```
 1  //This is the implementation file set.cpp.
 2  //This is the implementation of the class
Set.

 3  #include
 4  #include "listtools.h"
 5  #include "set.h"
 6  using std::cout;
 7  using std::endl;
 8  using LinkedListSavitch::Node;
 9  using LinkedListSavitch::search;
10  using LinkedListSavitch::headInsert;

11  namespace SetSavitch
12  {

13     template<class T>
14     Set::~Set( )
15     {
16         Node *toDelete = head;
17         while (head != nullptr)
18         {
19           head = head->getLink( );
20           delete toDelete;
21           toDelete = head;
22         }
23     }

24     template<class T>
25     bool Set::contains(T target) const
26     {
```

```
27          Node* result = search(head, target);
28          if (result == nullptr)
29             return false;
30          else
31             return true;
32      }

33      void Set::output( )
34      {
35          Node *iterator = head;
36          while (iterator != nullptr)
37          {
38            cout << iterator->getData( ) << "
";
39            iterator = iterator->getLink( );
40          }
```

```
21          //Adds a new element to the set.

22          void output( );
23          //Outputs the set to the console.

24          Set* setUnion(const Set& otherSet);
25          //Union calling object's set with
otherSet
26          //and  return a pointer to the new
set.

27          Set* setIntersection(const Set&
otherSet);
28          //Intersect calling object's set with
otherSet
29          //and return a pointer to the new
set.
30       private:
31            Node *head;
32      };    //Set
33   }     //SetSavitch
34   #endif //SET_H
```

```
 1  //This is the header file set.h. This is
the interface
 2  //for the class Set, which is a class for a
generic set.
 3  #ifndef SET_H
 4  #define SET_H

 5  #include "listtools.h"
The library "listtools.h" is the linked list
 6  using LinkedListSavitch::Node;
library interface from Display 17.14

 7  namespace SetSavitch
 8  {
 9    template <class T>
10    class Set
11    {
12     public:
13        Set( ) { head = nullptr; }
//Initialize empty set.

14        //Normally a copy constructor and
overloaded assignment
15        //operator would be included. They
have been omitted
16        //to save space.

17        virtual ~Set( ); //Destructor
destroys set.
18        bool contains(T target) const;
19        //Returns true if target is in the
set, false otherwise.

20        void add(T item);
```

```
41          cout << endl;
42      }

43      template<class T>
44      void Set::add(T item)
45      {
46          if (search(head, item)==nullptr)
47          {
48            //Only add the target if it's not
in the list
49            headInsert(head, item);
50          }
51      }

52      template<class T>
53      Set* Set::setUnion(const Set& otherSet)
54      {
55          Set *unionSet = new Set( );
56          Node* iterator = head;
57          while (iterator != nullptr)
58          {
59              unionSet->add(iterator->getData(
));
60              iterator = iterator->getLink( );
61          }
62          iterator = otherSet.head;
63          while (iterator != nullptr)
64          {
65              unionSet->add(iterator->getData(
));
```

```
66              iterator = iterator->getLink( );
67          }
68          return unionSet;
69      }

70      template<class T>
71      Set* Set::setIntersection(const Set&
otherSet)
72      {
73          Set *interSet = new Set( );
74          Node* iterator = head;
75          while (iterator != nullptr)
76          {
77              if
(otherSet.contains(iterator->getData( )))
78              {
79               interSet->add(iterator->getData(
));
80              }
81              iterator = iterator->getLink( );
82          }
83          return interSet;
84      }
85  }  //SetSavitch
```