Lab 03: More on Classes

Section 1: Guess program outputs.

1.

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

class HTMLTable
{
private:
   vector<string> headers;
   vector<vector<string> > rows;
   void writeRow(ostream& out, string tag, vector<string> row);
public:
   void setHeaders(const vector<string>& headers)
   {
      this->headers = headers;
   }
   void addRow(const vector<string>& row)
   {
      rows.push_back(row);
   }
   friend
   ostream& operator<<(ostream& out, HTMLTable htmlTable);
};


void HTMLTable::writeRow(ostream& out, string tag,
                                       vector<string> row)
{
   out << "<tr>\n";
   for (unsigned int k = 0; k < headers.size(); k++)
   {
      out << "<" << tag << "> "
         << row[k] << " </" << tag << "> ";
   }
   out << "\n</tr>\n";
}

ostream& operator<<(ostream& out, HTMLTable htmlTable)
{
   out << "<table border = \"1\">\n";

   htmlTable.writeRow(out, "th", htmlTable.headers);

   for (unsigned int r = 0; r < htmlTable.rows.size(); r++)
   {
      htmlTable.writeRow(out, "td", htmlTable.rows[r]);
   }
   out << "</table>\n";
   return out;
}

int main()
{
   vector<string> headers { "Name", "Address", "Phone" };
```

```
   vector<string> person1
          { "Mike Sane", "1215 Mills St", "630-728-1293" };
   vector<string> person2
   { "Natasha Upenski", "513 Briarcliff Ln", "412-672-1004" };

   HTMLTable hTable;
   hTable.setHeaders(headers);
   hTable.addRow(person1);
   hTable.addRow(person2);

   ofstream outFile("c:\\temp\\table.html");
   outFile << hTable;
   outFile.close();

   cout << hTable;

   system("c:\\temp\\table.html");

   return 0;
}
```

2.

```
#include <iostream>
#include <iomanip>
using namespace std;

class NumberArray
{
private:
  double *aPtr;
  int arraySize;
public:
  NumberArray(const NumberArray &);
  NumberArray(int size, double value);
  ~NumberArray() { if (arraySize > 0) delete [] aPtr; }
  void print() const;
  void setValue(double value);
};

NumberArray::NumberArray(const NumberArray &obj)
{
  arraySize = obj.arraySize;
  aPtr = new double[arraySize];
  for(int index = 0; index < arraySize; index++)
    aPtr[index] = obj.aPtr[index];
}

NumberArray::NumberArray(int size, double value)
{
  arraySize = size;
  aPtr = new double[arraySize];
  setValue(value);
}

void NumberArray::setValue(double value)
{
  for(int index = 0; index < arraySize; index++)
     aPtr[index] = value;
}
```

```
void NumberArray::print() const
{
  for(int index = 0; index < arraySize; index++)
    cout << aPtr[index] << "  ";
}

int main()
{
  NumberArray first(3, 10.5);

  NumberArray second = first;

  cout << setprecision(2) << fixed << showpoint;
  cout << "Value stored in first object is ";
  first.print();
  cout << "\nValue stored in second object is ";
  second.print();
  cout <<  "\nOnly the value in second object will "
       <<  "be changed.\n";

  second.setValue(20.5);

  cout << "Value stored in first object is ";
  first.print();
  cout << endl << "Value stored in second object is ";
  second.print();

  return 0;
}
```

Section 2: Review Questions and Exercises

1. The class Stuff has both a copy constructor and an overloaded = operator. Assume that blob and clump are both instances of the Stuff class. For each of the statements, indicate whether the copy constructor or the overloaded = operator will be called.
Stuff blob = clump;
clump = blob;
showValues(blob);

2. Describe the difference between making a class a member of another class (object composition) and making a class a friend of another class.

3. Explain why a class's copy constructor is called when an object of that class is passed by value into a function.

4. Assume a class named Bird exists. Write the header for a member function that overloads the = operator for that class.

5. Assume a class named Dollar exists. Write the headers for member functions that overload the prefix and postfix ++ operators for that class.

Section 3: Programming Challenges

1. Check Writing
Design a class Numbers that can be used to translate whole dollar amounts in the range 0 through 9999 into an English description of the number. For example, the number 713 would be translated into the string seven hundred thirteen, and 8203 would be translated into eight thousand two hundred three.
The class should have a single integer member variable
int number;
and a collection of static string data members that specify how to translate key dollar amounts into the desired format. For example, you might use static strings such as
string lessThan20[ ] =
{ "zero", "one", ..., "eighteen", "nineteen" };
string hundred= "hundred";
string thousand= "thousand";
The class should have a constructor that accepts a non-negative integer and uses it to initialize the Numbers object. It should have a member function print() that prints the English description of the Numbers object. Demonstrate the class by writing a main program that asks the user to enter a number in the proper range and then prints out its English description.

2. Day of the Year
Assuming that a year has 365 days, write a class named DayOfYear that takes an integer representing a day of the year and translates it to a string consisting of the month followed by day of the month. For example,
Day 2 would be January 2.
Day 32 would be February 1.
Day 365 would be December 31.
The constructor for the class should take as parameter an integer representing the day of the year, and the class should have a member function print() that prints the day in the month-day format. The class should have an integer member variable to represent the day and should have static member variables of type string to assist in the translation from the integer format to the month-day format.
Test your class by inputting various integers representing days and printing out their representation in the month-day format.