C++ quiz
You have about 1 minute per question.

1. Which one of the following statements is true with respect to the sample code shown below? *

```
1   class A
2   {
3   public:
4       virtual void RunIt();
5   };
6
7   class B : virtual public A
8   {
9   public:
10      void RunIt();
11  };
12
13  int main()
14  {
15      A *obj = new B();
16      obj->RunIt();
17
18      return 0;
19  }
```

1) A::RunIt() and B::RunIt() are invoked.
2) B::RunIt() is invoked.
3) There is a syntax error.
4) This is an example of the "diamond of death" inheritance problem.
5) A::RunIt() is invoked.

2. Referring to the sample code shown below, which one of the following statements is true? *

```
1   class A {};
2   class B : public A {};
3
4   template <class T> class C {};
5
6   int main()
7   {
8       C<A*> ca;
9       C<B*> cb;
10      return 0;
11  }
12
```

1) C<B*> can be converted to the base type C<A*>.
2) C<B*> has no inheritance relationship to C<A*>.
3) Neither C<A*> nor C<B*> can be instantiated.
4) C must define a default constructor.
5) cb will be sliced in the assignment to ca.

3. Referring to the sample code below, how do you access the member variable Counter WITHOUT creating an instance of the class Foo? *

```
1   class Foo
2   {
3       public:
4       static int Counter;
5   };
6
```

1) Foo().Counter
2) Foo->Counter
3) ::Counter
4) Foo::Counter
5) Foo.Counter

4. Referring to the sample code below, how does the compiler know which v to use in the increment function? *

```
1   class V { public: int v; };
2   class A : public virtual V { };
3   class B : public virtual V { };
4   class C : public A, public B { };
5
6   void incrementV()
7   {
8       C c;
9       c.v++;
10  }
11
```

1) C class uses non-virtual inheritance, so it has its own copy of "v."
2) Virtual inheritance means the compiler determines it polymorphically, based on the type of the object.
3) Since virtual inheritance is used to define A and B, there is only one "v."
4) Classes A and B both inherit from V, so the compiler will complain that this is an ambiguous resolution.
5) While four integer variables exist, each object can only see one, so it does not matter which one is visible, because the runtime behavior is consistent in all cases.

5. What is wrong with the sample code below? *

```
1   class SomeClass
2   {
3   protected:
4       void f(int z) const {x = y = z;}
5   private:
6       mutable int x;
7       int y;
8   };
9
```

1) f() is protected.
2) There are no errors in the code.
3) The mutable keyword is not being used in the proper context.
4) Assignments of the x = y = z form are invalid (multiple assignment).
5) Although x can be modified in f(), y cannot.

6. Which one of the following statements is correct? *

```
class A {
public:
    A() {
        if (x > 1)
            throw "x overflow";
        else
            x++;
    }

private:
    static int x;
};

int A::x;

```

1) A class can never be instantiated without throwing an exception.
2) It can not be determined how many times the class can be instantiated (unknown value for x).
3) The class can be instantiated without throwing an exception one time.
4) The class can be instantiated without throwing an exception two times.
5) A class can be instantiated as many times as desired without throwing an exception.

7. Regarding the time-critical function below, which one of the following statements is NOT correct? *

```
1   template <class C>
2   void testItems(C const &c)
3   {
4       for (C::const_iterator scan = c.begin();
5           scan < c.end();
6           scan++)
7       {
8           test(*scan);
9       }
10  }
11
```

1) The preference should be given to scan != c.end() instead of scan < c.end() for compatibility reasons.
2) Repeated calls to c.end() can be avoided by replacing them with a variable initialized once on function entry.
3) C::const_iterator should be replaced with C::iterator to improve performance.
4) The preference should be given to ++scan instead of scan++ for performance reasons.
5) Replacing the loop with a call to std::for_each() will improve performance on some C++ implementations.

8. Templates are often thought to be very similar to C-style macros. Which one of the following statements about templates is true? *

1) Macros are evaluated at compile time, while templates are at runtime.
2) Templates are strongly typed, while macros are not.
3) Macros have static linkage, while templates have external linkage.
4) Class templates are less efficient than class macros.
5) You can inherit from a macro.

9. Which one of the following solutions will allow to avoid the compile-time error in the sample code below? *

```
1   class SimpleString
2   {
3       char strBody[256];
4   public:
5       SimpleString(char *str)
6       {
7           strcpy(strBody, str);
8       }
9
10      std::ostream& operator<<(std::ostream &os, const SimpleString &s);
11  };
12
13  std::ostream& operator<<(std::ostream &os, const SimpleString &s)
14  {
15      os << s.strBody;
16      return os;
17  }
18
```

1) Using strncpy in the constructor instead of strcpy.
2) Defining a default constructor.
3) Making the operator<< function a friend of SimpleString.
4) Defining a destructor to cleanup the string data.
5) Including the <string> file header.

10. Which of the below options you will use to pass specific parameters from a derived class constructor to a base classes constructor? *

1) Late binding (polymorphism)
2) Virtual public inheritance
3) An initialization list
4) Virtual constructors
5) Dynamic allocation

11. If you want elements of pArray to point to the individual elements of array, which one of the following statements CANNOT be used? *

```
1   int array[3];
2   int* pArray[3];
3
4   int main()
5   {
6       //what cannot got here?
7   }
8
```

1)
```
for (int i = 0 ; i < 3; ++i)
pArray[i] = array + i;
```

2)
```
for (int i = 0 ; i < 3; ++i)
pArray[i] = array[i];
```

3)
```
*pArray = array;
pArray[1] = &array[1];
pArray[2] = &array[2];
```

4)
```
int ** pp = pArray;
for (int i = 2 ; i >= 0; i--)
*(pp + i) = array + i;
```

5)
```
for (int i = 0 ; i < 3; ++i)
pArray[i] = &array[i];
```

12. The sample code below has a memory leak. What has to be changed for this code to work properly? *

```
1   class Foo
2   {
3       char *fb;
4   public:
5       Foo(int x = 10) { fb = new char[x]; }
6       ~Foo() { delete[] fb; }
7   };
8
9   class Bar : public Foo
10  {
11      char *bb;
12  public:
13      Bar(int x = 10) { bb = new char[x]; }
14      ~Bar() { delete[] bb; }
15  };
16
17  int main()
18  {
19      Foo *f = new Bar;
20      delete f;
21      return 0;
22  }
23
```

1) Use a placement new when allocating.
2) Explicitly call ~Bar from ~Foo.
3) Explicitly call ~Foo from ~Bar.
4) Make ~Foo virtual.
5) Make the constructors virtual.

13. Referring to the sample code below, choose valid implementation of constructors. *

```
1   class Parent {
2   public:
3       Parent(int, char*, int);
4   };
5
6   class Child : public Parent {
7   public:
8       Child(int, int);
9   };
10
11  class GrandChild : public Child {
12  public:
13      GrandChild(int);
14  };
15
```

1)
```
Parent::Parent(int, char*, int) {}
Child::Child(int i, int x) {}
GrandChild::GrandChild(int i)
: Child(i, 0)
, Parent(i, 0, 0) {}
```

2)
```
Parent::Parent(int, char*, int) {}
Child::Child(int i, int x)
: Parent(i, 0, x) {}
GrandChild::GrandChild(int i)
: Child(i, 0) {}
```

3)
```
Parent::Parent(int, char*) {}
Child::Child(int i, int x)
: Parent(i, 0, x) {}
GrandChild::GrandChild(int i)
: Child(i, 0) {}
```

4)
```
Parent::Parent(int, char*, int) {}
Child::Child(int i, int x)
: Parent(i, 0, x) {}
GrandChild::GrandChild(int i) {}
```

5)
```
Parent::Parent(int, char*, int) {}
Child::Child(int i, int x)
: Parent {}
GrandChild::GrandChild(int i)
: Child(i, 0) {}
```

14. Referring to the sample code below, which one of the following data members is accessible from class Y? *

```
1   class X
2   {
3       int iCounter;
4
5   private:
6       char * pszSomeString;
7
8   protected:
9       float fSomeNumber;
10
11  public:
12      char cEos;
13  };
14
15  class Y : protected X { };
16
```

1) cEos only
2) fSomeNumber and cEos only
3) iCounter and cEos only
4) iCounter and fSomeNumber only
5) all members of X class

15. Why the parameter to a copy constructor MUST be a reference? *

1) Because a const reference is more efficient to pass than a copy
2) Because a reference type can be changed by the callee
3) Because a copy would invoke the copy constructor recursively
4) Because a class cannot contain pointers to its own type; it must use a reference
5) Because a const reference can be inlined by the compiler

16. Regarding C++ friendship, which one of the following statements is FALSE? *

1) If A class is a friend of B class, then A class can see all of B class's private data and methods.
2) If A class is a friend of B class and C class inherits from B, then A class is a friend of C.
3) Friendship must be granted, not taken.
4) You can declare both a friend function and a friend class.
5) If A class is a friend of B class, you cannot assume that B class is a friend of A class.

17. Why it is important to explicitly define a copy constructor when a class contains a pointer to dynamically allocated data? *

1) Because pointers must be copied using bitwise operators.
2) Because the default copy constructor only copies pointer values.
3) Because the default copy constructor deletes all pointer types.
4) Because pointer values are not retained in default copy constructors.
5) Because the default copy constructor uses deep copies.

18. Which one of the following statements about abstract classes is FALSE? *

1) They must have at least one function that is explicitly defined.
2) They can't be instantiated.
3) You can have pointers and references to abstract base classes.
4) They have at least one pure virtual function or derive from another abstract class.
5) They can be derived from.

19. What is a true statement about X::pc in the sample code above? *

```
1   class X
2   {
3   public:
4       X() { pc = new char [10]; }
5       ~X() { delete [] pc; }
6   private:
7       char *pc;
8   };
9
10  class Y : public X {};
11
12  void SomeFunction()
13  {
14      Y *py = new Y();
15      delete py;
16  }
17
```

1) It gets created, but it will not get properly destroyed.
2) It will both get created and properly destroyed.
3) Y can't inherit from X since X contains a private member variable that is referenced in the constructor for X.
4) Since pc is private, it is not created when a new class Y is created; it will neither get created nor destroyed.
5) It does not get created, but when py goes out of scope, an error will occur when it attempts to destroy it.

20. Which one of the following is the output of the below sample program? *

```
int main()
{
    try
    {
        std::cout << "A";
        try
        {
            throw "SomeText";
        }
        catch(const char * err)
        {
            std::cout << "B";
            throw;
        }
        catch(...)
        {
            std::cout << "C";
        }
    }
    catch(...)
    {
        std::cout << "D";
    }
    std::cout << "E";
}

```

1) AB
2) ABC
3) ABCE
4) ABDE
5) ACE

21. Is anything WRONG with the sample code below? *

```
1   char ca[] = "Foo";
2   char ca2[3];
3
4   int n = sizeof(ca);
5
6   for (int x = 0; x < n; ++x)
7       ca2[x] = *(ca + x);
8
```

1) No, the characters in ca are copied to ca2 without any side effects.
2) Yes, the syntax *(ca + x) is illegal.
3) Yes, the condition in that for loop should be x <= n, not x < n.
4) Yes, the size of ca was not given in the declaration.
5) Yes, too many elements will be copied from ca.

22. Which one of the following statements is a characteristic of protected inheritance? *

1) Public members of the derived class are privately accessible from the base class.
2) The derived class has non-public inheritance access to all but the private members of the base class.
3) The derived class has access to all members of the base class.
4) The base class has access only to the public or protected members of the derived class.
5) The private members of the base class are visible within the derived class.

23. Which one of the following expressions is equivalent to arr[1][5]? *

```
int arr[10] [10];
```

1) *(arr[10])

2) *((int*)arr + 10)

3) *(arr[15])

4) arr[5][1]

5) *((int*)arr + 15)

24. Referring to the sample code below, select a proper implementation of the operator= overload within the Bar class? *

```
1   class Foo
2   {
3       int x;
4   public:
5       const Foo& operator=(const Foo& rhs);
6   };
7
8   class Bar : public Foo
9   {
10      int y;
11  public:
12      const Bar& operator=(const Bar& rhs);
13  };
14
```

1)

```
const Bar& Bar::operator=(const Bar& rhs)
{
    if (this != &rhs)
    {
        y = rhs.y;
        Foo::operator=(rhs);
    }
    return *this;
}
```

2)

```
const Bar& Bar::operator=(const Bar& rhs)
{
    if (this != &rhs)
    {
        y = rhs.y;
        x = rhs.x;
    }
    return *this;
}
```

3)

```
const Bar& Bar::operator=(const Bar& rhs)
{
    if (this != &rhs)
    {
        y = rhs->y;
        Foo::operator=(rhs);
    }
    return *this;
}
```

4)

```
const Bar& Bar::operator=(const Bar& rhs)
{
    y = rhs.y;
    x = rhs.x;
    return this;
}
```

5)

```
const Bar& Bar::operator=(const Bar& rhs)
{
    if (this != &rhs)
    {
        y = rhs.y;
        Foo::operator=(rhs);
    }
    return this;
}
```

25. Examine the sample code below. What number will be printed? *

```
#include <iostream>
#include <utility>

int f(int&) { return 1; }
int f(int&&) { return 2; }

template <class T> int a(T&& x) { return f(x); }
template <class T> int b(T&& x) { return f(std::move(x)); }
template <class T> int c(T&& x) { return f(std::forward<T>(x)); }

int main() {
    int i = 10;
    std::cout << a(i) << a(20);
    std::cout << b(i) << b(20);
    std::cout << c(i) << c(20);
    return 0;
}

```

1) 112222
2) 112211
3) 112212
4) 121212
5) 122211

26. Which of the statements is true regarding std::move() applied to the instance of A class? *

1) A compilation error will occur if A class is not default constructible.
2) A runtime error will occur if A class doesn't support move semantics.
3) std::move() will copy only those members of A class, which belong to fundamental types.
4) std::move() will move only those members of A class, which belong to user-defined types.
5) std::move() is just a type conversion.

27. Which one of the statements is true? *

1) std::make_shared() is just a syntactic sugar for std::shared_ptr<> construction.
2) std::make_shared() is a little more efficient than direct construction of std::shared_ptr<>.
3) std::make_shared() will not throw.
4) std::make_shared() allows to specify a custom deleter.
5) std::make_shared() is a synonym for std::allocate_shared().

28. The sample code below doesn't compile. How would you fix it? *

```
1   typedef void(*Callback)(int sum);
2   void CalcSumAsync(int a, int b, Callback c)
3   {
4       c(a + b);
5   }
6
7   int main()
8   {
9       int a = 100;
10      int b = 23;
11      CalcSumAsync(a, b, [](int sum)
12      {
13          std::cout << a << "+" << b << "=" << sum;
14      });
15      return 0;
16  }
17
```

1) Get rid of lambda since it can't be used in this context.
2) Capture everything by value to fix compilation.
3) Capture everything by reference to fix compilation.
4) Rewrite "Callback" signature and "CalcSumAsync" function to pass "a" and "b" as parameters to the callback "c".
5) Make "Callback" a template parameter of "CalcSumAsync" and capture "a" and "b" by value.

29. Examine the sample code below. What number will be printed? *

```
#include <iostream>

int main()
{
    auto a = 1;
    auto b = 2;
    auto c = [&](auto d, auto e)
    {
        return [=]()
        {
            return (a + b) - (d + e);
        };
    }(++a, b++);
    std::cout << c();
    return 0;
}

```

1) 0
2) 1
3) 2
4) 3
5) 4

30. Which of the statements is true regarding the sample code below? *

```
1   #include <string>
2   #include <utility>
3
4   struct T
5   {
6       int i;
7       bool b;
8       std::string s;
9   };
10
11  int main()
12  {
13      T t{ 42, true, "test" };
14      T t2 = std::move(t);
15      return 0;
16  }
17
```

1) It will not compile.
2) It will throw exception at runtime.
3) It will work, t.i = 0, t.b = false, t.s = "".
4) It will work, t.i = 42, t.b = true, t.s = "test".
5) It will work, t is left in a valid but unspecified state (e.g. t.i = 42, t.b = true, t.s = "").