

**Instructions:**

- Print your name in the space provided below.
- Answer each question in the space provided. If you need to continue an answer onto the back of a page, clearly indicate that you have done so, and label the continuation with the question number.
- If you want partial credit, justify your answers briefly and concisely, even when justification is not explicitly required.
- There are 18 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           **Solution**            
printed

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

\_\_\_\_\_  
signed

1. [4 points] In C++, the keywords `public` and `private` may be used to promote which of the following S/E goals?
- |                   |                              |
|-------------------|------------------------------|
| 1) Generalization | <b>4) Information hiding</b> |
| 2) Abstraction    | 5) Rationalization           |
| 3) Separation     | 6) None of these             |
2. [4 points] In C++, when an object is used as an actual parameter and passed to a function by value, 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) the same object as the formal parameter.
  - 4) a pointer to the formal parameter.
  - 5) None of these

Consider the following class declaration:

```
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:

```
CPacMonster Doggett, SmokingMan;
```

3. [12 points] For each statement below indicate the line number(s) corresponding to the member function(s) that would be invoked if the statement were executed. If there is no appropriate member function write "error".

<code>Doggett.setColor(GREEN);</code>	<u>3</u>
<code>Doggett.Move(DOWN);</code>	<u>5</u>
<code>SmokingMan.setColor().Move(LEFT, 2);</code>	<u>3, then 5</u>
<code>SmokingMan.Move();</code>	<u>error</u>

Consider the following classes for keeping track of vote totals for candidates in an election:

```
class VM {
private:
    int    numCand;
    CCand* Total;
public:
    VM(const string P[], const int N);
    bool  countVote(const string& Name);
    CCand Report(int Idx) const;
    ~VM();
};

VM::VM(const string P[], int N) {

    Total = NULL;
    if (N <= 0) {
        numCand = 0;
    }
    else {
        Total = new CCand[N];
        numCand = N;
        for (int i = 0; i < numCand; i++) {
            Total[i] = CCand(P[i]);
        }
    }
}

bool VM::countVote(const string& Name) {

    for (int i = 0; i < numCand; i++)
        if (Name == Total[i].getName()) {
            Total[i].addVote();
            return true;
        }
    return false;
}

CCand VM::Report(int Idx) const {

    if (Idx < 0 || Idx >= numCand) {
        return CCand("None");
    }
    return Total[Idx];
}

VM::~~VM() {
    delete [] Total;
}
```

```
class CCand {
private:
    string Name;
    int    Votes;
public:
    CCand(string N = "");
    string getName() const;
    int    getVotes() const;
    void  addVote();
};

CCand::CCand(string N) {

    Name = N;
    Votes = 0;
}

string CCand::getName() const {

    return Name;
}

int CCand::getVotes() const {

    return Votes;
}

void CCand::addVote() {

    Votes++;
}
```

4. [4 points] Is the relationship between the classes VM and CCand an association? Why or why not?

**Each VM object contains a pointer to an array of CCand objects. That array is allocated when the VM object is created and destroyed by the VM destructor when the VM object is destroyed. So, these CCand objects have no existence apart from the encapsulating VM object, and the relationship is NOT one of association.**

For the next three questions, assume that a VM object named `VotingMachine` has been declared and properly initialized.

5. [5 points] Calling the function below with the actual parameter `VotingMachine` will result in an unfortunate side effect, even though the body of the function is correctly implemented. Describe the side effect briefly but clearly.

```
// Function to print a table of the results from a particular voting
// machine object.
void Results(const VM V) {

    int Idx = 0;
    CCand Current = V.Report(Idx);
    string CurrName = Current.getName();
    cout << "Candidate      Votes" << endl;

    while (CurrName != "None") {
        cout << CurrName
             << setw(20 - CurrName.length()) << Current.getVotes() << endl;

        Idx++;
        Current = V.Report(Idx);
        CurrName = Current.getName();
    }
}
```

**The `CCand` array of `VotingMachine` will be deleted at the end of the execution of the function `Results()`, when the local object `V` is destructed.**

**[The question is about what happens, not why it happens. The why is that the VM class doesn't provide a deep copy constructor, so the object `V` is a shallow copy of `VotingMachine` and their pointers have the same value, so the two objects share the same array.]**

6. [5 points] Which of the following terms best characterizes the side effect (NOT possible consequences of statements following the function call) referred to in question 5?
- 1) a memory leak
  - 2) **a dangling pointer**
  - 3) an access violation
  - 4) memory corruption
  - 5) none of these

**Explanation: after the destruction of `V`, `VotingMachine`'s pointer will be unchanged but its target will have been deallocated.**

7. [5 points] Which of the following should be done, specifically in order to eliminate the side effect referred to above?
- 1) **add a proper deep copy constructor to VM.**
  - 2) add a proper deep assignment operator to VM.
  - 3) add a proper deep copy constructor to `CCand`.
  - 4) add a proper deep assignment operator `CCand`.
  - 5) all of these
  - 6) 1 and 2 only
  - 7) 3 and 4 only
  - 8) none of these

**When a parameter is passed by value, the copy is handled by the copy constructor. It would be good to also provide an assignment overload, but that's not related to the side effect in question.**

Consider the following program:

```
class Associate {
private:
    int iX;
public:
    Associate(int n = 0);
    void Show() const;
};

Associate::Associate(int n) {
    iX = n;
}

void Associate::Show() const {
    cout << "Hmm, value is " << iX << endl;
}

class Association {
private:
    Associate* pA;
public:
    Association();
    Association& operator=(const Association& Other);
    void addAssociate(Associate& Other);
    void addAssociate(Associate* pOther);
    void remAssociate();
    void showAssociate() const;
    ~Association();
};

Association::Association() {
    pA = NULL;
}

void Association::addAssociate(Associate& Other) {
    pA = &Other;
}

void Association::addAssociate(Associate* pOther) {
    pA = pOther;
}

void Association::remAssociate() {
    delete pA;
    pA = NULL;
}

void Association::showAssociate() const {
    cout << "Object address is: " << pA << endl;
    pA->Show();
}

Association& Association::operator=(const Association& Other) {
    pA = Other.pA;
    return (*this);
}

Association::~~Association() {
}
```

```

int main() {
    Association A;
    Association B;
    Associate X(17);
    Associate* pA = new Associate;
        A.addAssociate(*pA);           // Line 1: creates an alias
    pA->Show();                         // Line 2:
    pA = NULL;                          // Line 3:
    A.addAssociate(X);                  // Line 4: causes a leak
    B.addAssociate(new Associate(7));    // Line 5:
    B.showAssociate();                 // Line 6:
    A = B;                              // Line 7: creates an alias
    B.remAssociate();                  // Line 8: causes a dangler
    A.showAssociate();                 // Line 9: illegal access
    return 0;
}

```

**Trace through the execution.  
Draw diagrams to keep things straight.**

8. [4 points] Circle the numbers of the lines, if any, that will cause the creation of an alias via pointers.

1      2      3      4      5      6      7      8      9      none

9. [4 points] Circle the numbers of the lines, if any, that will cause the creation of a dangling pointer; i.e., a pointer to an address that is no longer owned by the program.

1      2      3      4      5      6      7      8      9      none

10. [4 points] Circle the numbers of the lines, if any, that will cause a memory leak; i.e., the creation of memory owned by the program but inaccessible to it.

1      2      3      4      5      6      7      8      9      none

11. [4 points] Circle the numbers of the lines, if any, that will logically cause an illegal access; i.e., an attempt to access an address not owned by the program.

1      2      3      4      5      6      7      8      9      none

12. [4 points] Which of the following is printed by line 2?

- 1) **Hmm, x is 0**
- 2) Hmm, x is 17
- 3) Hmm, x is 42
- 4) none of these

**The correction that 'x' should be "value" was announced at the test.**

Consider the description below of an investment management system:

An investor manages a portfolio of stocks. She periodically receives information about the price being offered for a single share of a stock. Based upon that information she will decide whether to sell some, or all, of her shares of that stock. To make that decision she will consider the price she paid for each share when she bought the stock, as well as how long she has held the shares.

[4 points each] Choosing from the following answers,

object          class          attribute          behavior          none

determine whether each of the entities listed below is best characterized as a(n) \_\_\_\_\_ in the system, or if it is none.

13. date of purchase          **attribute (of a share)**
14. stock          **class**
15. sell          **behavior (of ??)**
16. share          **object (of type stock)**

Consider the description below of a part of a railroad system:

A railroad yard has several dozen train engines and hundreds of train cars. A train is composed of some number of engines and cars, with the requirement that every train have at least one engine. At different times, each engine and car may be a part of different trains, or of no train at all. The railroad yard manager must know which engines and cars belong to which trains, and which are unused. However, he does not particularly care about the order in which the cars of a train are lined up. The railroad yard also has a number of tracks (sidings) on which trains, engines, and cars may be stored temporarily. The manager must know which track each train, engine, and car is on.

A designer settles upon the following classes for this system, in no particular order:

Car                  Engine                  Train                  Track

17. [10 points] Which pairs of classes, if any, would exhibit an association relationship? Do not worry about whether the association would be one- or two-directional. Note: answers are pairs of classes!

**There are six possible pairs of classes here. The only pair that does not exhibit an association is (Engine, Car).**

Consider the description below of a refrigerator:

A **controller** in a **refrigerator** has four **sensors** that report the **temperature** and **humidity**, both inside and outside the refrigerator. Based upon those **readings**, the controller determines whether to turn on the **compressor** to lower the temperature inside the refrigerator, and for how long to run the compressor. The compressor consists of an **electric motor** which drives a **pump** and a **length of tubing** within which the pump circulates **coolant**. The compressor monitors its internal temperature and will turn itself off if that temperature is too high.

18. [15 points] Identify a reasonable set of classes for this system. Give a descriptive name and a one-line description of the purpose of each class. Your analysis leading to the set of classes will not be graded, only the end result. Nevertheless, you should apply some structured process, such as that of Abbott and Booch, or Coad and Yourdon.

**Identifying nouns, we have the highlighted terms. The most basic ones are:**

**Controller**  
**Temperature Sensor**  
**Humidity Sensor**  
**Compressor**

**Each of these is essential (although you may have subsumed both types of sensor under one class).**

**In addition, the system as a whole is an object. It has attributes and serves as an organizational interface for the subparts listed above, so we should also have the class:**

**Refrigerator**

**Beyond these, there are arguable candidates:**

**Electric Motor, Pump, Tubing, Coolant(?)**

**Temperature and Humidity are not themselves classes; rather, they are attributes (measurements) that may belong to other classes. A "reading" may be considered a candidate for class-hood provided it encapsulates more than just a single value to be reported to another class.**