```
// FILE: IntSet.h - header file for IntSet class
// CLASS PROVIDED: IntSet (a container class for a set of
//                 int values)
//
// CONSTANT
//   static const int MAX_SIZE = ____
//     IntSet::MAX_SIZE is the highest # of elements an IntSet
//     can accommodate.
//
// CONSTRUCTOR
//   IntSet()
//     Pre:  (none)
//     Post: The invoking IntSet is initialized to an empty
//           IntSet (i.e., one containing no relevant elements).
//
// CONSTANT MEMBER FUNCTIONS (ACCESSORS)
//   int size() const
//     Pre:  (none)
//     Post: Number of elements in the invoking IntSet is returned.
//   bool isEmpty() const
//     Pre:  (none)
//     Post: True is returned if the invoking IntSet has no relevant
//           relevant elements, otherwise false is returned.
//   bool contains(int anInt) const
//     Pre:  (none)
//     Post: true is returned if the invoking IntSet has anInt as an
//           element, otherwise false is returned.
//   bool isSubsetOf(const IntSet& otherIntSet) const
//     Pre:  (none)
//     Post: True is returned if all elements of the invoking IntSet
//           are also elements of otherIntSet, otherwise false is
//           returned.
//           By definition, true is returned if the invoking IntSet
//           is empty (i.e., an empty IntSet is always isSubsetOf
//           another IntSet, even if the other IntSet is also empty).
//   void DumpData(std::ostream& out) const
//     Pre:  (none)
//     Post: Contents of the invoking IntSet have been inserted into
//           out with 2 spaces separating one item from another if
//           if there are 2 or more items.
//   IntSet unionWith(const IntSet& otherIntSet) const
//     Pre:  size() + (otherIntSet.subtract(*this)).size() <= MAX_SIZE
//     Post: An IntSet representing the union of the invoking IntSet
//           and otherIntSet is returned.
//     Note: Equivalently (see postcondition of add), the IntSet
//           returned is one that initially is an exact copy of the
//           invoking IntSet but subsequently has all elements of
//           otherIntSet added.
//   IntSet intersect(const IntSet& otherIntSet) const
//     Pre:  (none)
//     Post: An IntSet representing the intersection of the invoking
```

```
//           IntSet and otherIntSet is returned.
//     Note: Equivalently (see postcondition of remove), the IntSet
//           returned is one that initially is an exact copy of the
//           invoking IntSet but subsequently has all of its elements
//           that are not also elements of otherIntSet removed.
//   IntSet subtract(const IntSet& otherIntSet) const
//     Pre:  (none)
//     Post: An IntSet representing the difference between the invoking
//           IntSet and otherIntSet is returned.
//     Note: Equivalently (see postcondition of remove), the IntSet
//           returned is one that initially is an exact copy of the
//           invoking IntSet but subsequently has all elements of
//           otherIntSet removed.
//
// MODIFICATION MEMBER FUNCTIONS (MUTATORS)
//   void reset()
//     Pre:  (none)
//     Post: The invoking IntSet is reset to become an empty IntSet
//           (i.e., one containing no relevant elements).
//   bool add(int anInt)
//     Pre:  contains(anInt) ? size() <= MAX_SIZE : size() < MAX_SIZE
//     Post: If contains(anInt) returns false, anInt has been
//           added to the invoking IntSet as a new element and
//           true is returned, otherwise the invoking IntSet is
//           unchanged and false is returned.
//   bool remove(int anInt)
//     Pre:  (none)
//     Post: If contains(anInt) returns true, anInt has been
//           removed from the invoking IntSet and true is
//           returned, otherwise the invoking IntSet is unchanged
//           and false is returned.
//
// NON-MEMBER FUNCTIONS
//   bool equal(const IntSet& is1, const IntSet& is2)
//     Pre:  (none)
//     Post: True is returned if is1 and is2 have the same elements,
//           otherwise false is returned; for e.g.: {1,2,3}, {1,3,2},
//           {2,1,3}, {2,3,1}, {3,1,2}, and {3,2,1} are all equal.
//     Note: By definition, two empty IntSet's are equal.

#ifndef INT_SET_H
#define INT_SET_H

#include <iostream>

class IntSet
{
public:
   static const int MAX_SIZE = 10;
   IntSet();
   int size() const;
```

```
   bool isEmpty() const;
   bool contains(int anInt) const;
   bool isSubsetOf(const IntSet& otherIntSet) const;
   void DumpData(std::ostream& out) const;
   IntSet unionWith(const IntSet& otherIntSet) const;
   IntSet intersect(const IntSet& otherIntSet) const;
   IntSet subtract(const IntSet& otherIntSet) const;
   void reset();
   bool add(int anInt);
   bool remove(int anInt);

private:
   int data[MAX_SIZE];
   int used;
};

bool equal(const IntSet& is1, const IntSet& is2);

#endif
```