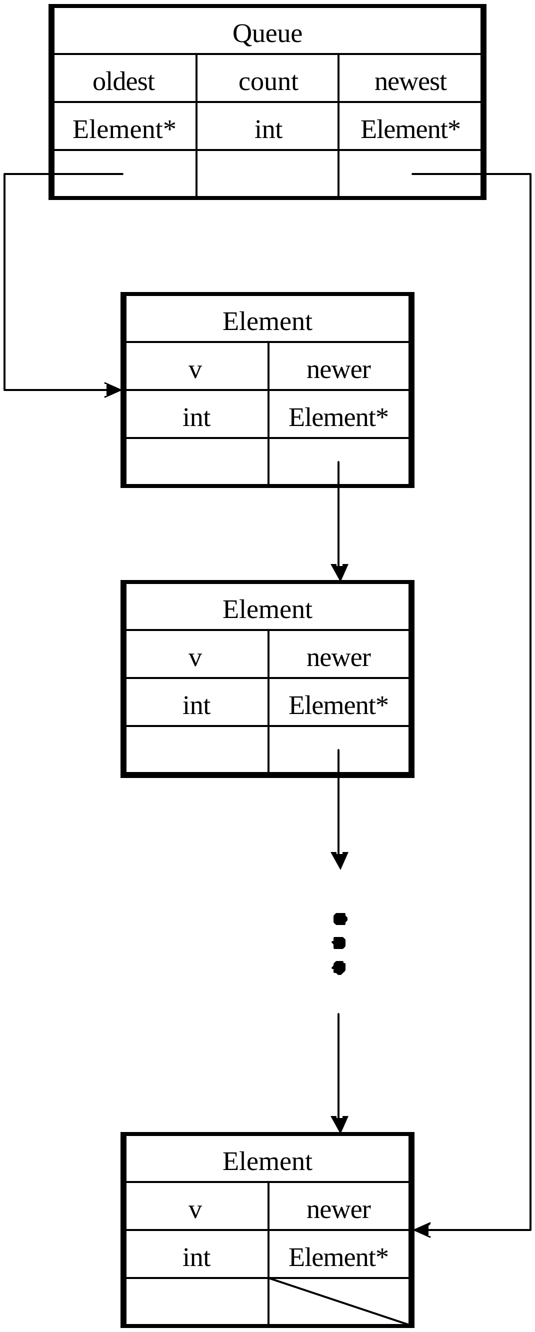

Figure 1. Queue implemented using dynamically allocated elements.

Notes for Figure 1.

1. **class** *Queue* has two data members, *oldest* and *newest*. Members *oldest* and *newest* point to the oldest and newest elements on the queue respectively.
2. Elements are linked via member *newer* in **struct** *Element*. Elements are linked to permit the following operations.
   - 2.1. A new element can be placed on the newest end of the queue.
   - 2.2. The oldest element can be removed.
   - 2.3. Elements can be traversed from the newest element to the oldest element.

Member functions of **struct** *Element*:

```
class Queue {
    struct Element {
        Element* newer;
        int v;
        Element(int val);
    };
    Element* newest;
    Element* oldest;
    int count;

    void Kill(Element* e);
public:
    Queue();
    ~Queue();
    bool IsEmpty(void);
    bool IsFull(void);
    void Enq(int v);
    int Deq(void);
    int Length(void);
};
```

Figure 2. Specifications for **class** *Queue*.

1. *Element*(**int** *val*) - Function *Element* assigns the value of parameter *val* to member *v*. Function *Element* assigns a NULL-value to member *newer*. The newest element terminates the list of element on the queue.

Member functions of **class** *Queue*:

2. *Queue()* - Function *Queue* creates an empty queue. Function *Queue* assigns a NULL-value to pointers to the oldest and newest elements on the queue. Member *count* is set to zero. Member *newest* points to the most recently added element. Member *oldest* points to the element that has been on the queue the longest.
3. *~Queue()* - Function *~Queue* calls private member function *Kill* to free all dynamically allocated elements on the queue. After function *Kill* reclaims storage member *count* is set to zero.
4. *Kill* – Function *Kill* removes all remaining elements on the queue. Function *Kill* traverses the queue moving from the oldest element to the newest element. As each element is visited, it is deleted and storage is reclaimed.
5. *Enq* - Function *Eng* creates a new element and inserts on the newest end of the queue.
6. *Deq* - Function *Deq* removes an element from the oldest end of the queue and returns the value in the element to the caller.
7. *IsEmpty* - Function *IsEmpty* determines if the queue is empty.
8. *IsFull* - Function *IsFull* determines if the queue is full.
9. *Length* - Function *Length* returns the number of elements on the queue.

Queue::*Queue*();

1. Assign a NULL pointer to member *newest*.
2. Assign a NULL pointer to member *oldest*.
3. Assign zero to member *count*.

Notes for construction function *Queue*()

1. Recall that a **class** and **struct**ure are essentially identical. The only difference between a **class** and a **struct**ure is that members of a class are **private** by default and members of a structure are **public** by default.

Queue::~*Queue*();

1. Kill the elements on the queue. Reset the *count* to zero.

void *Queue*::*Kill*(*Element** *e*);

1. Parameter *e* points to the oldest element on the queue when me mber function *Kill* is called.
   1.1. Does parameter *e* point to an element on the queue? If the answer is yes then perform steps 1.2 through 1.6. If the answer is no then perform step 2.
   1.2. Create local variable *p* to point to the same element referenced by parameter *e*.
   1.3. Make parameter *e* point to the next newer element.
   1.4. Free the element referenced by local variable *p*.
   1.5. Repeat step 1.
2. Return to the caller. The queue is empty.

Member function *Kill* notes:

1. Member function *Kill* is a private member function because only members of **class** *Queue* should be permitted to remove every element on the queue.
2. Parameter *e* of function *Kill* points to the oldest member when it is called. Elements on the queue are deleted traversing the queue from the oldest element to the newest element.
3. The *while-statement* determines if parameter *e* points to an element. The element of the queue referenced by parameter *e* will be deleted. The loop defined by the *while-statement* terminates when parameter *e* is assigned the NULL-value. Parameter *e* is assigned a NULL-value when the NULL-value terminating the list in the newest element is assigned to parameter *e* by the statement

   e=*e*->*newer*.

   Elements on the queue are deleted using a four-step process illustrated in Figure 4.
   3.1. Diagram 1 shows the queue as function *Kill* sees it before any elements have been deleted. Upon entry to function *Kill*, parameter *e* points to the oldest element as shown in diagram 1.
   3.2. Local variable *p* is created in diagram 2. The oldest element is about to be deleted. Local variable *p* is made to point to the oldest element also.
   3.3. Parameter *e* is moved to the next element to be deleted. This step is essential. If parameter *e* is not moved to the next element, remaining elements will be lost. There is no way to access remaining elements once the chain defined by member *newer* is broken. Please refer to diagram 3.
   3.4. Storage is reclaimed. Local variable *p* goes out of scope at the end of the loop defined by the *while-statement*. When a variable goes out of scope it is destroyed. Parameter *e* points to the next element to be deleted. The process described above is repeated for every element on the queue.

void *Queue*::*Enq*(**int** *v*);

1. Throw exception *QueueException* if the queue is full. Note that function *QueueException* does not return. Statements following this check depend on the queue having at least one element.
2. Allocate a new element and assign a pointer to the new element to local variable *e*.
3. If the queue is empty assign a pointer to the element to member *oldest*.
4. If the queue has one or more elements assign the pointer to the new element to member *newer* in the element currently referenced by member *newest*.
5. Assign the pointer to the new ele ment to member *newest*.
6. Increment the value of member *count*.

Member function *Enq* notes:

1. Member function *Enq* always places the new element at the newest end of the queue. Member *newest* always points to the element that was most recently placed on the queue.

int *Queue*::*Deq***(void)**

1. Throw function *QueueException* if the queue is empty. Note that function *QueueException* does not return. Statements following this check depend on the queue having at least one element.
2. Assign a pointer to the oldest element on the queue to local variable *e*.
3. Extract the value of member *v* from the oldest element on the queue and assign it to local variable *v*.
4. If there is exactly one element on the queue, assign a NULL value to member *newest*.
5. Assign the value of member *newer* in the oldest element on the queue to member *oldest* in the anchor.
6. Free the oldest member on the queue.
7. Decrement member *count*.
8. Return the value of local variable *v*.

bool *Queue*::*IsEmpty***(void)**

1. The queue is empty if the value of member *count* is zero.

bool *Queue*::*IsFull***(void)**

1. The queue is never full.

int *Queue*::*Length***(void)**

1. Return the value of member *count*.