```
template <class T>
class List {
    struct Element {            //An element has
        Element* smaller;       //A pointer to the element having the next smaller key
        T key;                  //The key itself
        Element* larger;        //A pointer to the element having the next larger key
        Element(Element* s,T k,Element *l):smaller(s),key(k),larger(l){}
    };
    Element* largest;           //Points to the sentinel
    Element* cursor;            //Points to the current element
    void Kill(Element* e)
    {   while (e->key!=MAX) {
            Element* p=e;
            e=e->larger;
            delete p;
        }
    }
    void Sentinel(void)
    {   Element* n=new Element(0,MAX,0);
        cursor=largest=n->smaller=n->larger=n;
    }

protected:
    const T MAX;                //Maximum value in type T used as a sentinel
public:
    //---------------------------------------------------------------------------------------------
    //Constructor – create an empty list having only the sentinel
    //---------------------------------------------------------------------------------------------
    List(T m):MAX(m){Sentinel();}
    //---------------------------------------------------------------------------------------------
    //Constructor – Read input file stream i into the list.
    //---------------------------------------------------------------------------------------------
    List(T m,istream& i):MAX(m){Sentinel();Scan(i);}
    //---------------------------------------------------------------------------------------------
    //Destructor
    //---------------------------------------------------------------------------------------------
    ~List(){Kill(largest->larger); delete largest;}
    //---------------------------------------------------------------------------------------------
    //Create an empty listg
    //---------------------------------------------------------------------------------------------
    void Empty(void){Kill(largest->larger); delete largest; Sentinel();}
```

Figure 1. class *Set* (element implementation)

```
void Insert(T k)          //Insert a key
{   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;
}
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;
}                         //Print the list
void Print(ostream& o,const char* title)
{   o << endl << 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;
}                         //Scan items from input stream i into the list
void Scan(istream& i)
{   for (;;) {
        T k;
        i >> k;
        if (i.eof()) break;
        Insert(k);
    }
}
```

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

```
#ifndef List_h
#define List_h 1
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;
//-----------------------------------------------------------------------------------------------------------
//Put the contents of figure 1 here
//-----------------------------------------------------------------------------------------------------------
#endif
```

Figure 2. File **List.h**.

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <limits>
using namespace std;
int main()
{    List <int> L(INT_MAX);
     for (int a=0;a<20;a++) if(a%2) L.Insert(a);
     L.Print(cout,"L=");
     for (int b=0;b<10;b++) L.Remove(b);
     L.Print(cout,"L=");
     return 0;
}
```

Figure 3. File **p02.cpp**.

Example: Program p02 exercises template class List (element implementation). Odd integers $i : 0 \le i \le 20$ are inserted into List L. List L is printed. Odd integers $i : 0 \le i \le 10$ are removed and the list is printed again.

Program p02 prints:

L={1,3,5,7,9,11,13,17,19}

L={1,3,5,7,9}

Program **p02** consists of two source files, **List.h** and **p02.cpp** as given above.