

READ THIS NOW!

- Print your name in the space provided below.
- Print your name and ID number on the Opscan form; be sure to code your ID number on the Opscan form. Code **Form A** on the Opscan.
- Choose the single best answer for each question — some answers may be partially correct. If you mark more than one answer, it will be counted wrong.
- Unless a question involves determining whether given C++ code is syntactically correct, assume that it is valid. The given code has been compiled and tested, except where there are deliberate errors. Unless a question specifically deals with compiler `#include` directives, you should assume the necessary header files have been included.
- Be careful to distinguish integer values from floating point (real) values (containing a decimal point). In questions/answers which require a distinction between integer and real values, integers will be represented without a decimal point, whereas real values will have a decimal point, [2704 (integer), 2704.0 (real)].
- The answers you mark on the Opscan form will be considered your official answers.
- 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.
- There are 20 questions, equally weighted. The maximum score on this test is 100 points.

Do not start the test until instructed to do so!

Print Name (Last, First) _____

Solution

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

N. D. Barnette
signature

#1 Association is what type of relationship?

- (1) "is_a" relationship
- (2) "has_a" relationship
- (3) "knows_a" relationship
- (4) None of the above.

#2 Aggregation is what type of relationship?

- (1) "is_a" relationship
- (2) "has_a" relationship
- (3) "knows_a" relationship
- (4) None of the above.

#3 Inheritance is what type of relationship?

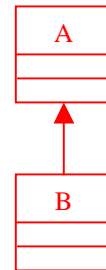
- (1) "is_a" relationship
- (2) "has_a" relationship
- (3) "knows_a" relationship
- (4) None of the above.

For the next three questions, assume that `classA` and `classB` are C++ classes, and that the class `classB` is derived, using public inheritance, from the class `classA`.

#4 If `BB` is an object of type `classB`, then the member functions of `BB`:

- (1) cannot directly access any of the members of class `classA`.
- (2) can directly access only the public members of class `classA`. (= 2.5 points)
- (3) can directly access only the protected members of class `classA`. (= 2.5 points)
- (4) can directly access the public, protected, and private members of class `classA`.
- (5) None of the above.

Class Diagram



#5 Suppose that an object `BB` of type `classB` is declared. Assume that in the default constructor for class `classB` that no constructor member initializer list is present. Then when the object `BB` is instantiated :

- (1) the constructor for the base class, `classA`, will NOT be executed at all.
- (2) a constructor for the base class, `classA`, will be executed after any constructor for the derived class, `classB`.
- (3) a constructor for the base class, `classA`, will be executed before any constructor for the derived class, `classB`.
- (4) constructors for both classes will be executed concurrently.
- (5) None of the above.

#6 Suppose that an object `BB` of type `classB` is declared. When the lifetime of object `BB` ends:

- (1) no destructor for the base class, `classA`, will be executed at all.
- (2) the destructor for the base class, `classA`, will be executed after the destructor for the derived class, `classB`.
- (3) the destructor for the base class, `classA`, will be executed before the destructor for the derived class, `classB`.
- (4) destructors for both classes will be executed concurrently.
- (5) None of the above.

For the next two questions, assume that `classA` and `classB` are C++ classes, and that the class `classB` is derived, using public inheritance, from the class `classA`. Further, suppose that `classA` has a void public member function `G()`, which `classB` overrides, (NOT extends), with its own void public member function `G()`. During execution, assume that the address of a `classB` object is passed, [e.g., `H(&BB);`] to the function below:

```
void H(classA* AA) {  
    AA->G();  
}
```

#7 Which version of function `G()` is executed in function `H()`, assuming that `G()` is **NOT** declared virtual in `classA`?

- (1) `classA`'s function `G()` is invoked
- (2) `classB`'s function `G()` is invoked
- (3) both versions are invoked
- (4) None of the above

Without any virtual functions no polymorphism can occur.

#8 Which version of function `G()` is executed in function `H()`, assuming that `G()` is declared virtual in `classA`?

- (1) `classA`'s function `G()` is invoked
- (2) `classB`'s function `G()` is invoked
- (3) both versions are invoked
- (4) None of the above

With virtual functions polymorphism (dynamic binding) occurs.

For the next two questions, assume that `classA` and `classB` are C++ classes, and that the class `classB` is derived, using public inheritance, from the class `classA`. Further, suppose that `classA` has a void public member function `G()`, which `classB` redefines, (NOT extends), with its own void public member function `G()`. During execution, assume that the a `classB` object is passed, [e.g., `H(BB);`] to the function below:

```
void H(classA AA) {  
    AA.G();  
}
```

The dot member selection operator used to invoke functions results in static binding regardless of whether the function is declared virtual.

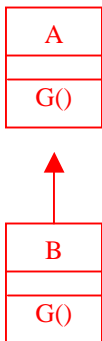
#9 Which version of function `G()` is executed in function `H()`, assuming that `G()` is **NOT** declared virtual in `classA`?

- (1) `classA`'s function `G()` is invoked
- (2) `classB`'s function `G()` is invoked
- (3) both versions are invoked
- (4) None of the above

#10 Which version of function `G()` is executed in function `H()`, assuming that `G()` is declared virtual in `classA`?

- (1) `classA`'s function `G()` is invoked
- (2) `classB`'s function `G()` is invoked
- (3) both versions are invoked
- (4) None of the above

Class Diagram
for #7 - #10



Given the following similar partial class declarations:

```
class Associate {
public:
    Associate() : x(0) {}
    void doSomething() { cout << "Wow, x is " << x << endl; }
private:
    int x;
};

class Association {
public:
    Association() : aptr(NULL) {}
    Association(const Association& a) : aptr(NULL) {}
    void add(Associate& a) { aptr = &a; }
    void use() { aptr->doSomething(); }
    Association& operator=(const Association& a) { aptr = NULL; }
    ~Association() {}
private:
    Associate* aptr;
};
```

Since the link from an Association object to an Associate object can be changed multiple times using the add() function it is a dynamic association.

#11 What type of association(s) does the above class declaration allow between the class objects?

- (1) Static
- (2) **Dynamic**
- (3) both (Static and Dynamic)
- (4) None of the above

Given the following partial class declarations:

```
class Associate {
public:
    Associate() : x(0) {}
    void doSomething() { cout << "Wow, x is " << x << endl; }
private:
    int x;
};

class Association {
public:
    Association(Associate& A) : aptr(&A) {}
    Association(const Association& a)
        {aptr= new Associate; *aptr = *(a.aptr);}
    void use() { aptr->doSomething(); }
    Association& operator=(const Association& a) { aptr = NULL; }
    ~Association() {}
private:
    Associate* aptr;
};
```

Since the link from an Association object to an Associate object can only be set once through the Association constructor function it is a static association.

#12 What type of association(s) does the above class declaration allow between the class objects?

- (1) **Static**
- (2) Dynamic
- (3) both (Static and Dynamic)
- (4) None of the above

Given the following partial class composition declarations:

```
class classA {
public:
    classA() : x(0) {}
    int  getX() {return x;}
    void setX(int b) {x=b;}
private:
    int x;
};

class Composition {
public:
    Composition() : aptr(NULL) {}
    Composition(int y) {aptr= new classA; aptr-> setX(y);}
    Composition (const Composition & a)
        {aptr= new classA; *aptr = *(a.aptr);}
    int  getX() { return aptr->getX(); }
    void setX(int y) { aptr->setX(y); }
    Composition& operator= (const Composition& a)
        {if (this != &a) {
            aptr= new classA;
            aptr->setX(a.aptr->getX());}
        }
    ~ Composition () {delete aptr;}
private:
    classA* aptr;
};
```

Since a classA object that is linked to a Composition object is created by the Composition object, (and does Not exist external to it), the composition/relationship between the classes is an aggregation.

#13 What type of composition does the above class declaration allow between the class objects?

- (1) Association
- (2) Aggregation
- (3) both (Association and Aggregation)
- (4) Inheritance
- (5) Polymorphism
- (6) None of the above.

#14 Assume that a catch block has matched a thrown exception and is executing. If the statement:

```
throw;
```

is executed in the catch block, which of the following would be the result?

- (1) an anonymous object would be thrown.
- (2) a catch all (. . .) would be thrown.
- (3) the exception that was caught would be rethrown.
- (4) the function throw list would cause the throw; to be skipped.
- (5) None of the above.

A throw; statement with no arguments executed inside a catch block causes the caught exception to be rethrown, continuing the un-raveling of the execution stack.

#15 If no catch handler matches the type of a thrown exception object which of the following results?

- (1) the thrown object goes out of scope.
- (2) the C++ language internal default supplied catch all (. . .) handler executes.
- (3) the fail() function in the assert library executes.
- (4) the program terminates and aborts execution.
- (5) None of the above.

Un-handled exceptions results in the terminate() function being executed which in turn calls the abort() function which halts program execution.

#16 Assume that the following error checks described are carried out in encapsulated classes and are handled by throwing an exception. Which one of the error checks described would be an inappropriate application of exception handling?

- (1) An attempt to take the square root of a negative number.
- (2) An extremely large dynamic memory allocation request.
- (3) An attempt to access an aggregated object through a NULL pointer.
- (4) An attempt to store a negative value in a weight data member field.
- (5) None of the above, (all are appropriate applications of exception handling).

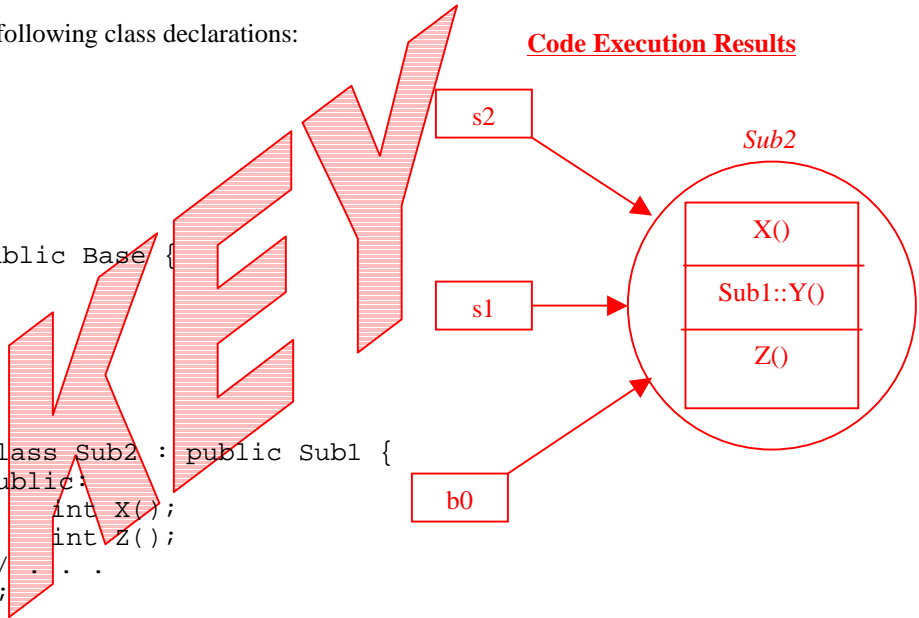
Exceptions should not be used for normal error checking, such as invalid data storage.

For the next three questions assume the following class declarations:

```
class Base {
public:
    int X();
    virtual int Y();
    virtual int Z() = 0;
};
```

```
class Sub1 : public Base {
public:
    int X();
    int Y();
    int Z();
    // . . .
};
```

```
class Sub2 : public Sub1 {
public:
    int X();
    int Z();
    // . . .
};
```



Assuming that appropriate implementations are supplied for each class, consider the following main() function:

```
void main() {
    Sub2* s2 = new Sub2;
    Sub1* s1 = (Sub1*) s2;
    Base* b0 = (Base*) s1;

    s2->Y(); // call 1
    s1->X(); // call 2
    b0->Z(); // call 3
}
```

#17 Which function is called in the statement labeled call 1?

- (1) Base::Y()
- (2) Sub1::Y()
- (3) Sub2::Y()
- (4) None of the above.

#17 Since function Y() is virtual, dynamic binding occurs. The type of object being pointed at is determined during execution. In this case, since the Sub2 class object does not contain a Y() member function, the call resolution proceeds up the inheritance hierarchy resulting in a call to the

#18 Which function is called in the statement labeled call 2?

- (1) Base::X()
- (2) Sub1::X()
- (3) Sub2::X()
- (4) None of the above.

#18 Since function X() is NOT virtual, static binding occurs based upon the type of the pointer, s1, resulting in a call to the Sub1::X() function.

#19 Which function is called in the statement labeled call 3?

- (1) Base::Z()
- (2) Sub1::Z()
- (3) Sub2::Z()
- (4) None of the above.

#19 Since function Z() is virtual, dynamic binding occurs. The type of object being pointed at, (Sub2) is determined during execution. In this case, since the Sub2 class object does contain a Z() member function, this results in a call to the Sub2::Z() function.

Consider the class composition declared below:

```

class M {
private:
    int m;
public:
    M() {
        cout << "Construct M" << endl;
        m = 0;
    }
    M(const M& Source) {
        m = Source.m;
        cout << "Construct M" << endl;
    }
    ~M() { cout << "Destruct M" << endl; }
};

class N {
private:
    M n;
public:
    N() : n() {cout << "Construct N" << endl;}
    N(M y) {
        cout << "Construct N" << endl;
        n = y;
    }
    ~N() { cout << "Destruct N" << endl; }
};
    
```

Trace the execution of the following simple program, (assuming the necessary inclusions):

```

void main( ) {
    M objM;
    N objN(objM);
}
    
```

#20 Which of the output sequences below would be produced by the instrumented functions above:

- | | |
|---|--|
| <p>(1) Construct M
 Construct M
 Construct M
 Construct N
 Destruct M
 Destruct N
 Destruct M
 Destruct M</p> | <p>(2) Construct M
 Construct M
 Construct N
 Destruct M
 Destruct N
 Destruct M</p> |
|---|--|

- (3) Construct M
 Construct M
 Construct N
 Construct M
 Destruct M
 Destruct M
 Destruct N
 Destruct M

The first output line is due to the constructor for objM being called. The second output line is due to the constructor for the n object inside of objN being called when objN is instantiated. The third output line is due to the copy constructor for class M being called when objM is passed by value to the N constructor. The fourth output line is due to the class N constructor for objN being called. The fifth output line is due to the destructor for y being called as the N constructor terminates. The sixth output line is due to the destructor for objN being called. The seventh output line is due to the destructor for the aggregated n object inside of objN being called. The eighth output line is due to the destructor for the objM object being called.