```
#include "IntSet.h"
#include <iostream>
#include <cassert>
using namespace std;

IntSet::IntSet() : used(0) {}

int IntSet::size() const {
   return used;
}

bool IntSet::isEmpty() const {
   return used==0;
}

bool IntSet::contains(int anInt) const {
   bool found = false;
   for (int i=0; i<used; i++) {
      if (data[i]==anInt){
         found = true;
         break;
      }
   }
   return found;
}

bool IntSet::isSubsetOf(const IntSet& otherIntSet) const {
   bool success = true;
   for (int i=0; i<used;i++) {
      if (!otherIntSet.contains(data[i])) {
         success=false;
```

```
            break;
        }
    }
    return success;
}

void IntSet::DumpData(ostream& out) const {
    if (used > 0) {
        out << data[0];
        for (int i = 1; i < used; ++i)
            out << "  " << data[i];
    }
}

IntSet IntSet::unionWith(const IntSet& otherIntSet) const {
    IntSet temp;
    if (size() + (otherIntSet.subtract(*this)).size() <= MAX_SIZE) {
        for (int i=0;i<used;i++){
            temp.add(data[i]);
        }
        for (int i=0;i<otherIntSet.size();i++){
            temp.add(otherIntSet.data[i]);
        }
    }
    return temp;
}

IntSet IntSet::intersect(const IntSet& otherIntSet) const {
    IntSet temp;
    for (int i=0;i<used;i++) {
        if (otherIntSet.contains(data[i])) temp.add(data[i]);
```

```
    }
    return temp;
}


IntSet IntSet::subtract(const IntSet& otherIntSet) const {
    IntSet temp;
    for (int i=0;i<used;i++) {
        if (!otherIntSet.contains(data[i])) temp.add(data[i]);
    }
    return temp;
}

void IntSet::reset() {
    while (!isEmpty()) {
        used--;
        data[used] = 0;
    }
}

bool IntSet::add(int anInt) {
    bool success = false;
    if (used != MAX_SIZE && !contains(anInt)) {
        data[used] = anInt;
        used++;
        success = contains(anInt);
    }
    return success;
}

bool IntSet::remove(int anInt) {
```

```
    bool success = false;
    int location = -1;
     for (int i=0;i<MAX_SIZE;i++) {
         if (anInt == data[i]) {
             location = i;
             break;
         }
     }
     if (location>-1) {
         for (int i=location;i<used-1;i++){
             data[i] = data[i+1];
         }
         data[used] = 0;
         used--;
         success = !contains(anInt);
     }
     return success;
}

bool equal(const IntSet& is1, const IntSet& is2) {
     bool success = true;
     if (is1.size() == is2.size()) {
         if(!is1.isSubsetOf(is2)) success = false;
     } else {
        success = false;
     }
     return success;
}
```