CS193D Midterm Examination

This is a closed-book, closed-note exam. Your exam should have 8 pages, and there are two questions totaling 40 points. You have as much time as you need (within reason). Unless the exam says otherwise, details of syntax are not important so long as it is clear what you mean. Comments are not required but may help you receive partial credit if your intentions are correct but your code is wrong. Unless otherwise noted, you are always free to write helper/support routines or embedded classes as needed. If you are running out of ideas, pseudo-code sketches may earn some partial credit.

Remote SITN students should take the exam in one sitting, sending back both a faxed version (immediately, to 650-723-6092), and the hardcopy, via your SITN courier.

Good luck! And take your time!

leland username: ______________________

Last Name: ______________________

First Name: ______________________

SUID: ______________________

I accept the letter and spirit of the honor code. I have not given nor received aid on this exam. I pledge to write more neatly than I ever have before in my entire life.

(signed) ____

		Score	Grader
1. Presidents/Angels	(20)	______	______
2. Cacti	(20)	______	______
Total	**(40)**	______	

Problem 1: Constructors and Destructors (20 points)

a.) For the following code, assume that STL's deep copying `string` object has already been included. Assume that the `candidates` function has just been called. Specify the ordering for all the calls to the `string` memory management routines: the default constructor, the (`char*`) constructor, the copy constructor, the `operator=` assignment operator, and the destructor. Include consideration of the parameter set-up for function calls. Assume that function and method arguments are evaluated from left to right.

```
static const string *vote(string party, string governor,
                          const string *vpp, const string& vp)
{
   string democrat(*vpp);
   party = democrat;
   governor[1] = governor[0];
   return &vp;
}

static void candidates()
{
   string gore("Al");
   string bush = gore;
   string& nader = bush;
   string buchanan;
   buchanan = gore;

   nader = *vote("Republican", bush, &gore, gore);
}
```

Answer (continue enumeration of calls after the one I've provided):

1. `char *` constructor for `gore`.
2.

b.) For the following code, assume that STL's deep copying `string` object has already been included. Assume that the `bosley` function has just been called. Once again, specify the ordering for all the calls to the `string` memory management routines: the default constructor, the (`char*`) constructor, the copy constructor, the `operator=` assignment operator, and the destructor. This time consider the manner in which superclass and subclass constructors and destructors are called, but only print memory management information as it pertains to `string` objects.

```
class Angel {

   public:
      Angel() : drew(NULL) {}
      Angel(const string& heroine) : lucy(heroine), drew(&heroine) {}

   protected:
      string lucy;
      const string *drew;
};

class CharliesAngel : public Angel {

   public:
      CharliesAngel() { cameron = "diaz"; }
      CharliesAngel(const CharliesAngel& charlie) :
         Angel(charlie) , cameron("diaz") {}

   protected:
      string cameron;
};

static void bosley()
{
   CharliesAngel alex;
   CharliesAngel dylan(alex);
}
```

Answer:

1.
2.

Problem 2: Inheritance and Cacti (20 points)

For this problem you will design classes for a cactus patch. It turns out there are three types of cacti: `Sad`, `Stoic`, and `Angry`. Each cactus contains some integer amount of water. The only two events which ever happen in the life of a cactus are intense sunshine and occasional rain. Cacti lead pretty simple lives, waiting eagerly for weather systems to pass.

- All cacti respond to the `water()` method. Whenever a cactus receives a `water()` message its amount of water goes up by `1` up to a maximum of `100`. The cactus is so excited that something happened that it prints `"Water!"` to standard output. Except the `Stoic` cactus which, after the `"Water!"`, also prints `" (well.. not that I care.)"`.

- All cacti also respond to the `sun()` method. Whenever a cactus receives a `sun()` message, it prints to standard output `"Sun!"`. The sun also causes the cactus' amount of water to go down by 1, but it cannot go below zero. This may cause the cactus to become unhappy. The happiness factor of a cactus is generally `( 2 * water ) - 5`, but with a maximum value of `20`. `Sad` cacti are the exception—they are always exactly `10` units less happy than a nomal cactus with a maximum happiness of `10`. A cactus is "unhappy" whenever its happiness factor is negative. When unhappy, and only when unhappy, a cactus responds by printing to standard output `"Not happy!"`. Except the `Angry` cactus, which instead responds by printing `"REALLY not happy!"`. The exertion of printing the extra word so taxes the `Angry` cactus that it loses an additional unit of water.

Design a class hierarchy to model the cacti. Draw a little tree of your hierarchy. List the instance variables and methods (including their types / prototypes) for each class. This drawing will serve as your `.h` so include all the necessary type information and method headers. You will not need to deal with initialization or constructors. You may assume that all of you instance variables have been correctly set before your code runs, so long as it's clear from your class drawing what the correct value for each is. We will assume that all members are modified as `protected` so you do not need to deal with that either. Draw your tree below:

Now provide the `.cc` code to implement the `water` and `sun` methods and any support methods. Use this and the rest of the exam to provide your answers. Maximize code consolidation by placing as much code and programming logic in your base class as possible.