Instructions: This homework assignment focuses on basic facts regarding classes in C++. Submit your answers via the Curator System as Quiz: Classes.

For the next three questions, consider the class declaration:

```
class CreditCard {
public:
    CreditCard(double Amount)       { Bal = Amount; }
    void   Pay( double Amount )     { Bal -= Amount; }
    void   Charge( double Amount )  { Bal += Amount; }
    double Balance() const          { return Bal; }

private:
    double Bal;
};
```

Member function implementations put inline to save space.

1. Which member function provides an observer/reporter operation?

1) `Pay`	3) `Balance`	5) 1, 2, and 3
2) `Charge`	4) 1 and 2 only	6) None of these

2. Which member function provides a mutator operation?

1) `Pay`	3) `Balance`	5) 1, 2, and 3
2) `Charge`	4) 1 and 2 only	6) None of these

3. Consider the following statements, assuming the class declaration for `CreditCard` is in scope:

```
CreditCard WesternUnion(1000.0);
CreditCard BankAmerica(0.0);
WesternUnion.Pay(400.0);          // line 1
cout << WesternUnion.Bal;         //      2
BankAmerica = WesternUnion;       //      3
CreditCard.Charge(120.0);         //      4
```

Which statements would <u>not</u> cause compilation errors?

1) 1 only	5) 1 and 3 only	9) 2, 3 and 4 only
2) 2 only	6) 1 and 4 only	10) None of these
3) 3 only	7) 2 and 3 only	
4) 4 only	8) 2 and 4 only	

---

4. Which of the following statements about C++ classes is <u>false</u>?

1) Classes can have private member functions.
2) Classes can have public, private and protected data members.
3) By default, members of classes are private.
4) Classes can have public data members.
5) Aggregate assignment is permitted for classes.
6) None of these, (all are true).

---

5. Which of the following C++ built-in operations are <u>not</u> automatically defined for class objects?

1) =	4) 1 and 2 only	7) All of them
2) ==	5) 1 and 3 only	8) None of these
3) <<	6) 2 and 3 only	

6. A class `A` has a public member function `G` that takes one `int` parameter for input, returns a `bool` value, and does not modify any of the class data members. Which of the following would be the best correct function prototype for G in the class declaration?

   1) `bool G(int& x);`
   2) `const bool G(int x);`
   3) `bool G(int& x) const;`
   4) `const bool G(int& x);`
   5) `bool G(int x) const;`
   6) None of these

---

7. Suppose that the declaration of the class `A` includes the following function prototype.

   ```
   bool LessThan( const A& RHS );
   ```

   Which of the following tests in the client code correctly compares two `A` objects named `Alpha` and `Beta`?

   1) `if (Alpha < Beta)`
   2) `if (Alpha.LessThan(Beta))`
   3) `if (LessThan(Alpha, Beta))`
   4) `if (Alpha.LessThan.Beta)`
   5) `if (LessThan(Alpha).Beta)`
   6) None of these

---

8. If the designer of a C++ class wishes to allow clients to inspect and modify a data member, what is the best approach?

   1) Make the data member private and provide a public observer function as a class member.
   2) Make the data member private and provide a public mutator function as a class member.
   3) Make the data member private and provide a public mutator function and a public observer function as class members.
   4) Declare the data to be public, not private.
   5) Provide a public constructor that takes a value for that data member as a parameter.
   6) Do nothing because it is not acceptable to let clients modify data members.
   7) None of these

Consider the following class:

```
class B {
private:
   int S;
public:
   B() {}
   B(int initS) { S = initS; }
};
```

9. Assuming everything necessary is in scope, consider the declaration: `B Foo;`

   What is the value of `Foo.S`?

   1) 0
   2) Unknown
   3) The declaration isn't allowed.
   4) None of these

10. Assuming everything necessary is in scope, what is the value of `Bar.S` after executing the code fragment:

   ```
   B Foo(17), Bar(32);
   Bar = Foo;
   ```

   1) 17
   2) 32
   3) Unknown
   4) The assignment isn't allowed.
   5) None of these

For the next six questions, consider the class declaration:

```
class Farey {
private:
   int Top, Bottom;
public:
   Farey();
   Farey(int T, int B);
   Farey operator+(const Farey& RHS) const;
   Farey operator-(const Farey& RHS) const;
   bool  operator==(const Farey& RHS) const;
   void  Display(ostream& Out) const;
};

Farey::Farey() {

   Top = Bottom = 0;
}

Farey::Farey(int T, int B) {

   Top    = T;
   Bottom = B;
}

Farey Farey::operator+(const Farey& RHS) const {

   return Farey(Top + RHS.Top, Bottom + RHS.Bottom);
}

Farey Farey::operator-(const Farey& RHS) const {

   return Farey(Top - RHS.Top, Bottom - RHS.Bottom);
}

bool Farey::operator==(const Farey& RHS) const {

   return ( (Top == RHS.Top) && (Bottom == RHS.Bottom) );
}

void Farey::Display(ostream& Out) const {

   Out << Top << '/' << Bottom;
}
```

Again, assuming everything necessary is in scope, consider the following code fragment:

```
Farey A(3, 5), B(1, 4), C(2, 4), D(0, 5), E;
E = A + B;                        // line 1
A.Display(cout);                  //      2
E = A + B - C;                    //      3
E = 2*A;                          //      4
```

11. After the execution of line 1, what are the values of `E.Top` and `E.Bottom`, respectively?

1) 0 and 0
2) 3 and 5
3) 1 and 4
4) 4 and 9
5) Unknown
6) None of these

12. What is written to the stream `cout` when line 2 is executed?

1) `"3/5"`	2) Nothing	3) None of these

13. After the execution of line 3, what are the values of `E.Top` and `E.Bottom`, respectively?

1) 0 and 0	3) 2 and 5	5) Unknown
2) 4 and 9	4) The statement isn't allowed.	6) None of these

14. After the execution of line 4, what are the values of `E.Top` and `E.Bottom`, respectively?

1) 6 and 10	3) 3 and 10	5) Unknown
2) 6 and 5	4) The statement isn't allowed.	6) None of these

Consider the following code fragment:

```
Farey X(1, 2), Y(2, 4);

if ( X == Y )                       // line 5
   cout << "X == Y" << endl;
else
   cout << "X != Y" << endl;

if ( X + X == Y )                   // line 6
   cout << "X + X == Y" << endl;
else
   cout << "X + X != Y" << endl;
```

15. When the `if` statement beginning in line 5 is executed, what is written to `cout`?

1) `"X == Y"`   2) `"X != Y"`

16. When the `if` statement beginning in line 6 is executed, what is written to `cout`?

1) `"X + X == Y"`   2) `"X + X != Y"`

For the next five questions, consider the following class:

```
class Quadratic {
private:
   double Coefficient[3];
public:
   Quadratic(double a = 0.0, double b = 0.0, double c = 0.0);
   double Evaluate(double x) const;
};

Quadratic::Quadratic(double a, double b, double c) {
   Coefficient[0] = a;
   Coefficient[1] = b;
   Coefficient[2] = c;
}
```

```
double Quadratic::Evaluate(double x) const {
   return ( (Coefficient[2]*x + Coefficient[1])*x + Coefficient[0] );
}
```

17. Given the declaration: `Quadratic F(1, 2, 3);`

    What value is output by the statement: `cout << F.Evaluate(2);`

    1) `6` 2) `11` 3) `17` 4) None of these

18. Given the declaration: `Quadratic G(1);`

    What value is output by the statement: `cout << G.Evaluate(2);`

    1) `1` 2) `4` 3) Not allowed. 4) None of these

A designer wants to add an addition operation to the class `Quadratic`. Consider the partial implementation:

```
______________ Quadratic::operator+(const Quadratic& RHS) const {   // line 1

   double a = Coefficient[0] + RHS.Coefficient[0];
   double b = Coefficient[1] + RHS.Coefficient[1];
   double c = Coefficient[2] + RHS.Coefficient[2];

   return ________________;                                        // line 2
}
```

19. How should the blank in line 1 be filled?

    1) `void`
    2) `Sum`
    3) `Quadratic`
    4) It should be left blank.
    5) None of these

20. How should the blank in line 2 be filled?

    1) `Quadratic(a, b, c)`
    2) `Sum`
    3) `Quadratic(c, b, a)`
    4) It should be left blank.
    5) None of these

21. The algebraic expression in the member function Evaluate() could have been written as:

    `Coefficient[2]*x*x + Coefficient[1]*x + Coefficient[0]`

    Why was the approach shown above not used?

    1) It isn't the right algebraic expression.
    2) The original version requires fewer operations to evaluate.
    3) The original version required typing fewer characters.
    4) All of these
    5) 1 and 2 only
    6) 1 and 3 only
    7) 2 and 3 only
    8) None of these

For the next six questions, consider the following class declaration:

```
enum Response {PRODUCT, CHECKPRICE, OUTOFSTOCK};
class Vendor {
private:
   unsigned int Stock;    // number of units available
   unsigned int Balance;  // total payments received
   unsigned int Price;    // unit price for product
   string passCode;       // security passcode
   unsigned int Capacity; // max units vendor can hold

public:
   Vendor();
   Vendor(unsigned int Stk, unsigned int P, string Code);
   Response MakePurchase(unsigned int Payment);
   void addProduct(unsigned int Quantity, string Code);
   unsigned int removeMoney(string Code);
   bool changePrice(unsigned int P, string Code);
   ~Vendor();
};
```

For the next three questions, consider implementing the member function `MakePurchase()`.

```
Response Vendor::MakePurchase(unsigned int Payment) {

      if ( ________________ ) return CHECKPRICE;  // correct amount?
      if ( ________________ ) return OUTOFSTOCK;  // stock available?
      Stock--;                                    // update stock
      Balance += Payment;                         // record payment
      return ______________;                      // return product
}
```

22. The purpose of the first `if` statement is to make sure that the correct amount of money has been offered. A `Vendor` does not support making change. How should the blank in that line be filled?

   1) `Payment < Price`
   2) `Payment != Price`
   3) `Payment != Vendor.Price`
   4) There is no correct way to do this.
   5) None of these

23. The purpose of the second `if` statement is to make sure that there is an item in stock to dispense in return for payment. How should the blank in that line be filled?

   1) `Stock != 0`
   2) `Stock <= Capacity`
   3) `Vendor.Stock == 0`
   4) There is no correct way to do this.
   5) None of these

24. What value should be returned in the final statement?

   1) `CHECKPRICE`
   2) `OUTOFSTOCK`
   3) `PRODUCT`
   4) There is no correct way to do this.
   5) None of these

For the next question, consider an implementation the member function `changePrice()`:

```
bool Vendor::changePrice(unsigned int P, string Code) {

   if ( Code == passCode ) {  // authenticate caller
      Price = P;              // reset price
      return true;            // confirm reset
   }
   return false;              // deny reset
}
```

25. What design features of `Vendor` support restricting who can change the price of an item?

1) Making `passCode` private.	5) 1 and 2 only
2) Not providing a mutator for `passCode`.	6) 1 and 3 only
3) Providing `changePrice()` as public.	7) 2 and 3 only
4) All of these	8) None of these

26. Which of the following would be logically and syntactically valid declarations of `Vendor` objects?

```
Vendor M0000;                         // 1
Vendor M0001(100, 50, "wowonfie");    // 2
Vendor M0002(300, 100, "", 300);      // 3
```

1) 1 only	5) 1 and 2 only
2) 2 only	6) 1 and 3 only
3) 3 only	7) 2 and 3 only
4) All of them	8) None of these

27. Which of the following are valid reasons why the implementation of `Vendor` uses `unsigned int` instead of `int` for some data members?

1) To reduce the amount of memory a `Vendor` object requires.	6) 1 and 3 only
2) Because those data members cannot, logically, be negative.	7) 2 and 3 only
3) To reduce the amount of code needed for member functions.	8) None of these
4) All of them	
5) 1 and 2 only	

For the next three questions, consider the following class that represents a todo list. Tasks are represented by `string` objects, and stored in an array of dimension 100. Tasks are always added at the tail of the list, and removed from the front. To make the use of the array efficient, it is used in a circular manner.

```
const unsigned int SIZE = 100;

class todoList {
public:
   todoList();                             // set up empty task list
   bool addTask(const string& Task);       // add a task to the list
   string getNextTask() const;             // see what the next task is
   bool delTask();                         // remove the next task
   void Display(ostream& Out) const;       // see the whole list
   void Clear();                           // remove all the tasks
   ~todoList();

private:
   unsigned int putNext;                   // index where next task will go
   unsigned int doNext;                    // index of next task to do
   unsigned int nTasks;                    // number of tasks in the list
   string       List[SIZE];                // list of tasks
};
```

28. What, if anything, is wrong with the following implementation of the member function to add a new task to the list?

```
bool todoList::addTask(const string& Task) {

   List[putNext] = Task;
   putNext = (putNext + 1) % SIZE;
   nTasks++;
   return true;
}
```

1) If the list is full, it will replace the first uncompleted task with a new one.
2) It updates the index `putNext` incorrectly.
3) It doesn't need a return value, since it always returns the same thing.
4) All of these
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) Something not listed
9) Nothing is wrong

29. What, if anything, is wrong with the following implementation of the member function to delete a task from the list?

```
bool todoList::delTask() {

   if ( nTasks == 0 )
      return false;

   doNext = ( doNext + SIZE - 1 ) % SIZE;
   nTasks--;
   return true;
}
```

1) If the list is empty, it will cause a runtime error.
2) It updates the index `doNext` incorrectly.
3) It should be declared as a `const` member function.
4) All of these
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) Something not listed
9) Nothing is wrong

30. What, if anything, is wrong with the following implementation of the member function to display the tasks in the list?

```
void todoList::Display(ostream& Out) const {

   if ( nTasks == 0 )

      Out << "Todo list is empty." << endl;

   else {

      for (unsigned int Pos = 0; Pos < nTasks; Pos++) {

         Out << setw(3) << Pos << ":  ";
         Out << List[Pos] << endl;
         Pos++;
      }
   }
}
```

1) It doesn't necessarily start with the next task to do.
2) It doesn't necessarily stop with the last task to do.
3) It doesn't necessarily print all of the tasks waiting to be done.
4) All of these
5) 1 and 2 only
6) 1 and 3 only
7) 2 and 3 only
8) None of these