

**Instructions:**

- Print your name in the space provided below.
- Answer each question in the space provided. Legibility and organization count. If I can't read your answer, it's wrong.
- If you want partial credit, justify your answers briefly and concisely, even when justification is not explicitly required.
- There are 13 questions, priced as marked. The maximum score is 100.
- The code examples given in the test have been compiled and executed to verify correctness. Unless a question explicitly concerns itself with whether syntax is correct, you should assume that it is.
- 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.
- When you have completed the test, sign the pledge at the bottom of this page and turn in the test.

Do not start the test until instructed to do so!

Name _____
printed

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

signed

For questions 1 through 5, consider the following class declaration and implementation:

```
class Point {
    friend ostream& operator<<(ostream& Out, const Point& P);
private:
    int    x, y;
public:
    Point(int ix = 0, int iy = 0);
    Point Shift(int deltaX, int deltaY);
    Point xReflect();
    Point yReflect();
};

Point::Point(int ix, int iy) {

    x = ix;  y = iy;
}

Point Point::Shift(int deltaX, int deltaY) {

    x += deltaX;  y+= deltaY;
    return (*this);
}

Point Point::xReflect() {

    y = -y;
    return (*this);
}

Point Point::yReflect() {

    x = -x;
    return (*this);
}

ostream& operator<<(ostream& Out, const Point& P) {

    Out << "(" << P.x << ", " << P.y << ")";
    return Out;
}
```

1. [8 points] The given class does not implement `operator=`. Should an implementation of `Point::operator=` be added to the class in order to make it more useful to client code? Justify your conclusion carefully.

2. [5 points] What would be output by the following code?

```
Point A(14, 32);  
cout << A.Shift(1, 1).xReflect() << endl;
```

3. [5 points] What would be output by the following code?

```
Point B(14, 32);  
B.Shift(1, 1).xReflect();  
cout << B << endl;
```

4. [8 points] In the given implementation, `operator<<` is **not** a member of the class `Point`. **Could** it be a member of `Point`? Explain.

5. [7 points] The class `Point` declares that `operator<<` is a friend. Is this necessary to make the **given** implementation of `operator<<` valid? Explain.

-
6. [5 points] In C++, when an object is used as an actual parameter (i.e., in a function call) and passed to a function by constant reference, the formal parameter (i.e., in the function implementation) 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

For questions 7 through 9, consider the following class:

```
class Buffer {
private:
    char* B;
    int   Size;
public:
    Buffer(int Sz = 100);
    bool Acquire(istream& In);
    void Display(ostream& Out);
    void Clear();
    ~Buffer();
};

Buffer::Buffer(int Sz) {
    Size = Sz;
    B = new char[Size];
    if ( B == NULL )
        Size = 0;
    else
        Clear();
}
. . .
Buffer::~~Buffer() {
    delete [] B;
}
```

7. [8 points] Is the relationship between the class `Buffer` and the class (type) `char` an association or an aggregation? Justify your answer carefully.

8. [8 points] The relationship between the class `Buffer` and the class `istream` is an association, **not** an aggregation. Explain why.

9. [8 points] Is the relationship between `Buffer` and `istream` a **static** association or a **dynamic** association? Explain.

For questions 10 and 11, consider the following classes:

```

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);
    ~Boxcar();
};

BoxCar::BoxCar(string L, int Sz) {

    Label = L;
    numCars = 0;
    if (Sz <= 0) {
        Size = 0;
        Cargo = NULL;
    }
    else {
        Size = Sz;
        Cargo = new Crate*[Size];
    }
}

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

bool BoxCar::addCrate(Crate*& newCrate) {

    if (numCars == Size)
        return false;
    Cargo[numCars] = newCrate;
    numCars++;
    newCrate = NULL;
    return true;
}

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

Boxcar::~~Boxcar() {
    delete [] Cargo;
}

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

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

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

```

10. [8 points] Consider execution of the following code fragment:

```

for (int I = 0; I < 10; I++) {
    BoxCar* B = new BoxCar(100); // note: NOT an array declaration!!
    delete B;
}

```

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.

11. [8 points] Does the class `Crate` need a destructor in order to prevent memory leaks? If not, give a complete explanation why not. If yes, write an implementation of the needed destructor.

For questions 12 and 13, consider the following class declaration and implementation:

```
class Polynomial {
private:
    int* Coeff;
    int Degree;

public:
    Polynomial(int Deg, int C[]);
    Polynomial(const Polynomial& Source);
    Polynomial operator=(const Polynomial& RHS);
    . . .
    int evalAt(int X) const;
    ~Polynomial();
};

Polynomial::Polynomial(int Deg, int C[]) {
    Degree = Deg;
    Coeff = new int[Deg + 1];           // # of coeff's == degree + 1
    for (int Pos = 0; Pos <= Deg; Pos++)
        Coeff[Pos] = C[Pos];
}

. . .
int Polynomial::evalAt(int X) const {
    int Value;
    Value = Coeff[Degree];
    for (int Pow = Degree - 1; Pow >= 0; Pow--)
        Value = Value * X + Coeff[Pos];
    return Value;
}

Polynomial::~~Polynomial() {
    delete [] Coeff;
}
```

12. [10 points] Write the implementation of the copy constructor for the class `Polynomial`. (You may not assume that there is already a correct implementation of `operator=` for `Polynomial` objects.)

13. [12 points] Complete the implementation of the following member operator:

```
Polynomial Polynomial::operator+(const Polynomial& RHS) const {
```

```
}
```