Name: ______________________________

CSc 328, Spring 2004
Final Examination
May 12, 2004

READ THIS FIRST

Fill in your name above. Do not turn this page until you are told to begin.

Books, and photocopies of pages from books MAY NOT be used during the exam. Electronic devices may not be used during the exam. Everything else, such as the full set of handouts, is permitted.

If you run out of room, write on the back of a page. DO NOT use sheets of your own paper.

If you have a question you wish to ask, raise your hand and the instructor will come to you. DO NOT leave your seat.

If you are unsure about the form or operation of a language construct that is central to a problem's solution, you are strongly encouraged to ask about it.

If you have a question that can be safely resolved by making a minor assumption, simply write down that assumption and proceed. Example: "Assuming setw(n) causes the next value to appear in an n-wide field." If a possible assumption seems to make a problem trivial, ask it about it!

You need not include any explanation in an answer if you are confident it is correct. However, if an answer is incorrect, any accompanying explanation may help you earn credit, even on multiple choice and true/false questions.

If you're completely puzzled on a problem, please ask for a hint. Try to avoid leaving a problem completely blank—that will certainly earn no credit.

Your C++ code need not have any #includes or using directives—save time by leaving them out. Trivial errors in C++ code, such as an omitted semicolon, will not result in any deduction.

This is a one hour and forty-five minute (105 minutes) exam with a total of 100 points of regular questions and 15 possible points of extra credit.

When told to begin, double-check that your name is at the top of this page, and then **put your initials in the lower right hand corner of each page**, being sure to check that you have all 17 pages.

When you have completed the exam, enter your name on the exam sign-out log and then hand your exam to the instructor.

Problem 1: (1 point each, 10 points total)

For the following multiple choice questions, circle the best answer.

(1) In the statement cout << "hello!" << endl;", endl is a/an:

a) extractor
b) inserter
c) manipulator
d) terminator

(2) In the definition int Rectangle::getArea() { ... }', the '::' is called the

a) class binder
b) scope resolution operator
c) member resolver
d) method specifier

(3) Consider the following fragment of a Java class definition:

```
class Lamp {
  ...
      private Bulb itsBulb;
      }
```

Which of the following is the closest equivalent for itsBulb in C++? (assume all are in a private section):

a) Bulb   itsBulb;
b) Bulb   *itsBulb;
c) Bulb&   itsBulb;
d) Bulb&   *itsBulb;

(4) Consider this fragment of C++ code:

```
char c;
while (cin >> c) { ... }
```

What is that causes the while loop to be exited at end-of-file on standard input?

a) At end-of-file the character c will have the value 0.
b) The extraction fails and the failure propagates to the while.
c) At end-of-file the expression cin >> c evaluates to 0.
d) A break statement is executed in the body of the loop (not shown).

(5) Given no other knowledge of a class X, which of the following is the best indicator that X requires a copy constructor?

a) The class defines this assignment operator: X& operator=(const X&).
b) Memory allocation is performed in the constructor.
c) The class has a destructor.
d) There is a data member that is a pointer.

(6) Default arguments can be used to

a) Specify missing arguments for overloaded operators.
b) Reduce the number of overloaded functions.
c) Control the order of evaluation in a member initialization list.
d) All of the above

(7) For overloading, the prefix and postfix forms of the increment (++) and decrement operators (--) are distinguished by:

a) The postfix keyword.
b) Differing numbers of parameters.
c) Use of operator+++ and operator--- for the prefix forms.
d) There's no need to distinguish the prefix and postfix forms when overloading.

(8) Which of the following best describes the ordering of destructor calls for stack-resident objects in a routine:

a) The first object created is the first object destroyed; last created is last destroyed.
b) The first object destroyed is the last object created; last created is first destroyed.
c) Objects are destroyed in the order they appear in memory—the object with the lowest memory address is destroyed first.
d) The order is undefined and may vary from compiler to compiler.

(9) The statement char *p = new char(100); does which of the following?

a) Allocates sufficient memory in the heap to hold 100 char values.
b) Directs the memory allocator to allocate memory starting at address 100.
c) Allocates one byte of memory in the heap and initializes with the character 'd'.
d) The statement won't compile.

(10) If a, b, and c are of class type V, V + V is overloaded with a free-standing function, and V * V is overloaded with a member function, both returning a value of type V, then the expression a + b * c is equivalent to which of the following?

a) c.operator*(operator+(a,b))
b) a.operator+(b.operator*(c))
c) operator*(a.operator+(b),c)
d) operator+(a, b.operator*(c))

Problem 2: (1 point each, 5 points total)

Indicate whether each of the following statements is true (T) or false (F)

_____ In C++ there is no direct analog to this Java class definition: `abstract class A { }`

_____ The `string` class in the C++ standard library is essentially equivalent to `java.lang.String`.

_____ The semantics associated with the `bool` type in C++ are essentially the same as the `boolean` type in Java.

_____ A good first step for solving performance problems is to make more extensive use of inline functions.

_____ Multiple inheritance provides a good way to define a class that consists of a single instance of several other classes.

Problem 3: (2 points each, 10 points total)

a) The routine `copyints(const int *p1, int *p2, int N)` copies `N` `int`s starting at one address to another address. In which direction is the data copied: from `p1` to `p2` or `p2` to `p1`? Justify your answer.

b) If a data member is declared as `mutable`, what does that mean?

c) Define a class `X` that makes the following free standing function compilable:

```
X f() { return 1;}
```

d) What does the `const` in `void X::f() const { ... }` indicate?

e) The instructor often said that in C++, "you only pay for what you use". What did he mean by that?

Problem 4: (8 points)

In Java, a method f() that requires a parameter of type X and returns void is usually declared like this:

```
void f(X value)
```

In C++, there are several choices for that same situation. Here are four of them:

```
void f(X& value)
void f(const X& value)
void f(X* p)
void f(X value)
```

Briefly describe the meaning of each alternative AND a situation where that alternative would be the best choice.

Problem 5: (5 points)

The following paragraph is written using terms that a Java programmer might use. Strike out those Java-centric terms and write in the terms that a C++ programmer would most likely use.

A common example for teaching inheritance is a hierarchy of geometric shapes. A class named Shape *has subclasses like* Rectangle *and* Circle*. Methods such as* getArea() *and* getPerimeter() *provide good examples of abstract methods. To see if students are thinking, an instructor will sometimes propose fields in* Shape *such as* length*,* width*, and* radius*. Some examples include a class* Square*, with* Rectangle *as its superclass, but by the Liskov Substitution Principle that is clearly a bad idea.*

Problem 6: (5 points)

Which element of C++ did you find the most difficult to understand? What did you ultimately find to be the key to understanding that element?

Problem 7: (4 points)

What is meant by the term "type extensibility"? Name two elements of C++ that support type extensibility.

Problem 8: (5 points)

In the style of an e-mail message to a friend who knows C and Java, briefly describe one of the following features of C++: (1) destructors, and the guarantee of destruction, (2) operator overloading, (3) IO Streams, (4) templates, (5) multiple inheritance. **IMPORTANT: Your "message" should include some examples of code.**

Problem 9: (3 points)

Write a class named Circle such that this program

```
int main()
{
    Circle c(7);

    cout << c.radius() << endl;
    c.radius() = 11;
    cout << c.radius() << endl;
}
```

produces this output:

```
7
11
```

Problem 10: (5 points)

Write a program that prints the integers from 1 through 1000, one per line. **RESTRICTION: Your solution may not use any control structures (if, while, for, etc.), goto statements, local variables, or employ recursion.**

Problem 11: (10 points)

In this problem you are to create two overloaded operators that operate on C++ Standard Library strings:

(1) Overload string & string to cause the value of the right-hand operand to be appended to the left-hand operand, changing the left-hand operand. (This operation is imperative, not applicative.)

```
string s("520"), dash("-");

s & dash;                        // s is now '520-', dash is unchanged

s & "577" & "-" & "6431";    // s is now '520-577-6431'
```

The left hand operand is returned, providing the ability to "chain" several & operations as shown in the second case.

Another example:

```
string s2;
for (char *p = "testing"; *p; p++) {
    s2 & "-" & string(1,*p);    // string(1,*p) creates a one-character string from *p
    }
s2 & "-";

cout << s2 << endl; // Output: -t-e-s-t-i-n-g-
```

(2) Overload ~string to produce a copy of the string wrapped in angle-brackets. Example:

```
string s("testing");

cout << ~s << endl;                   // Output: <testing>; s is unchanged

cout << ~~(s + " this") << endl;      // Output: <<testing this>>
```

Note: There's no problem overloading tilde, even though that symbol is also used to denote a destructor.

The string class may not be changed, of course—your solution must use string "as is".

Recall that the string class overloads + to represent concatenation and has a string(const char *) constructor, just like the String class developed in the slides. (And that's just about all you need to recall about string to do this problem.)

[Space for answer for problem 11]

Problem 12: (10 points)

Create a templated class named Pair whose instances are an aggregate of two values of possibly different types. For example,

```
Pair<int, string> p1(10, "ten");
```

creates a Pair named p1 that holds the int 10 and the string "ten". Provide methods getA() and getB() that return the first and second values, respectively. Note that the return types of getA() and getB() depend on the types the template is instantiated with. In this example, the return type of getA() is int and the return type of getB() is string.

Be sure to provide an inserter for Pairs. Example:

```
cout << p1 << endl; // Output: <10,ten>
```

Pair should be able to accommodate any two types, call them A and B, as long as both A and B support copy construction and assignment. Do not assume the presence of a default constructor for A or B.

Another example (assuming p1 from above):

```
Pair<double, char> p2(3.1, '4');
Pair< Pair<int,string>, Pair<double,char> > p3(p1, p2);

cout << p3 << endl; // Output: <<10,ten>,<3.1,4>>
```

There is no interaction between A and B. One might declare Pair<Cat,Dog> pets<c,d>.

Problem 13: (15 points)

In this problem you are to define a three-class inheritance hierarchy, with the following requirements:

There are two types of Containers: Buckets and Boxes.

Every Container has an integer serial number.

To create a Bucket, its height and radius (both integers), and serial number must be specified.

To create a Box, its length, width, and height (integers), serial number, and material type ('m' for metal, 'c' for cardboard) must be specified.

A container may be queried for its serial number, volume, and whether it can hold a fluid. Metal boxes and all buckets can hold fluids; cardboard boxes cannot hold fluids.

Additionally, Container is to have a static member function totalFluidCapacity() that accepts a collection of Containers and calculates the total fluid capacity of all the containers that are capable of holding fluids.

Think of these requirements as coming from your manager; all the other decisions, such as data representation, are up to you. Note that one key problem is how to pass the collection of containers to totalFluidCapacity(). Hint: Here's a very bad way:

```
int totalFluidCapacity(Container [ ])  // Don't do this!
```

The volume of a bucket is $\pi * radius^2 * height$. Use M_PI for π.

[Additional space for answer for problem 13]

Problem 14: (5 points)

This problem explores your understanding of when objects are created and destroyed, and where they reside. At each of the points of execution that are marked with a series of dashes, enter the number of instances of X that exist at that moment. Note that separate totals are maintained for stack-resident and heap-resident objects. The first pair of entries has been completed as an example.

```
int main()
{
    X x1;

    - - - - - - - - - - -  Stack: __1__  Heap: __0__

    X *p = new X[3];

    - - - - - - - - - - -  Stack: _____  Heap: _____

    f(p);
```

NOTE: The following totals are AFTER `f(p)` returns!

```
    - - - - - - - - - - -  Stack: _____  Heap: _____
}

void f(X *arg)
{
    X p(*arg);

    - - - - - - - - - - -  Stack: _____  Heap: _____

    delete arg;
    X *xp = 0;
    delete xp;

    - - - - - - - - - - -  Stack: _____  Heap: _____

    X a[3];
    a[1] = a[2];

    - - - - - - - - - - -  Stack: _____  Heap: _____

    X **a2 = new X*[5];

    - - - - - - - - - - -  Stack: _____  Heap: _____
    (Totals from previous page:) Stack: _____  Heap: _____

    for (int i = 1; i <= 3; i++) {
        X x1;
        a2[i] = new X;
        }

    - - - - - - - - - - -  Stack: _____  Heap: _____

    delete a2[1];
    delete a2[2];

    - - - - - - - - - - -  Stack: _____  Heap: _____

    delete [] a2;

    - - - - - - - - - - -  Stack: _____  Heap: _____
```

(Now go back and do the last one in `main()`.)

```
}
```

Extra Credit Section (1 point each unless otherwise noted)

(1) What is "cfront"?

(2) Name a C++ reserved word that really isn't needed, and describe how a C++ programmer could work around its absence. (Ignore the problem of existing code that might use it.)

(3) A person telling a C++ programmer about Java says this: "Think of it this way: In Java, they are always explicit." What is the "they" that's being referred to?

(4) Offer a GOOD definition of the term "memory leak".

(5) What do the # and ## preprocessor operators do?

(6) Why did Stroustrup choose the terms "base class" and "derived class" instead of "superclass" and "subclass"?

(7) How many lectures were there?

(8) Answer any <u>one</u> of these three:

At what university did Stan Lippman earn a graduate degree in English?

Who had the first Original Thought of the semester?

Who asked for coffee and doughnuts for the final exam?

(9) (3 points) For problem 14, which asks you to calculate the number of instances existing at various points in time, the instructor wrote a program, docounts, that takes a C++ program as an input file and produces a listing essentially identical to that shown in problem 14 but with the counts filled in. The C++ code did contain an indication of each of the points where the counts were to be calculated.

Here is the task for you: Sketch out the design for a program like docounts. Assume that the reader is a skilled developer.

(10) (4 points) Consider the following header file, Rectangle.h:

```
#ifndef _Rectangle_h_
#define _Rectangle_h_

class Rectangle {
   public:
      Rectangle(double width, double height);
      double getArea();
      // ...
   private:
      double itsWidth, itsHeight;
};

inline double Rectangle::getArea()
{
   return itsWidth * itsHeight;
}

#endif
```

Imagining a large system in which the Rectangle class is widely used, what problem will result if the reserved word inline (underlined) on Rectangle::getArea() is removed?