```
#include <fstream>
#include <iomanip>
using namespace std;
template <class T>
class List{
    //-----------------------------------------------------------------------------------------
    //Define class Element containing a key and links to smaller and larger elements
    //-----------------------------------------------------------------------------------------
    struct Element {
        Element* smaller;
        T key;
        Element* larger;
        Element(Element* s,T k,Element* l):smaller(s),key(k),larger(l){}
        //-----------------------------------------------------------------------------------
        //Print an element to aid in debugging this class
        //-----------------------------------------------------------------------------------
        void Print(ostream& o)
        {   o << endl;
            o << " smaller=" << hex << smaller;
            o << " key=" << decimal << key;
            o << " larger=" << hex << larger;
        }
    };
    Element* largest;
    //-----------------------------------------------------------------------------------------
    //The cursor is used by iterator functions First(), Next(), IsEol(), IsMember()
    //and Member()
    //-----------------------------------------------------------------------------------------
    Element* cursor;
    //-----------------------------------------------------------------------------------------
    //Reclaim storage remaining on the list when the list goes out of scope
    //-----------------------------------------------------------------------------------------
    void Kill(Element* e)
    {   while (e->key!=MAX) {
            Element* p=e;
            e=e->larger;
            delete p;
        }
    }
```

Figure 2. template <class *T*> class *List* (element implementation).

```
    //-----------------------------------------------------------------------------------------------
    //Create a sentinel
    //-----------------------------------------------------------------------------------------------
    void Sentinel(void)
    {   Element* n=new Element(0,MAX,0);
        cursor=largest=n->smaller=n->larger=n;
    }
    //-----------------------------------------------------------------------------------------------
    //Create an empty list
    //-----------------------------------------------------------------------------------------------
    void List::Empty(void){Kill(largest->larger); delete largest; Sentinel();}
    //-----------------------------------------------------------------------------------------------
    //Print an element
    //-----------------------------------------------------------------------------------------------
    void Print(ostream& o,Element* e){e->Print(o);}
protected:
    const T MAX;
public:
    //-----------------------------------------------------------------------------------------------
    //Constructor – create an empty list
    //-----------------------------------------------------------------------------------------------
    List(T m):MAX(m){Sentinel();}
    //-----------------------------------------------------------------------------------------------
    //Constructor – scan items in stream i into the list
    //-----------------------------------------------------------------------------------------------
    List(T m,istream& i):MAX(m){Sentinel();Scan(i);}
    //-----------------------------------------------------------------------------------------------
    //Reclaim storage when the list goes out of scope
    //-----------------------------------------------------------------------------------------------
    ~List(){Kill(largest->larger);delete largest;Sentinel();}
    void Insert(T k)        //Insert key k
    {   Element* e=largest->larger;
        while (k>e->key) e=e->larger;
        if (k==e->key) return;
        Element* n=new Element(e->smaller,k,e);
        e->smaller->larger=n;
        e->smaller=n;
    }
```

Figure 2. template <class *T*> class *List* (element implementation) (continued).

```
void Remove(T k)    //Remove key k
{   Element* e=largest->larger;
    while (k>e->key) e=e->larger;
    if (k!e->key) return;
    e->smaller->larger=e->larger;
    e->larger->smaller=e->smaller;
    delete e;
}

void Print(ostream& o,char* title)
{   o << endl;
    o << title << "{";
    Element* e=largest->larger;
    for (int a=0;e->key!=MAX;a++){
        if (a>0) o << ",";
        o << e->key;
        e=e->larger;
    }
    o << "}" << endl;
}
void Scan(istream& i)
{   for (;;){
        T k;
        i >> k;
        if (i.eof()) break;
        Insert(k);
    }
}
```

Figure 2. template <**class** *T*> **class** *List* **(element implementation) (continued).**

```
    //-----------------------------------------------------------------------------------------------------
    //Position the cursor on the first element of the list
    //-----------------------------------------------------------------------------------------------------
    void First(void){cursor=largest->larger;}
    //-----------------------------------------------------------------------------------------------------
    //Move the cursor to the next element on the list or remain at the end of the
    //list.
    //-----------------------------------------------------------------------------------------------------
    void Next(void){if (cursor!=largest) cursor=cursor->larger;}
    //-----------------------------------------------------------------------------------------------------
    //Determine if the cursor is at the end of the list
    //-----------------------------------------------------------------------------------------------------
    bool IsEol(void){return cursor==largest;}
    T Key(void){if (cursor) return cursor->key;}
    //-----------------------------------------------------------------------------------------------------
    //Determine if input parameter k can be found on the list
    //-----------------------------------------------------------------------------------------------------
    bool IsMember(T k)
    {   Element* e=largest->larger;
        while (k > e->key) e=e->larger;
        return k==e->key;
    }
    //-----------------------------------------------------------------------------------------------------
    //Return the value in the list referenced by the cursor
    //-----------------------------------------------------------------------------------------------------
    T Member(void){if (cursor) return cursor->key;}
};
```

Figure 2. template <class *T*> class *List* (element implementation) (continued).

```
template <class T>
class Set: public List<T> {
public:
    Set(T m):List<T>(m) {}
    Set(T m,istream& i):List<T>(m,i) {}
    void Union(Set& s1,Set& s2)                    //s1 ∪ s2
    {   List<T>::Empty();
        for (s1.First();!s1.IsEol();s1.Next()) Insert(s1.Key());
        for (s2.First();!s2.IsEol();s2.Next()) Insert(s2.Key());
    }
    void Intersection(Set& s1,Set& s2)             //s1 ∩ s2
    {   List<T>::Empty();
        for (s1.First();!s1.IsEol();s1.Next()) {
            if (s2.IsAMember(s1.Key())) Insert(s1.Key());
        }
    }
    void Difference(Set& M,Set& S)                 // M − S = {m | m ∈ M and m ∉ S}
    {   List<T>::Empty();
        for (M.First();!M.IsEol();M.Next()) Insert(M.Key());
        Set<T> I(List<T>::MAX); I.Intersection(M,S);
        for (I.First();!I.IsEol();I.Next()) Remove(I.Key());
    }
};
```

Figure 2. template <class *T*> class *Set* (element implementation)