
}


void queueClass::QueueInsert(queueItemType NewItem,
                             bool& Success)
{
   Success = bool(Count < MAX_QUEUE);

   if (Success)
   {
      Back = (Back+1) % MAX_QUEUE;
      Items[Back] = NewItem;

      ++Count;
   }
}
```

```
void queueClass::QueueDelete(bool& Success)
{
   Success = bool(!QueueIsEmpty());

   if (Success)
   {
      Front = (Front+1) % MAX_QUEUE;

      --Count;
   }
}


void queueClass::QueueDelete(queueItemType& QueueFront,
                             bool& Success)
{
   Success = bool(!QueueIsEmpty());

   if (Success)
   {
      QueueFront = Items[Front];
      Front = (Front+1) % MAX_QUEUE;

      --Count;
   }
}
```