```
struct QueueException {                    //Queue Exception
    QueueException(char* m)
    {   cout << endl;
        cout << "I am the Queue and I am " << m << ".";
        cout << endl;
    }
template <class T>
class Queue {
    struct Element {
        Element* newer;                    //Points to the next newer element
        T value;                           //Data value of type T queued
        Element(T v):newer(0),value(v){}   //Constructor
    };
    Element* oldest;                       //Points to the oldest element
    Element* newest;                       //Points to the newest element
    int count;                             //Records the number of elements on the queue.
    void Kill(Element* e)                  //Reclaims storage for elements remaining on
    {   while (e) {                        //the Q
            Element* p=e;
            e = e-> newer;
            delete p;
        } // end of while loop
    } // end of Kill
public:
    Queue():oldest(0),newest(0),count(0){} //Contructor
                                           //Destructor
    ~Queue(){Kill(oldest);oldest=newest=0;count=0;}
    bool IsEmpty(void){return count==0;}   //Is the Queue empty?
    bool IsFull(void) { return false; }    //Is the Queue full?
    void Enq(T v)                          //Store value v on the newest end of the Queue
    {   if (IsFull()) throw QueueException("full");
        Element* e=new Element(v);
        if (IsEmpty()) oldest=e; else newest->newer=e;
        newest=e;
        count++;
    } // end of Enq
```

Figure 1. template class *Queue* (element implementation)

```
    T Deq(void)                          //Remove and return the oldest value on the Q
    {   if (IsEmpty()) throw QueueException("empty");
        Element* e=oldest;
        T v=e->value;
        if (Length()==1) newest=0;
        oldest=e->newer;
        count--;
        delete e;
        return v;
    } // end of Deq
    int Length(void){return count;}      //Return the number of elements on the Queue
};
```

Figure 1. template class *Queue* (element implementation) (continued)

```
ifndef Queue_h
#define Queue_h 1
struct QueueException {                    //Queue Exception
    QueueException(char* m)
    {   cout << endl;
        cout << "I am the Queue and I am " << m << ".";
        cout << endl;
    }
template <class T>
class Queue {
    struct Element {
        Element* newer;                    //Points to the next newer element
        T value;                           //Data value of type T queued
        Element(T v):newer(0),value(v){}   //Constructor
    };
    Element* oldest;                       //Points to the oldest element
    Element* newest;                       //Points to the newest element
    int count;                             //Records the number of elements on the queue.
    void Kill(Element* e)                  //Reclaims storage for elements remaining on
    the {   while (e) {
            Element* p=e;
            e = e-> newer;
            delete p;
        } // end of while loop
    } // end of Kill
public:
    Queue():oldest(0),newest(0),count(0){} //Contructor
                                           //Destructor
    ~Queue(){Kill(oldest);oldest=newest=0;count=0;}
    bool IsEmpty(void){return count==0;}   //Is the Queue empty?
    bool IsFull(void) { return false; }    //Is the Queue full?
    void Enq(T v)                          //Store value v on the newest end of the Queue
    {   if (IsFull()) throw QueueException("full");
        Element* e=new Element(v);
        if (IsEmpty()) oldest=e; else newest->newer=e;
        newest=e;
        count++;
    } // end of Enq
```

Figure 2. File **Queue.h**

```
    T Deq(void)                          //Remove and return the oldest value on the Queue
    {   if (IsEmpty()) throw QueueException("empty");
        Element* e=oldest;
        T v=e->value;
        if (Length()==1) newest=0;
        oldest=e->newer;
        count--;
        delete e;
        return v;
    } // end of Deq
    int Length(void){return count;}      //Return the number of elements on the Queue
};
#endif
```

Figure 2. File **Queue.h (continued)**

Example: Program **p01** stores and retrieves the characters of string "Alan Turing" illustrating the first-in-first-out character of a queue.

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;
#include "Queue.h"
int main()
{   string t="Alan Turing";
    Queue <char> Q;
    for (int a=0;a<t.length();a++) Q.Enq(t[a]);
    cout << endl;
    while (!Q.IsEmpty()) cout << Q.Deq();
    cout << endl;
    return 0;
}
```

Figure 3. File **p01.cpp**

Program **p01** prints:
Alan Turing