Activity #1: IntList – Big 4

Instructions:

You are given the header file of the `IntList` class. You will need to complete the implementation of all public functionality in the `IntList.cpp` file. Test all the `IntList` functionality in main.

IntList.h

```
 1 // Specification file for the IntList class.
 2 #ifndef INTLIST_H
 3 #define INTLIST_H
 4
 5 class IntList
 6 {
 7 public:
 8    IntList(int);                // Constructor
 9    ~IntList();                  // Destructor
10    IntList(const IntList &);    // Copy constructor
11    IntList& operator=(const IntList &);
12                                 // Overloaded assignment operator
13    void addElement(int);
14 private:
15    int *array;                  // Array
16    int capacity;                // Size
17    int numElements;             // Number of elements
18    void resize();               // Resize the array when full
19 };
20
21 #endif
```

1. Write the constructor. (5 minutes)

```
IntList::IntList(int capacity)
{
      numElements = 0;
      this-> capacity = capacity;
      array = new int[capacity];
      for (int i = 0; i < numElements; i++)
            array [i] = 0;
}
```

2. Write the destructor. (2 minutes)

```
IntList::~IntList()
{
      delete [] array;
}
```

3. Write the copy constructor. (5 minutes)

```
IntList::IntList(const IntList &obj)
{
      // assign numElements and capacity (from obj)
      numElements = obj.numElements;
      capacity = obj.capacity;

      // allocate memory based on capacity
      array = new int[capacity];

      // copy over elements (from obj)
      for (int i = 0; i < numElements; i++)
            array[i] = obj.array[i];

}
```

4. Write the overloaded assignment operator. (5 minutes)

```
IntList& IntList::operator=(const IntList &obj)
{
      if (this != &obj)
      {
            // delete array
            delete [] array;

            // assign numElements and capacity (from obj)
            numElements = obj.numElements;
            capacity = obj.capacity;

            // allocate memory based on capacity
            array = new int[capacity];

            // copy over elements (from obj)
            for (int i = 0; i < numElements; i++)
                  array[i] = obj.array[i];
      }
      return *this;

}
```

5. Write the addElement method. (5 minutes)

```
void IntList::addElement(int el)
{
      // if the array is full, recapacity
      if (numElements >= capacity)
            recapacity();

      // the new element will be added whether
      // recapacity() is called or not!
      array[numElements] = el;
      numElements++;

}
```

6. Write the resize method. (5 minutes)

```
void IntList::resize()
{
      // update capacity
      capacity *= 2;

      // create new array based on updated capacity
      int * tempArr = new int[capacity];

      // copy old array values to new array
      for (i = 0; i < numElements; i++)
            tempArr[i] = array[i];

      // delete old array
      delete [] array;

      // reassign old array to new array
      array = tempArr;
}
```

6. Test all `IntList` functionality in main. (5 minutes)

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

void printIndex(int, int);

int main()
{
   const int SIZE = 20;
   IntList numbers(SIZE);

   // populate array
   for (int x = 0; x < SIZE; x++)
      numbers.addElement(x);

   cout << endl;

   int val = 5;
   int index = numbers.findElement(5);
   printIndex(val, index);
   index = numbers.findElement(SIZE);
   printIndex(SIZE, index);

   IntList numbers2(numbers);       // calls the copy constructor
   IntList numbers3(SIZE);
   numbers3 = numbers;              // calls the overloaded = operator

   return 0;
}

void printIndex(int val, int index)
{
      if (index == -1)
            cout << val << " not found!" << endl;
      else
            cout << val << " found at index " << index << endl;
}
```

Activity #2: FeetInches – Overloaded Operators

Instructions:

You are given the header file of the `FeetInches` class[1]. You will need to complete the missing implementation in the FeetInches.cpp file and provide sample tests on the side of each overloaded function. Do not focus on documentation – though some assumptions may be helpful to comment. The constructor, mutators, accessors, and simplify function have already been defined.

The `simplify()` function checks for values in the inches member greater than twelve or less than zero. If such a value is found, the numbers in feet and inches are adjusted to conform to a standard feet & inches expression. For example, 3 feet 14 inches would be adjusted to 4 feet 2 inches and 5 feet -2 inches would be adjusted to 4 feet 10 inches.

FeetInches.h

```
 1 // Specification file for the FeetInches class
 2 #ifndef FEETINCHES_H
 3 #define FEETINCHES_H
 4
 5 // The FeetInches class holds distances or measurements
 6 // expressed in feet and inches.
 7 class FeetInches
 8 {
 9 public:
10    // Constructor
11     FeetInches(int f = 0, int i = 0);
12
13    // Mutator functions
14    void setFeet(int f);
15
16    void setInches(int i);
17
18    // Accessor functions
19    int getFeet() const;
20
21    int getInches() const;
22
23    // Overloaded operator functions
24    FeetInches operator + (const FeetInches &); // Overloaded +
25    FeetInches operator - (const FeetInches &); // Overloaded -
26    FeetInches operator ++ ();      // Prefix ++
27    FeetInches operator ++ (int);   // Postfix ++
28    bool operator > (const FeetInches &);   // Overloaded >
29    bool operator < (const FeetInches &);   // Overloaded <
30    bool operator == (const FeetInches &);  // Overloaded ==
31 private:
32    int feet;        // To hold a number of feet
33    int inches;      // To hold a number of inches
45    void simplify(); // Defined in FeetInches.cpp
35 };
36
37 #endif
```

[1] Gaddis, p. 838-852

FeetInches.cpp (partial)

```
// Constructor
FeetInches::FeetInches(int f, int i) {
    feet = f;
    inches = i;
    simplify();
}

// Mutator functions
void FeetInches::setFeet(int f) {
    feet = f;
}

void FeetInches::setInches(int i) {
    inches = i;
    simplify();
}

// Accessor functions
int FeetInches::getFeet() const {
    return feet;
}

int FeetInches::getInches() const {
    return inches;
}

// The simplify function
void FeetInches::simplify() {
    if (inches >= 12) {
         feet += (inches / 12);
         inches = inches % 12;
    } else if (inches < 0) {
        feet -= ((abs(inches) / 12) + 1);
        inches = 12 - (abs(inches) % 12);
    }
}
```

1. Write the overloaded binary + operator. (5 minutes)

```
// Overloaded binary + operator.
FeetInches FeetInches::operator + (const FeetInches &right)
{
  FeetInches temp(inches + right.inches, feet + right.feet);
  return temp;

}
```

2. Write the overloaded binary - operator. (5 minutes)

```
// Overloaded binary - operator.
FeetInches FeetInches::operator - (const FeetInches &right)
{
  FeetInches temp(inches - right.inches, feet - right.feet);
  return temp;

}
```

3. Write the overloaded prefix ++ operator. (5 minutes)

```
// Overloaded prefix ++ operator. Causes the inches member to be incremented.
// Returns the incremented object.
FeetInches FeetInches::operator ++ ()
{
  ++inches;
  simplify();
  return *this;

}
```

4. Write the overloaded postfix ++ operator. (5 minutes)

```
// Overloaded postfix ++ operator. Causes the inches member to be
// incremented. Returns the value of the object before the increment
FeetInches FeetInches::operator ++ (int)
{
  FeetInches temp(feet, inches);
  inches++;
  simplify();
  return temp;

}
```

5. Write the overloaded > operator. (2 minutes)

```
// Overloaded > operator. Returns true if the current object is set to a
// value greater than that of right.
bool FeetInches::operator > (const FeetInches &right)
{
  return (feet > right.feet ||
          (feet == right.feet && inches > right.inches))
          ? true : false;

}
```

6. Write the overloaded < operator. (2 minutes)

```
// Overloaded < operator. Returns true if the current object is set to a
// value less than that of right.
bool FeetInches::operator < (const FeetInches &right)
{
   return (feet < right.feet ||
                (feet == right.feet && inches < right.inches))
                ? true : false;

}
```

7. Write the overloaded == operator. (2 minutes)

```
// Overloaded == operator. Returns true if the current object is set to a
// value equal to that of right.
bool FeetInches::operator == (const FeetInches &right)
{
  return (feet == right.feet && inches == right.inches)
                ? true : false;

}
```