ComS 228 Exam 1

September 27, 2004

Name: _________________________________________________

University ID: _______________________

Section: _____________ (10 percent penalty if incorrect)

This is a one-hour, closed-book, closed-notes, closed-calculator exam. The exam consists of 9 pages (including this one) and 6 questions; make sure your exam has all the pages and questions before you begin. Read each question carefully and provide clear and concise answers. Use correct C++ syntax when asked to write code. Please write neatly; illegible answers will get an automatic zero. If you need more room, you may write on the backs of the pages, but clearly indicate you have done so. To ensure fairness, everyone will be given the same amount of time, so you must stop writing when time is called. If you do not stop writing when instructed, you will receive a 25% deduction in your score.

<table>
<thead>
<tr>
<th>Question</th>
<th>Possible Points</th>
<th>Your Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 1</td>
<td>12</td>
<td>_____</td>
</tr>
<tr>
<td>Question 2</td>
<td>12</td>
<td>_____</td>
</tr>
<tr>
<td>Question 3</td>
<td>25</td>
<td>_____</td>
</tr>
<tr>
<td>Question 4</td>
<td>12</td>
<td>_____</td>
</tr>
<tr>
<td>Question 5</td>
<td>9</td>
<td>_____</td>
</tr>
<tr>
<td>Question 6</td>
<td>30</td>
<td>_____</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td>_____</td>
</tr>
</tbody>
</table>
1. (12 points) On the lines below, give the output of the following code:

```c++
void foo(double* d);
double* bar(double*& d);

int main()
{
    double* d1, d2;
    *d1 = 4.2;
    d2 = 2.4;
    double* d3;
    d3 = &d2;
    cout << "Main Start:" << *d1 << ", " << d2 << "," << *d3 << endl;
    foo(d1);
    cout << "After Foo:" << *d1 << "," << d2 << "\"," << *d3 << endl;
    d1 = bar(d3);
    cout << "After Bar:" << *d1 << "," << d2 << "\"," << *d3 << endl;
}

void foo(double* d)
{
    *d = 1.1;
    d = new double;
    *d = 2.2;
    cout << "Inside Foo:" << *d << endl;
}

double* bar(double*& d)
{
    *d = 3.3;
    d = new double;
    *d = 5.5;
    cout << "Inside Bar:" << *d << endl;
    return d;
}
```

output line 1: ______________________________________

output line 2: ______________________________________

output line 3: ______________________________________

output line 4: ______________________________________

output line 5: ______________________________________
2. (12 points) Consider the following example class and answer the questions below.

```cpp
class MyExample
{
    public:
        void func1(MyExample& other);
        void func2(MyExample& other) const;
        void func3(const MyExample& other) const;
        void func4(const MyExample& other);
    private:
        int value;
};
```

Will the following implementations compile? If not, give a short explanation why.

a)```cpp
void MyExample::func1(MyExample& other)
{
    value = 2;
    other.func3(*this);
}
```

b)```cpp
void MyExample::func2(MyExample& other) const
{
    other.func1(other);
    func3(*this);
}
```

c)```cpp
void MyExample::func3(const MyExample& other) const
{
    other.func3(*this);
    value = 13;
}
```

d)```cpp
void MyExample::func4(const MyExample& other)
{
    func3(other);
    other.func1(*this);
}
```
3. (25 points) In this problem you will design a C++ class to represent an electronic stopwatch.

Today’s stopwatches do a lot more than just measuring the time from start to finish, this stopwatch has a number of different modes. Mode 1 shows the current wall-clock time for the stopwatch to double as a simple watch. The time is stored as an integer counting 1/100\textsuperscript{th}s of a second from 8 pm, Sep 27\textsuperscript{th} 2004. Mode 2 is the stopwatch mode, where it measures the time from the start to the end of a race. In addition to the main running time, it can keep times for up to 10 laps. For each lap it stores the time for the lap as well as the cumulative time from the start of the race to the end of the lap. For training purposes it is possible to store the single lap times and the whole race time in secondary buffer members. These buffers are used in mode 3, which is a countdown mode. In countdown mode it counts down the race time from the stored time, and it also counts down the times of the individual laps starting at the first one, to support a simple comparison whether the run is better than the stored one.

Times are stored as integers, counting in 1/100\textsuperscript{th} second steps. The system has a hardware timer that calls a parameterless member function of the StopWatch class every 1/100\textsuperscript{th} of a second, to update all the class’s times. The stopwatch has 3 buttons (left, middle, right), which control all the functions. Pressing/releasing these buttons causes one of three Button-functions to be called. Each of these three functions gets a bool parameter indicating whether the button is pressed (true) or released (false). To drive the actual display, which has one line with exactly 40 characters, the class needs to be able to produce a simple character array with the numbers to show. It is returned as a char * to an internal buffer that is big enough for a single line, and that is kept in the class.

On the next page, write the class header file for a class called StopWatch. Don’t forget to write the needed preprocessor directives. In the class definition, include member variables to store all the state information and member function prototypes to perform all the actions described above. In addition to the default constructor, write a constructor that takes the current wall-clock time as its argument, to initialize the clock when the stopwatch is turned on.

Note: you are writing the header file only in this problem. You do not need to write any function implementations. You also do not need to include comments or pre/post conditions - unless any of your variable, arguments, function names, or function return types would be unclear to the graders.
(Problem 3 continued)
4. (12 points) Please write down the computational complexity (big –O) of the following functions:

a) `int afunc(int n)`
   ```
   int m = 16;
   int p = n / 2;
   int result = 0;
   
   for(int i = 0; i < n; ++i)
       m *= 2;
   
   for(int a = 0; a < n; ++a)
       { 
       for (int b = 0; b < p; ++b)
       { 
          result += m * n * b * a;
       }
       }
   return result;
   }```

b) `int bfunc(int n)`
   ```
   int m = n * log(n);
   int p = n - 2;
   int result = 0;
   
   for(int a = 0; a < 15; ++a)
       { 
       for (int b = p; b < n; ++b)
       { 
          result += n - b + a;
       }
       }
   return result;
   }```

c) `int cfunc(int n)`
   ```
   int m = 1;
   int p = n - 2;
   int result = 0;
   
   for(int i = 0; i < n; ++i)
       m *= 2;
   
   for(int a = 0; a < m; ++a)
       { 
       for (int b = 0; b < 12; ++b)
       { 
          result += n + b - a;
       }
       }
   return result;
   }```
5. (9 points) Consider the following class:

```cpp
class Widget
{
public:
    Widget();  //Sets the two values to zero
    Widget(int x, int y);  //Sets the first value to x and the second to y
    int value();  //Returns the sum of the first and second values
    void print() const;  //Prints out the two values to cout
private:
    int firstVal, secondVal;
};
```

In this problem, you will write an overloaded `<` operator as a member function of class Widget. If the client programmer has created two Widget objects w1 and w2, then he wants to be able to compare them using an expression like w1 < w2. Widgets are compared by adding their two values together and comparing the result; i.e., w1 would be less than w2 if the sum of firstVal and secondVal from w1 is less than the sum of firstVal and secondVal from w2.

a) In the space below, write the prototype for the `<` operator. Also, write down where in the code you would put the prototype.

b) In the space below, write the implementation for the `<` operator. Also, write down where in the code you would put the implementation.
6. (30 points) A set is a data structure that holds a list of data items that does not contain duplicates. The order in which the items are listed doesn’t matter, so typically, the implementation of a set will keep them sorted in ascending order. Sets can grow and shrink in size, so one way to implement them is to store the items in a dynamic array. In this problem, you will be writing member functions for a class that represents a set of integers. You can find the class definition of IntSet on the last page of this exam, which you may tear off to refer to as you work on these problems.

a) In the space below, implement the destructor for IntSet. Don’t forget to include the function header.

b) In the space below, implement the assignment operator for IntSet. Don’t forget to include the function header.
c) The `unionWith()` function for IntSet takes another IntSet and adds all the items that appear in the other IntSet to the items in this IntSet. However, items will not be duplicated; if the item was already in this IntSet, then it will not be added again. For example, suppose IntSet $s$ contains 2,4,6,8 and IntSet $t$ contains 2,5,7,8,12. Then suppose the function $s.unionWith(t);$ is called. Afterwards, $s$ will contain 2,4,5,6,7,8,12.

In the space below, implement the `unionWith()` member function for IntSet. Don’t forget to include the function header. You may not use the `addInt()` function to help you write this one; instead, work with the dynamic arrays directly.
class IntSet
{
    public:
        IntSet();
        //Post: this IntSet is an empty set of integers
        IntSet(const IntSet& other);
        //Post: if the other IntSet is empty, then so is this one. Otherwise,
        //this IntSet is a deep copy of the other IntSet
    ~IntSet();
        //Post: if this IntSet is not empty, then the memory allocated to this
        //IntSet is reclaimed
        IntSet& operator=(const IntSet& other);
        //Post: if the other IntSet is an empty IntSet, then this IntSet becomes
        //an empty IntSet too. Otherwise, if this IntSet and the other IntSet are
        //not the same object, then this IntSet is a deep copy of the other
        //IntSet.
    bool isIn(int num) const;
        //Post: returns true if num is in this IntSet
    void addInt(int newNum);
        //Post: If newNum was not already in this IntSet, it has been added to
        //this IntSet
    void removeInt(int num);
        //Post: If num is the only integer in this IntSet, then this IntSet
        //becomes empty. Otherwise, num is removed from this IntSet.
    void unionWith(const IntSet& other);
        //Post: If this IntSet is empty, then this IntSet is a deep copy of
        //other. Otherwise, the integers in the other IntSet but which are not
        //already in this IntSet have all been added to this IntSet.

    private:
        int* theInts;  //The pointer to the dynamic array containing the list of
                        //integers that are in this IntSet. The integers are
                        //listed in increasing order. If this IntSet is empty,
                        //the pointer is NULL.
        int size;       //The number of integers in the dynamic array.
};