CISC 220 – 010 Homework #2 Due Friday, September 15, 2000

The following exercises will give you practice designing, building, testing and using a class. We will continue to use separate header files and source files for all classes we create. In this homework, you will focus on managing dynamic storage in a simple class – involving destructor, copy constructor and overloaded assignment.

Background The class we will create will be a (hopefully! ☺) better implementation of strings. Normally we use pointers to chars (or arrays of chars), and basic functions in `<string.h>` – available in both `C` and `C++`. Then we use functions like `strcmp` and `strcpy` from `<string.h>`. But there a lot of problems and minor aggravations with using this built-in string implementation. Comparing and copying isn't as convenient as it is for integers (where we use == and =); and there a number of problem having to do with storage and error-handling.

If we call the class `String`, then the following code illustrates a little of what we would like to do – a good way to get ideas for *your* classes is to make some simple application to picture how your class will be used. Study this code to see what we are trying to do.

```
#include "String.h"
#include <iostream.h>
int main(){
   String X("Hi, I am a nifty string.");
   String Y("And I am a different one.");
   String Z = X;

   cout << X << endl << Y << endl << Z << endl;

   if( X < Y ) cout << X << endl;
      else cout << Y << endl;
   const char* p = "A C-style string";
   String A(p);

   X = p;

   String W(X);  // or String W = X;

   int k = X.Length(); // should be same as strlen

   X = "Shorty";

   String B;  //  null string;
   B = X;
}
```

Part B) Now here is a header file containing the declarations for the String class.

```
#include <string.h>
#ifndef String_H
#define String_H

class String {
   private:
      unsigned int BufferLen;
      char* Buffer;
      void MakeBuffer(int nchars);
   public:
      String( const char* = NULL );
      String( const String& );
      ~String();
      const String& operator=( const String& );
      int Length() const{ return strlen(Buffer);};
      friend ostream& operator<<( ostream&, const String& );
      friend istream& operator>>( istream&, String& );
      friend bool operator<( const String&, const String& );
      // do the same for >, ==, and !=
      operator const char*()const { return Buffer; };
};

#endif
```

There are some rather strange things here, so let's go through this header file carefully. First, `Buffer` points to the storage containing the actual string – and we will continue to mark the string with a null. `BufferLen` is the current size of the buffer (not necessarily the same as the length of the string!), and `MakeBuffer` is a (private) utility function used by other member functions when the `String` buffer needs to be created (or re-created!).

Next are the constructors and the destructor – remember, we have dynamic memory. The first constructor gives the compiler a way to make a `String` from a `string` – like having automatic type conversion! ☺ By the way, the last function in the class gives the compiler a way to do type conversion in the other direction, i.e., from `Sting` to `string`.

The operators <, >, ==, and != are implemented as friends so that these comparison operators work with a mixture of `String` and `string` operands. For example

```
String X("apple");

if(  X > "baker" )

if( "ann" < X )
```

Once you understand this class declaration, add the prototypes for the other comparison operators, and then write the implementation file for the class – called `String.cc`.

Test this file using the usual compile command: `CC –c String.cc` Once you have an error-free compile, test your code by compiling and running the above example driver code. When this works, make a script file (`HW-2-A.scr`) containing the three files and a compile and run.

Part C) Change the design of the Person class from homework 1, so that it uses a String for the name. Make the necessary small coding changes and re-compile the application: (call the new file `homework2.cc`)

Make an array of 50 of type Person
Read data into this array from file person.data (until eof)
Invoke the bubble sort
Print the resulting array, sorted in order, by idNumber.

Compile this application using the command

```
CC homework2.cc person.o utilities.o  String.o
```

Once this program works, make a script file (`HW-2-B.scr`) containing the source code for homework2.cc, person.h and person.cc , a compile, and a run.

What to hand in: Hand in the two script files generated above, stapled together as one set. (Don't forget to put your name on the front page!)