Instructions:

- Print your name in the space provided below.
- Answer each question in the space provided.
- If you want partial credit, justify your answers briefly and concisely, even when justification is not explicitly required.
- There are 15 questions, priced as marked. The maximum score is 100.
- When you have completed the test, sign the pledge at the bottom of this page and turn in the test.
- This is a closed-book, closed-notes examination. No calculators or other electronic devices may be used during this examination.
- You may not discuss (in any form: written, verbal or electronic) the content of this examination with any student who has not taken it. You must return this test form when you complete the examination. Failure to adhere to any of these restrictions is an Honor Code violation.

Do not start the test until instructed to do so!

Name ____________________________

Pledge: On my honor, I have neither given nor received unauthorized aid on this examination.

_______________________________

signed
Consider the following class declaration, *(slightly modified from the spring 2001 first test)*:

```cpp
enum Color {RED, GREEN, BLUE, YELLOW};
enum Direction {UP, DOWN, LEFT, RIGHT};

class CPacMonster {
    private:
        CLocation mLoc;
        Color mColor;
        bool mActive;
    public:
        CPacMonster(); // Line 1
        CPacMonster(CLocation L, Color C); // Line 2
        CPacMonster& setColor(Color C = RED); // Line 3
        Color getColor() const; // Line 4
        CPacMonster Move(Direction Dir, int Distance = 1); // Line 5
        CLocation getLocation() const; // Line 6
        ~CPacMonster(); // Line 7
    }
}
```

Assume the following object declarations are in scope:

```cpp
CPacMonster Doggett, SmokingMan;
```

1. **[7 points]** For the given statement below very briefly explain the effect of the execution of the statement upon the data members of the object Doggett.

   ```cpp
   Doggett.Move(LEFT).setColor(YELLOW);
   ```

   **The call to Move() modifies Doggett to reflect a move to the left (probably updates the CLocation sub-object). Move() returns a copy of the object Doggett, and setColor() operates on that copy rather than on Doggett.**

2. **[7 points]** For the given statement below very briefly explain the effect of the execution of the statement upon the data members of the object SmokingMan.

   ```cpp
   SmokingMan.setColor(RED).Move(LEFT, 1);
   ```

   **The call to setColor() (probably) sets SmokingMan.mColor to RED. setColor() returns - SmokingMan by reference, so the call to Move() also operates on SmokingMan, probably altering the CLocation sub-object.**

3. **[4 points]** If the CLocation object in the CPacMonster class was declared:

   ```cpp
   private:
       CLocation* mLoc;
   ```

   Assuming that no other changes are made to the class interface, would the composition be static or dynamic?

   **Aside from the constructor, there's no way to change which CLocation object a CPacMonster is associated with. That's a static association.**
For the next 4 questions, consider the following classes:

```cpp
class BoxCar {
private:
    string Label;
    Crate** Cargo; //ptr to array of ptrs
    int Size;
    int numCars;
public:
    BoxCar(string L = "None",
            int Sz = 10);
    BoxCar(const BoxCar& RHS);
    bool addCrate(Crate*& newCrate);
    BoxCar& operator=(const BoxCar& RHS);
};
```

```cpp
BoxCar::BoxCar(string L, int Sz) {
    Label = L;
    numCars = 0;
    if (Sz <= 0) {
        Size = 0;
        Cargo = NULL;
    }
    else {
        Size = Sz;
        Cargo = new Crate*[Size];
    }
}
```

```cpp
BoxCar::BoxCar(const BoxCar& RHS) {
    // implementation not shown
}
```

```cpp
bool BoxCar::addCrate(Crate*& newCrate) {
    if (numCars == Size)
        return false;
    Cargo[numCars] = newCrate;
    numCars++;
    newCrate = NULL;
    return true;
}
```

```cpp
BoxCar& BoxCar::operator=(const BoxCar& RHS) {
    // implementation not shown
}
```

```cpp
class Crate {
private:
    string Label;
public:
    Crate(string L = "None");
    string getLabel() const;
};
```

```cpp
Crate::Crate(string L) {
    Label = L;
}
```

```cpp
string Crate::getLabel() const {
    return Label;
}
```

4. [7 points] Consider execution of the following code fragment:

```cpp
for (int I = 0; I < 10; I++) {
    BoxCar* B = new BoxCar(100);  // Line 1
    delete B;                     // Line 2
}
```

Determine whether this code causes a memory leak. If yes, explain clearly how the leak occurs. If no, explain clearly what prevents a leak from occurring.

**First of all, the constructor invocation in Line 1 needs a `string` parameter. Absent that, the code would not compile. So if you said that, you got full credit. Ignoring that…**

**Line 1 allocates a new `BoxCar` object, which allocates an array of 100 `Crate` pointers (NOT 100 `Crate` objects). Line 2 deallocates the `BoxCar` object, causing its destructor to fire. However, the `BoxCar` destructor does not deallocate the array of `BoxCar` pointers, so that memory is never reclaimed.**

**That is a memory leak.**
5. [7 points] Does the class BoxCar need a destructor? If not, explain why not. If yes, write an implementation of the destructor.

Yes, a destructor is needed to deallocate the array of Crate pointers. Arguably, the destructor should also delete any Crate objects that are associated with the BoxCar object.

```cpp
BoxCar::~BoxCar() {
    for (int Idx = 0; Idx < Size; Idx++)
        delete Cargo[Idx];
    delete [] Cargo;
}
```

6. [7 points] List all class member functions that are invoked in executing the following code:

```cpp
BoxCar B1("Fred", 10), B2;
B2 = B1;
```

B1 is created by an invocation of the BoxCar constructor, and so is B2. B1 is then copied to B2 by BoxCar::operator=.

No Crate objects are created by the BoxCar constructors, so no Crate member functions are invoked.

7. [7 points] List all class member functions that are invoked in executing the following code:

```cpp
BoxCar B1("Fred", 10);
BoxCar B2 = B1;
```

B1 is created by an invocation of the BoxCar constructor. However, B2 is created by the BoxCar copy constructor, using B1 as its parameter.

Consider the following classes:

```cpp
class Operand {
private:
    int Op;
public:
    Operand(int V = 5) { Op = V; }
    int getOp() const { return Op; }
};

class Sum {
private:
    Operand RHS, LHS;
    int S;
public:
    Sum();
    // irrelevant fns not shownec
};
Sum::Sum() {
    S = RHS.getOp() + LHS.getOp();
};
```

8. [7 points] Is the relationship between Sum and Operand an association or something else? Justify your answer.

It is impossible for a Sum object to exist without two Operand objects, so this is NOT an association relationship. (It is, in fact, an aggregation.)
Consider the following class (which uses the BoxCar and Crate classes declared earlier):

```cpp
class Train {
private:
    int numCars;
    BoxCar* Cars[100];
public:
    Train();
    bool addCar(BoxCar* B);
    BoxCar* removeCar();
};
Train::Train() {
    numCars = 0;
    for (int I = 0; I < 100; I++)
        Cars[I] = NULL;
}
bool Train::addCar(BoxCar* B) {
    if (numCars == 100)
        return false;
    Cars[numCars] = B;
    numCars++;
    return true;
}
BoxCar* Train::removeCar() {
    if (numCars == 0)
        return NULL;
    BoxCar* T = Cars[numCars];
    Cars[numCars] = NULL;
    numCars--;
    return T;
}
```

9. [7 points] Is the relationship between Train and BoxCar an association or something else? Justify your answer.

A Train object can exist without any BoxCar objects (although not without any BoxCar pointers). Certainly BoxCar objects can exist without any Train objects. So, this IS an association.

Consider the following class:

```cpp
class Sentence {
private:
    string* Words;
public:
    Sentence(ifstream& In, int N);
    ~Sentence();
};
Sentence::Sentence(ifstream& In, int N) {
    if (N > 0) {
        Words = new string[N];
        // input code omitted
    } else Words = NULL;
}
Sentence::~Sentence() {
    delete [] Words;
}
```

10. [7 points] Is the relationship between Sentence and string an association or something else? Justify your answer.

A Sentence object does contain a pointer to something of type string. However, the use of a pointer does not mean the relationship is association.

The Sentence constructor creates an array of string objects, and the Sentence destructor deletes that array of string objects. There is no independence of existence, and so this is NOT an association.
11. [5 points] In C++, when an object is used as an actual parameter and passed to a function by reference, the formal parameter is:

1) a copy of the actual parameter, made by the assignment operator.
2) a copy of the actual parameter, made by the copy constructor.
3) **logically the same object as the actual parameter.**
4) This is not allowed.
5) None of these

Consider the description below of an alarm clock:

The Big Ben company came up with a clock that design experts still call one of the best ever, the Moon Beam. The Moon Beam clock flashes a gentle blinking alarm light for four minutes before the chime alarm sounds. Features a streamlined moonbeam-yellow case and genuine glass face with illuminated dial, snooze function, easy-to-read numerals, replaceable 25-watt bulb and a built in battery backup for power outages. 5”H, 6½”W, 2”D.

[7 points each] Choosing from the following answers,

```
object    class    attribute    behavior    none
```

in terms of designing an object-oriented model of the Moon Beam alarm clock determine whether each of the entities listed below is best characterized as a(n) ________ in the system, or if it is none.

12. alarm **Best answer is that this is a class.** A **Clock** contains two kinds of alarms, one that blinks and one that chimes. So "alarm" is a type of thing.

13. the built in battery **Best answer is an object.** "Battery" is clearly a type of thing (a class), but the "built-in battery" is a thing of that type (an object).

14. blinking **Best answer is a behavior** (of a sub-object, not of the **Clock** object).

15. 25-watt **Best answer is an attribute** (of a **Bulb** sub-object).