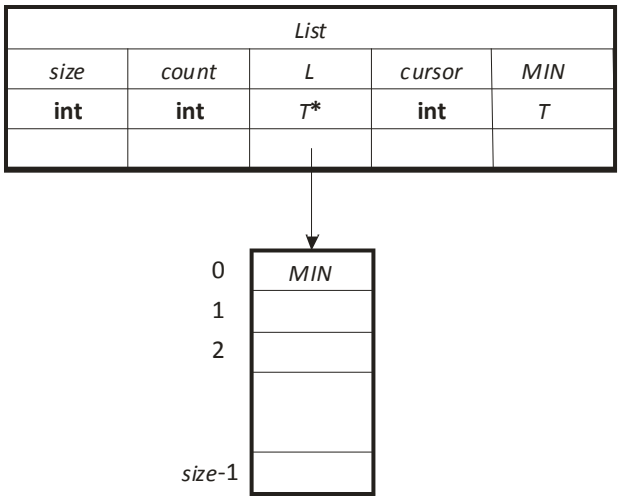

Figure 1. Member data for **class** *List*.

1. An ordered list is a sequence of items in some order. The list presented in this discussion contains a sequence of values of type *T* in ascending order.
2. A sentinel is employed to make the insertion and removal of a value simpler. The sentinel is the minimum value (*MIN*) and is placed at the beginning of an ascending sequence.
3. An element in the list can be found using a binary search. A *key* is given to the function performing the search and the index of the matching key is returned. If the list does not contain an element that matches the key, an invalid index is returned, usually zero (0).
4. A value is inserted in the list by shifting elements larger than the input value toward the end of the list.
5. A value is removed from the list by removing the element and shifting elements larger than the element removed one position toward the front of the list.
6. Support for iterating through the list is implemented by adding member *cursor* that records the current position in the anchor for the list.
7. Member data for **class** *List* are shown in Figure 1.
   - 7.1. Member *L* points to a dynamically allocated array used to store values of type *T* in the list and the sentinel.
   - 7.2. Member *size* records the number of elements available to store values on the list. Member *size* includes the storage allocated for the sentinel.
   - 7.3. Member *count* records the number of integers currently stored in the list. Member *count* does not include the sentinel.
   - 7.4. Member *cursor* records the current position.
   - 7.5. Member *MIN* is the value assigned to the sentinel. Member *MIN* is initialized to the smallest value of type *T*. Maximum and minimum values for types native to C++ can be found in **#include <***limits***>**.
   - 7.6. Constructor *List***(***T m***,int** *sz***=100)** initializes **class** *List*, allocates storage for the list, and assigns the sentinel to element *L***[0]**.
   - 7.7. Constructor *List***(***T m***,***istream***& *i***,int** *sz***=100)** initializes **class** *List*, allocates storage for the list, and assigns the sentinel to element *L***[0]** and scans input stream *i* for strings having type *T* to store in elements of the list.
   - 7.8. Destructor **~***List***()** reclaims storage allocated by the constructor.
   - 7.9. Function *IsFull* determines if there is space for another key.
   - 7.10. Function *Insert* inserts a unique *key* into the list.

7.11. Function *Remove* deletes a *key* from the list. If the *key* is not in the list no action is taken.
7.12. Function *Print* prints values on the list separated by commas and enclosed in curly braces as shown *title* $\{i_1, i_2, \cdots, i_{count}\}$.
7.13. Function *Scan* scans input stream *i* for strings having type *T* to store in elements of the list in ascending order.
7.14. Function *First* puts the cursor on the smallest element in the list.
7.15. Function *IsEol* determines if the cursor is past the last element in the list.
7.16. Function *Next* puts the cursor on the next largest element on the list.
7.17. Function *Member* returns the value of the current element as determined by the cursor.
7.18. Function *IsMember* determines if parameter *key* is a member of the list.
7.19. Function *Index* returns the index of parameter *key* or zero if the *key* is not in the list.

```
struct ListException {
    ListException(char* m);
};
template <class T>
class List {
    int size;
    int* L;
    int count;
    int cursor;
    const T MIN;
    int Index(int key);
public:
    List(T m,int sz=100);
    List(T m,istream& i,int sz=100);
    ~List(T m);
    bool IsFull(void);
    void Insert(T key);
    void Remove(T key);
    void Scan(istream& i);
    void Print(ostream& o,char* title);
    void First(void);
    bool IsEol(void);
    void Next(void);
    T Member(void);
    bool IsMember(T key);
};
```

Figure 2. class *List*.

```
List(T m,int sz=100):size(sz),count(0),cursor(0),MIN(m){L=new T[size];L[0]=MIN;}
```

Figure 3. Constructor *List*(*T m*,**int** *sz*=**100**)

```
List(T m,int sz=100):size(sz),count(0),cursor(0),MIN(m)
{    L=new T[size];L[0]=MIN;
     Scan(i);
}
```

Figure 4. Constructor *List*(*T m*,*istream*& *i*,**int** *sz*=**100**).

```
~List(){if (L) delete[] L;}
```

Figure 5. Destructor ~*List*().

```
bool IsFull(void){return count>size-1;}
```

Figure 6. Function *IsFull*.

A sentinel is placed at the beginning of the list. The sentinel is the smallest value. The sentinel is smaller than any value in the list. The sentinel is placed in element zero of array *L*. The shaded rectangle in element zero represents the sentinel as shown in Figure 7.

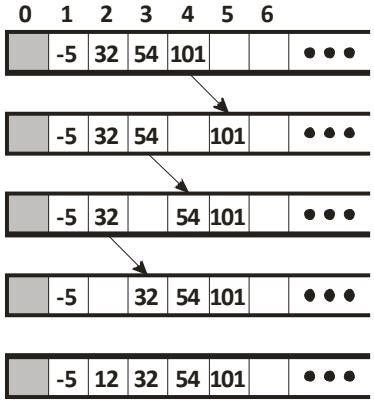

Figure 7. Insert key 12

Member function *IsMember* can be used to determine if a key exists. The code in figure 8 contains the guard against inserting a duplicate key.

Member function *IsFull* is used to prevent overfilling the array containing keys as shown in figure 8.

```
void Insert(T key)
{    if (IsAMember(key)) return;
     if (IsFull()) throw ListException("full");
     int i;
     for (i=++count;key<L[i-1];i--) L[i]=L[i-1];
     L[i]=key;
}
```

Figure 8. Function *Insert*.

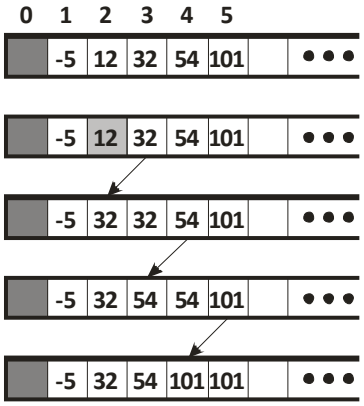

Figure 9. Remove key 12

```
void Remove(T key)
{    int i=Index(key);
     if (i==0) return;
     for (;i<count;i++)L[i]=L[i+1];
     count--;
}
```

Figure 10. Function *Remove*..

```
void Print(ostream& o,char* title)
{    o << endl << title << "{";
     for (int a=1;a<=count;a++) {
          if (a>1) o << ",";
          o << L[a];
     }
     o << "}" << endl;
}
```

Figure 11. Function *Print*.
title $\{i_1, i_2, \cdots, i_{count}\}$

```
void Scan(istream& i)
{    for (;;) {
          T key;
          i >> key;
          if (i.eof()) break;
          Insert(key);
     }
}
```

Figure 12. Function *Scan*.

```
void First(void){cursor=1;}
```

Figure 13. Function *First*.

```
void Next(void){if (cursor<count+1) cursor++;}
```

Figure 14. Function *Next*.

```
bool IsEol(void){return cursor>count;}
```

Figure 15. Function *IsEol*.

```
T Member(void){if (cursor) return L[cursor];}
```

Figure 16. Function *Member*.

```
bool IsMember(T key){return Index(key);}
```

Figure 17. Function *IsMember*.

```
int Index(T key)
{    int lo=1,hi=count;
     while (lo<=hi) {
          int m=(lo+hi)/2;
          if (key==L[m]) return m;
          if (key<L[m]) hi=m-1; else lo=m+1;
     }
     return 0;
}
```

Figure 18. Function *Index*.