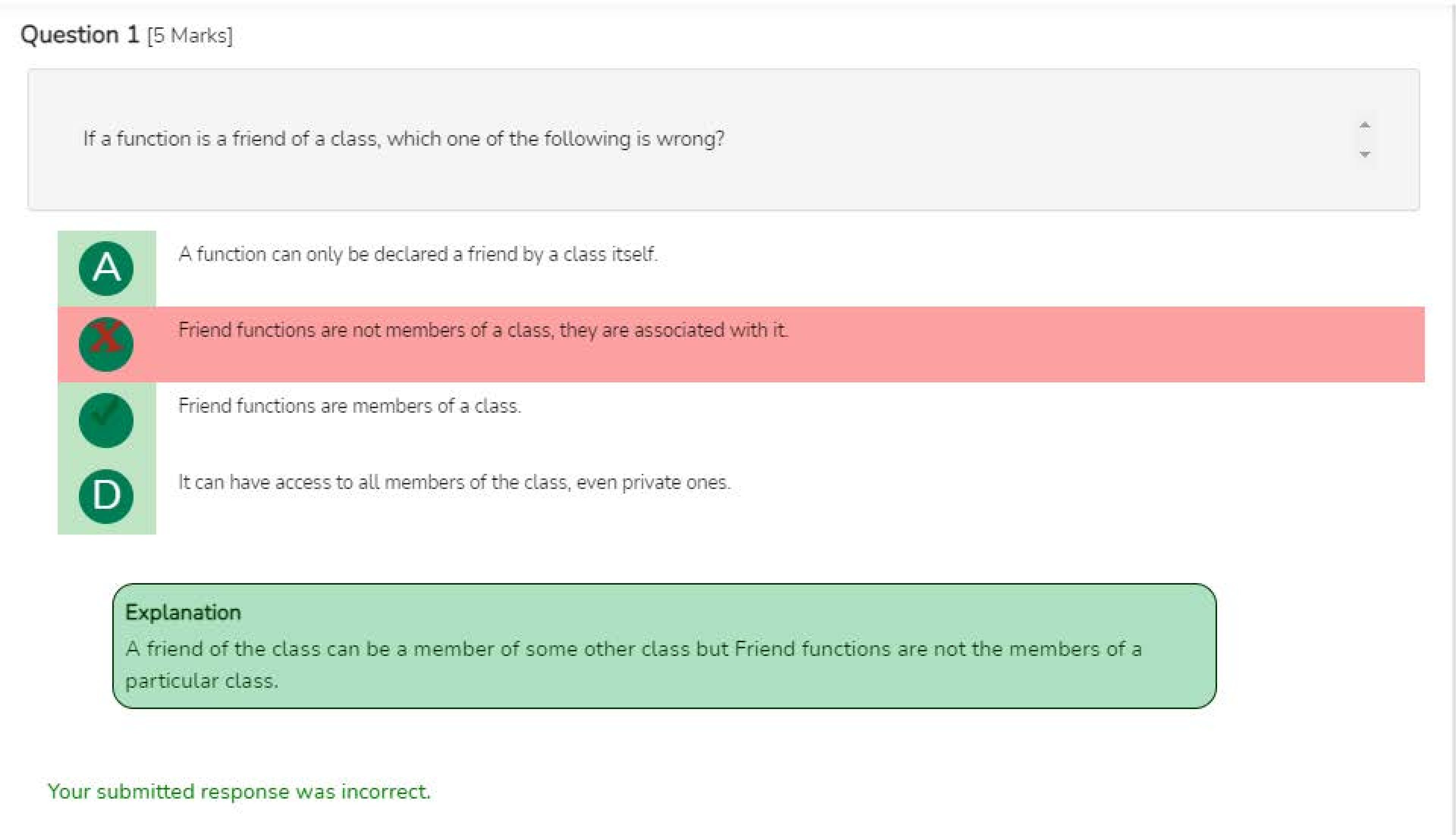


Question 1 [5 Marks]

If a function is a friend of a class, which one of the following is wrong?

A A function can only be declared a friend by a class itself.

✗ Friend functions are not members of a class, they are associated with it.

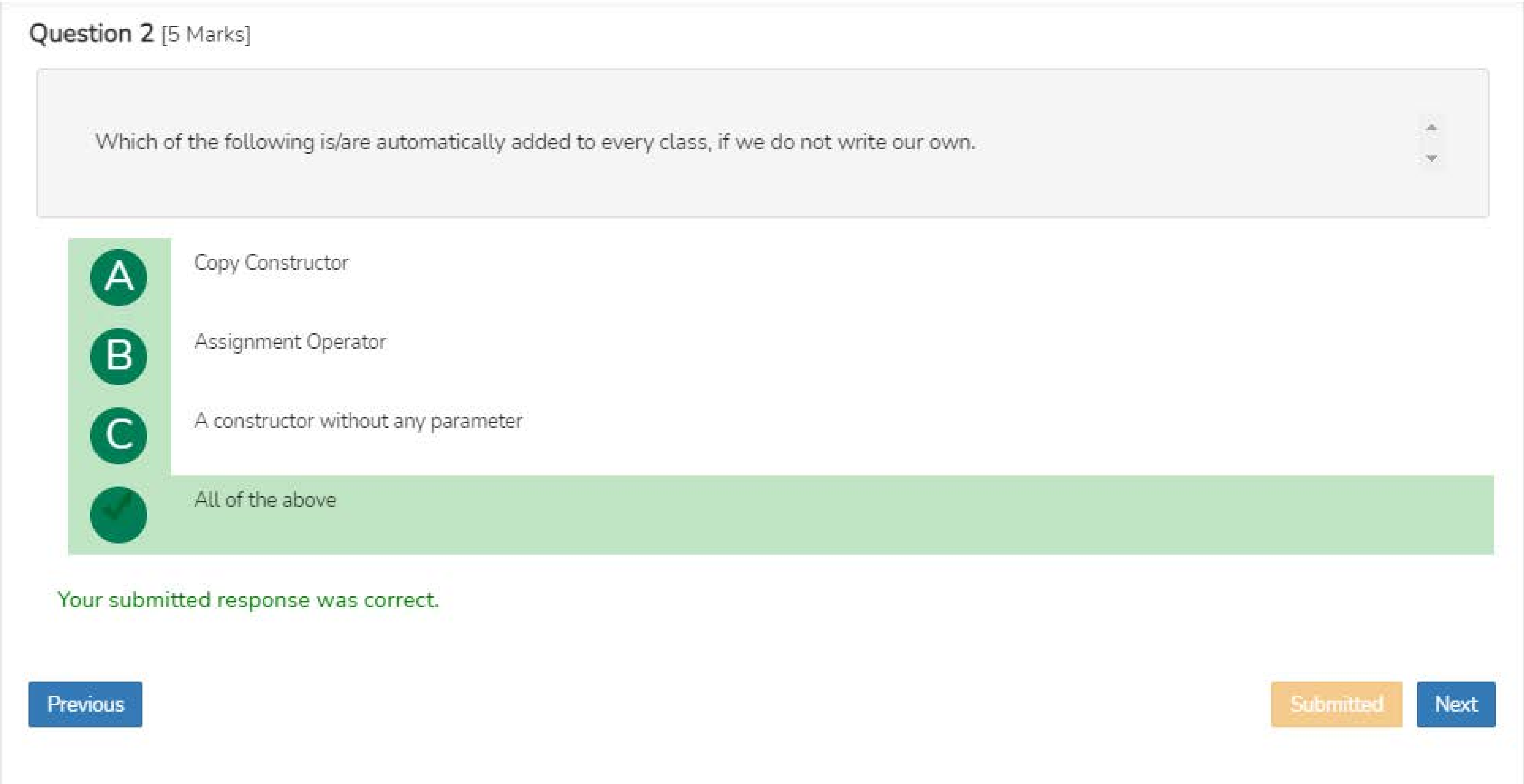
✓ Friend functions are members of a class.

D It can have access to all members of the class, even private ones.

Explanation

A friend of the class can be a member of some other class but Friend functions are not the members of a particular class.

Your submitted response was incorrect.



Question 2 [5 Marks]

Which of the following is/are automatically added to every class, if we do not write our own.

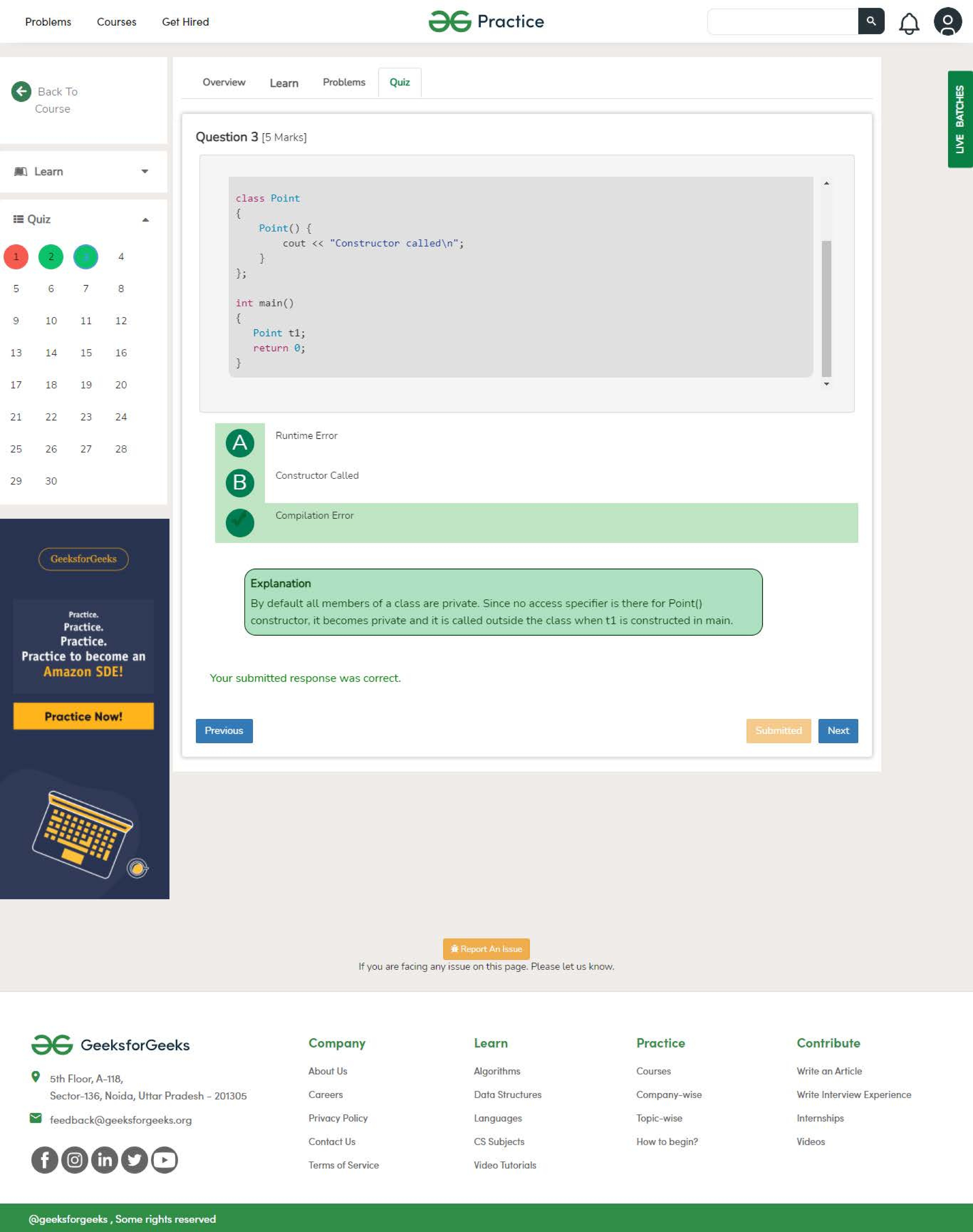
A Copy Constructor

B Assignment Operator

C A constructor without any parameter

✓ All of the above

Your submitted response was correct.



Question 3 [5 Marks]

```
class Point
{
    Point() {
        cout << "Constructor called\n";
    }
};

int main()
{
   Point t1;
   return 0;
}
```

A Runtime Error

B Constructor Called

✓ Compilation Error

Explanation

By default all members of a class are private. Since no access specifier is there for Point() constructor, it becomes private and it is called outside the class when t1 is constructed in main.

Your submitted response was correct.

If you are facing any issue on this page. Please let us know.

Back To Course

Learn

Quiz

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20
21 22 23 24
25 26 27 28
29 30

Overview Learn Problems Quiz

Question 4 [5 Marks]

```
{
        public:
                Point() { cout << "Normal Constructor called\n"; }
            Point(const Point &t) { cout << "Copy constructor called\n"; }
};

int main()
{
        Point *t1, *t2;
        t1 = new Point();
        t2 = new Point(*t1);
        Point t3 = *t1;
        Point t4;
        t4 = t3;
```

A

```
Normal Constructor called
Normal Constructor called
Normal Constructor called
Copy Constructor called
Copy Constructor called
Normal Constructor called
Copy Constructor called
```

B

```
Normal Constructor called
Copy Constructor called
Copy Constructor called
Normal Constructor called
Copy Constructor called
```

✓

```
Normal Constructor called
Copy Constructor called
Copy Constructor called
Normal Constructor called
```

Explanation

```
Point *t1, *t2;   // No constructor call
t1 = new Point(10, 15);  // Normal constructor call
t2 = new Point(*t1);   // Copy constructor call
Point t3 = *t1;  // Copy Constructor call
Point t4;   // Normal Constructor call
t4 = t3;   // Assignment operator call
```

Hence, the correct option is (C).

Your submitted response was correct.

Previous Submitted Next

Report An Issue

If you are facing any issue on this page. Please let us know.

Back To Course

Learn

Quiz

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20
21 22 23 24
25 26 27 28
29 30

Overview Learn Problems Quiz

Question 5 [5 Marks]

```
using namespace std;

class Test
{
        public:
                Test() { cout << "Constructor called"; }
};

int main()
{
    Test *t = (Test *) malloc(sizeof(Test));
    return 0;
}
```

A Constructor called

✓ Empty

C Compilation Error

✗ Runtime Error

Explanation
Unlike new, *malloc()* doesn't call the constructor. If we replace *malloc()* with new, the constructor is called. Malloc simply allocates memory equal to the requirement of the class and returns a void pointer.

Your submitted response was incorrect.

Previous Submitted Next

Report An Issue
If you are facing any issue on this page. Please let us know.

Back To Course

Learn

Quiz

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20
21 22 23 24
25 26 27 28
29 30

Overview Learn Problems Quiz

Question 6 [5 Marks]

Which of the following is true about constructors?

1. They cannot be virtual.
2. They cannot be private.
3. They are automatically called by *new* operator.

A All of the statements

✓ 1 and 3

C 1 and 2

D 2 and 3

Explanation

1. **TRUE**: Virtual constructors doesn't make sense as it is meaningless for the C++ compiler to create an object polymorphically.
2. **FALSE**: Constructors can be private. e.g. We make copy constructors private when we don't want to create copyable objects. The reason for not making copyable object could be to avoid shallow copy.
3. **TRUE**: Constructors are automatically called by new operator, we can in-fact pass parameters to constructors.

So, option (B) is correct.

Your submitted response was correct.

Previous Submitted Next

Report An Issue
If you are facing any issue on this page. Please let us know.

Back To Course

Learn

Quiz

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20
21 22 23 24
25 26 27 28
29 30

Question 7 [5 Marks]

Which of the following functions must use reference.

A Assignment operator function

✓ Copy constructor

C Destructor

✗ Parameterized Constructor

Explanation
A copy constructor is called when an object is passed by value. Copy constructor itself is a function. So if we pass the argument by value in a copy constructor, a call to copy constructor would be made to call copy constructor which becomes a non-terminating chain of calls. Therefore compiler doesn't allow parameters to be passed by value.

Your submitted response was incorrect.

Previous Submitted Next

Report An Issue
If you are facing any issue on this page. Please let us know.

Back To Course

Learn

Quiz

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20
21 22 23 24
25 26 27 28
29 30

Question 8 [5 Marks]

```
int &fun()
{
    static int x = 10;
    return x;
}

int main()
{
    fun() = 30;
    cout << fun();
    return 0;
}
```

A 10

B Compilation Error

✓ 30

D A value equal to address of x

Explanation

When a function returns by reference, it can be used as an *lvalue*. Since x is a static variable, it is shared among function calls and the initialization line *static int x = 10;* is executed only once. The function call fun() = 30, modifies x to 30. The next call *cout << fun();* returns the modified value.

Your submitted response was correct.

Previous Submitted Next

Report An Issue

If you are facing any issue on this page. Please let us know.

Back To Course

Learn

Quiz

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20
21 22 23 24
25 26 27 28
29 30

Overview Learn Problems Quiz

LIVE BATCHES

Question 9 [5 Marks]

```
class Test
{
    static int x;
    int *ptr;
    int y;
};

int main()
{
    Test t;
    cout << sizeof(t) << " ";
    cout << sizeof(Test *) << endl;
    return 0;
}
```

A 12 4

B 12 12

✓ 8 4

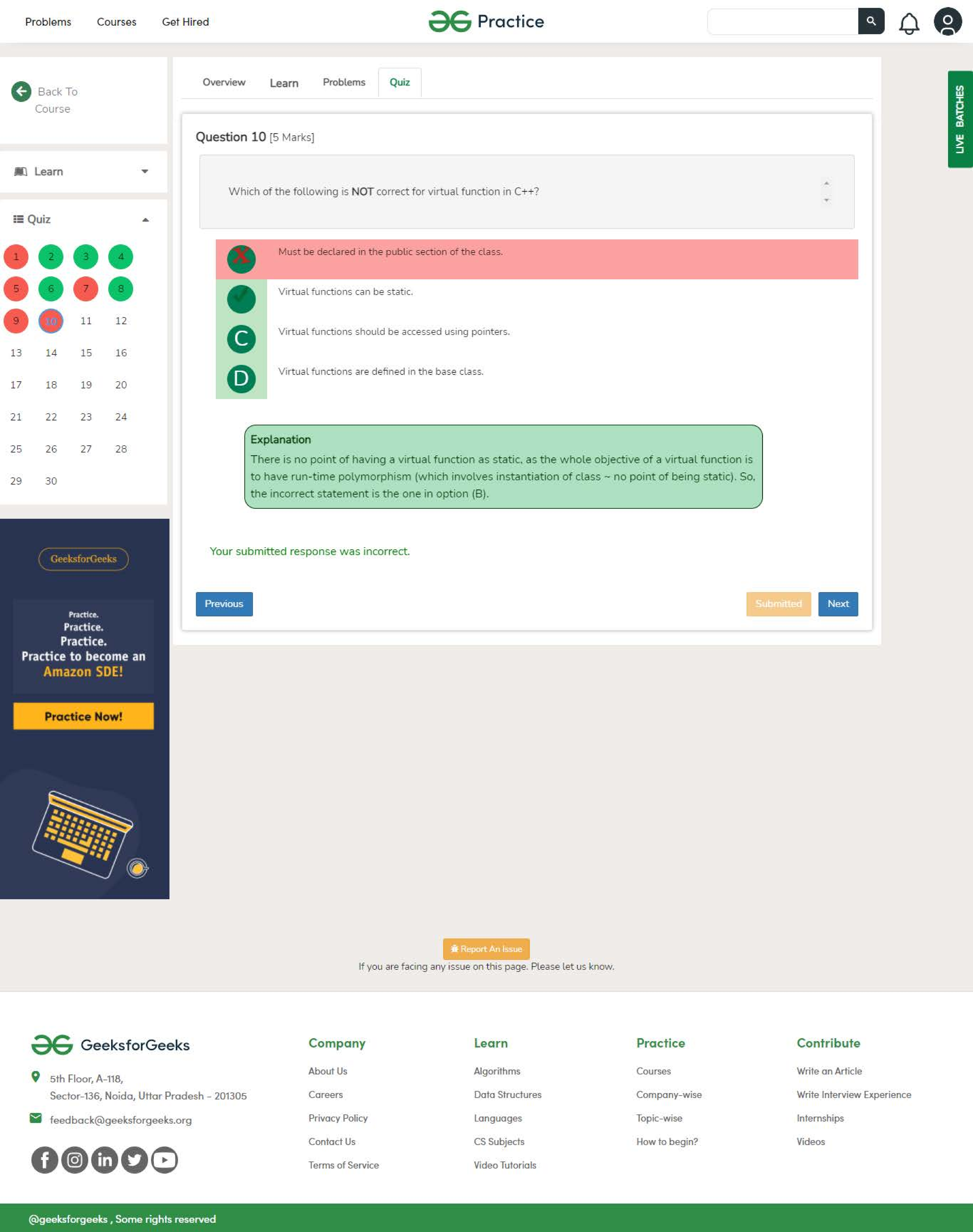
✗ 8 8

Explanation
For a compiler where pointers take 4 bytes, the statement *sizeof(Test *)* returns 4 (size of the pointer ptr). The statement *sizeof(t)* returns 8. Since static is not associated with each object of the class, we get (8 not 12).

Your submitted response was incorrect.

Previous Submitted Next

Report An Issue
If you are facing any issue on this page. Please let us know.



Back To Course

Learn

Quiz

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20
21 22 23 24
25 26 27 28
29 30

Overview Learn Problems Quiz

Question 10 [5 Marks]

Which of the following is **NOT** correct for virtual function in C++?

✗ Must be declared in the public section of the class.

✓ Virtual functions can be static.

C Virtual functions should be accessed using pointers.

D Virtual functions are defined in the base class.

Explanation

There is no point of having a virtual function as static, as the whole objective of a virtual function is to have run-time polymorphism (which involves instantiation of class ~ no point of being static). So, the incorrect statement is the one in option (B).

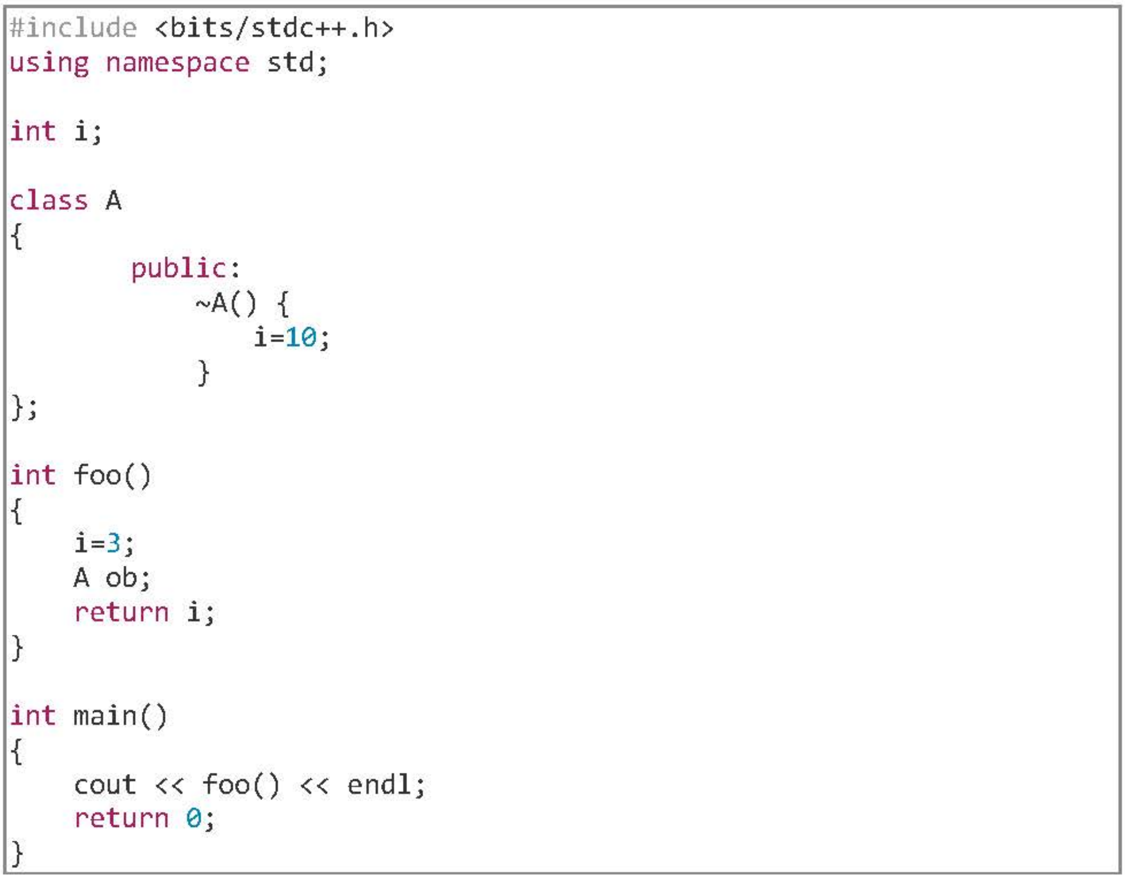
Your submitted response was incorrect.

Previous Submitted Next

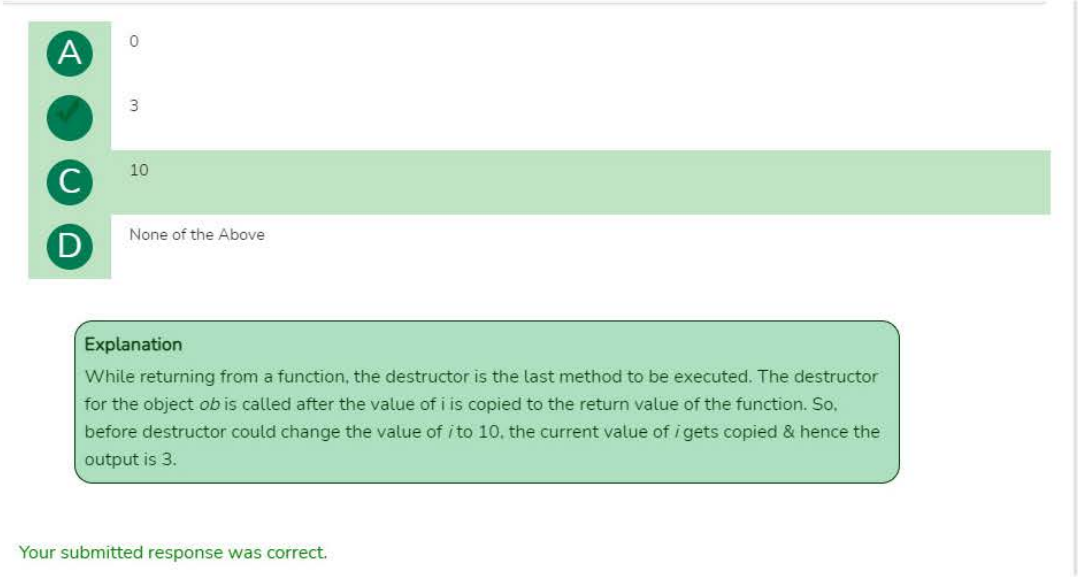
Report An Issue
If you are facing any issue on this page. Please let us know.

Question 11 [5 Marks]

Choose the correct output from the options given below:



```
#include <bits/stdc++.h>
using namespace std;

int i;

class A
{
        public:
            ~A() {
                i=10;
            }
};

int foo()
{
    i=3;
    A ob;
    return i;
}

int main()
{
    cout << foo() << endl;
    return 0;
}
```



A 0

✓ 3

C 10

D None of the Above

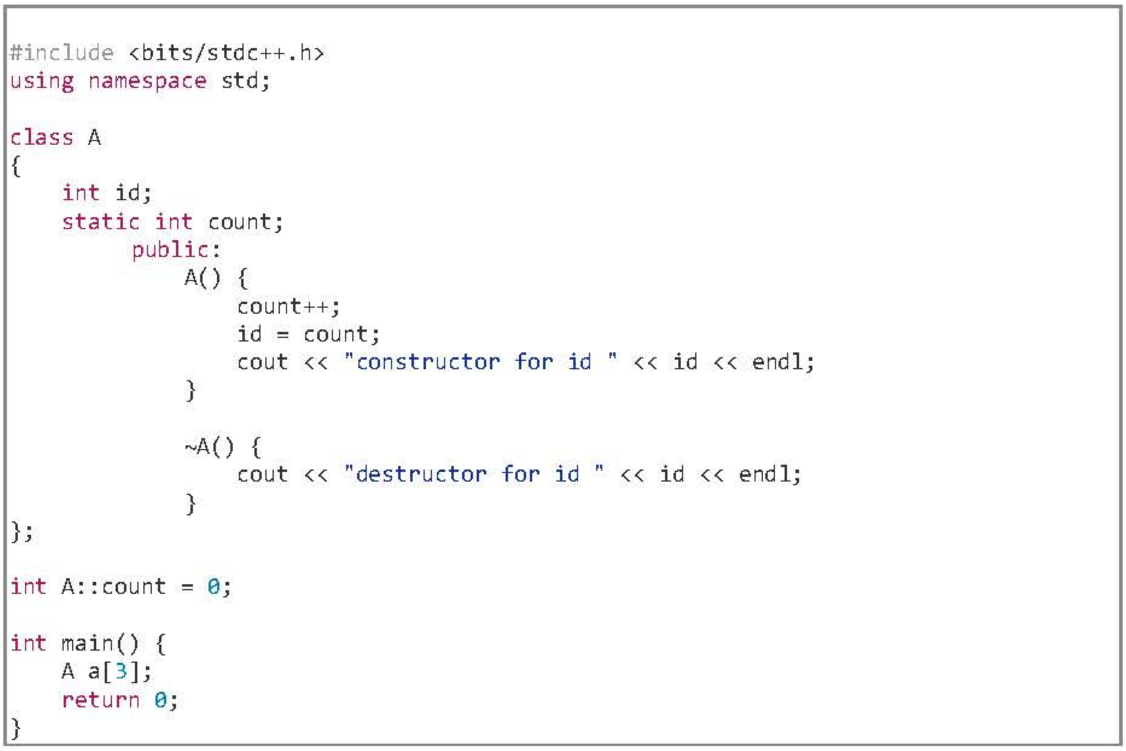
Explanation

While returning from a function, the destructor is the last method to be executed. The destructor for the object *ob* is called after the value of i is copied to the return value of the function. So, before destructor could change the value of *i* to 10, the current value of *i* gets copied & hence the output is 3.

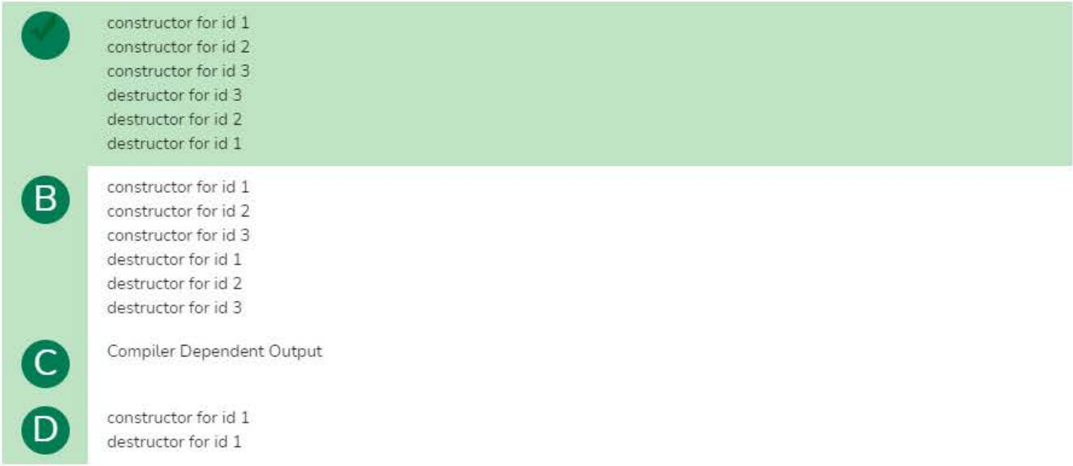
Your submitted response was correct.

Question 12 [5 Marks]

Choose the correct output from the options given below:



```
#include <bits/stdc++.h>
using namespace std;

class A
{
    int id;
    static int count;
          public:
              A() {
                  count++;
                  id = count;
                  cout << "constructor for id " << id << endl;
              }

              ~A() {
                  cout << "destructor for id " << id << endl;
              }
};

int A::count = 0;

int main() {
    A a[3];
    return 0;
}
```



✓

```
constructor for id 1
constructor for id 2
constructor for id 3
destructor for id 3
destructor for id 2
destructor for id 1
```

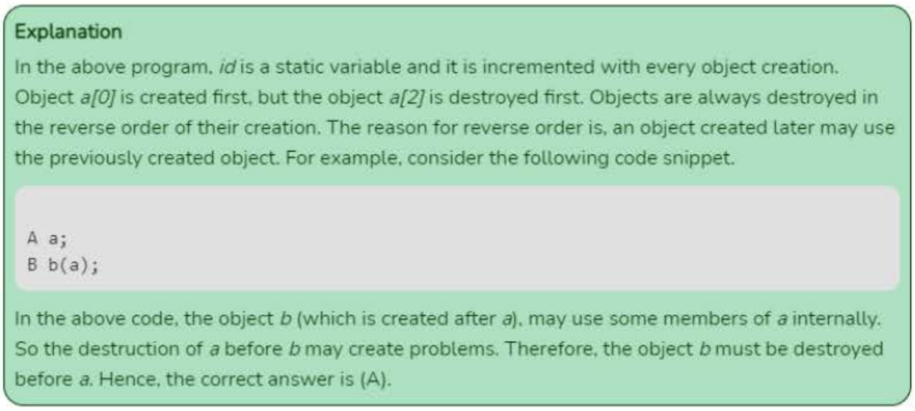
B

```
constructor for id 1
constructor for id 2
constructor for id 3
destructor for id 1
destructor for id 2
destructor for id 3
```

C Compiler Dependent Output

D

```
constructor for id 1
destructor for id 1
```



Explanation

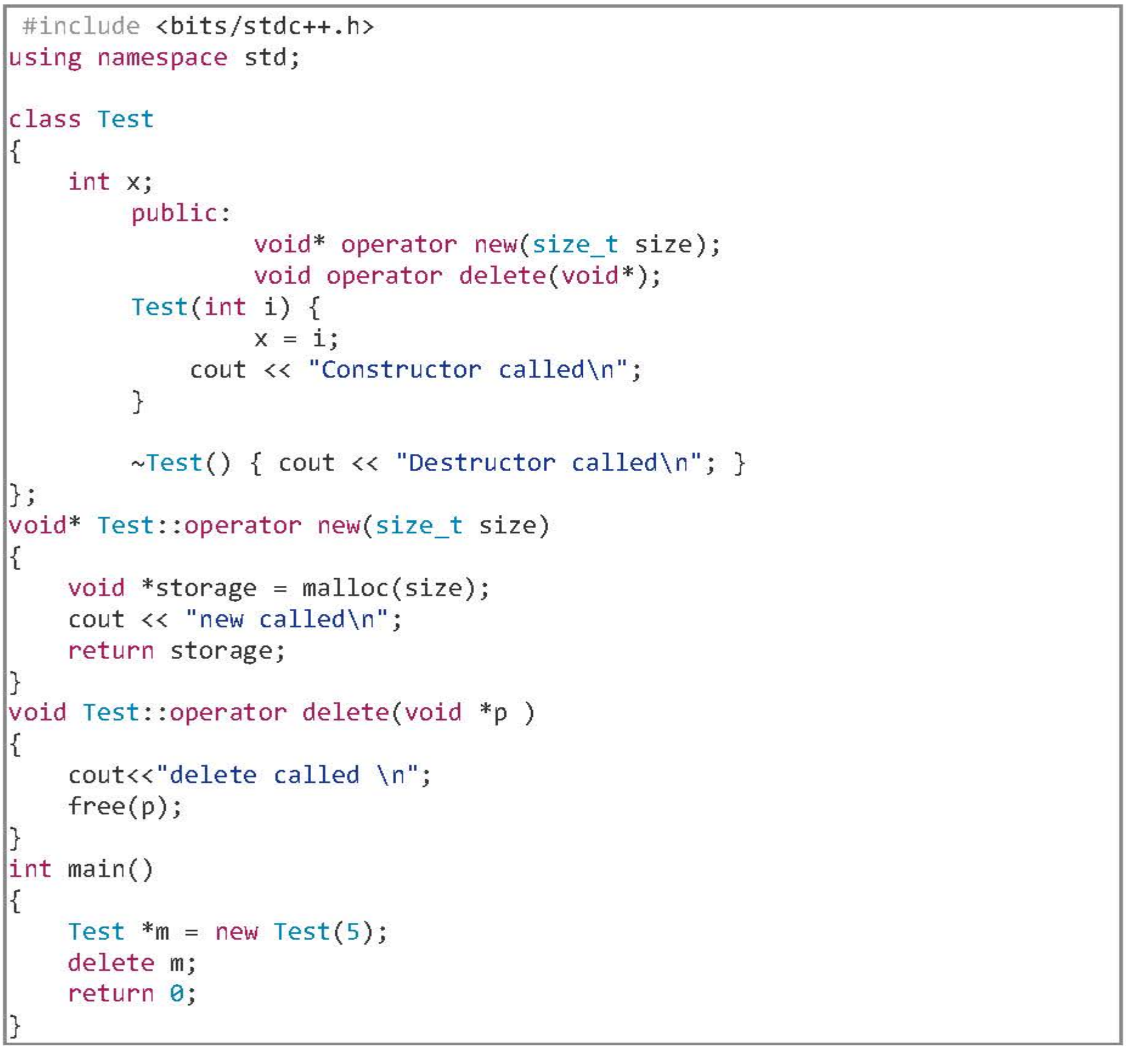
In the above program, *id* is a static variable and it is incremented with every object creation. Object *a[0]* is created first, but the object *a[2]* is destroyed first. Objects are always destroyed in the reverse order of their creation. The reason for reverse order is, an object created later may use the previously created object. For example, consider the following code snippet.

```
A a;
B b(a);
```

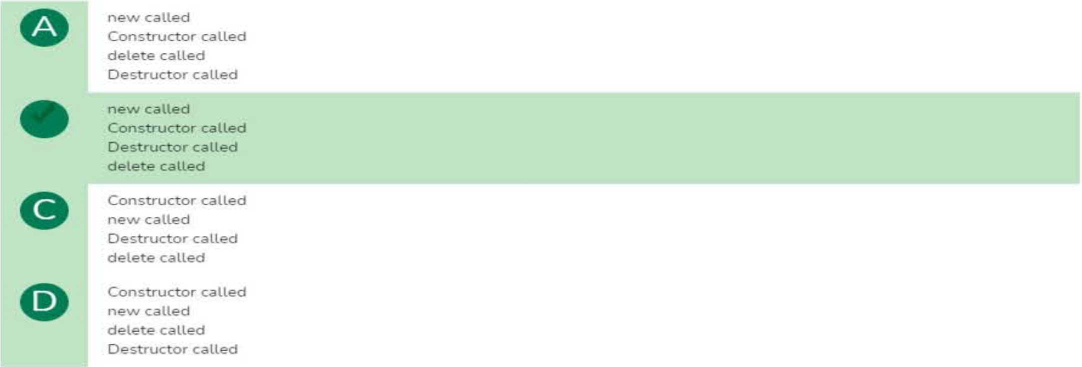
In the above code, the object *b* (which is created after *a*), may use some members of *a* internally. So the destruction of *a* before *b* may create problems. Therefore, the object *b* must be destroyed before *a*. Hence, the correct answer is (A).

Question 13 [5 Marks]

Choose the correct output from the options given below:



```
 #include <bits/stdc++.h>
using namespace std;

class Test
{
    int x;
        public:
                void* operator new(size_t size);
                void operator delete(void*);
        Test(int i) {
                x = i;
            cout << "Constructor called\n";
        }

        ~Test() { cout << "Destructor called\n"; }
};
void* Test::operator new(size_t size)
{
    void *storage = malloc(size);
    cout << "new called\n";
    return storage;
}
void Test::operator delete(void *p )
{
    cout<<"delete called \n";
    free(p);
}
int main()
{
    Test *m = new Test(5);
    delete m;
    return 0;
}
```

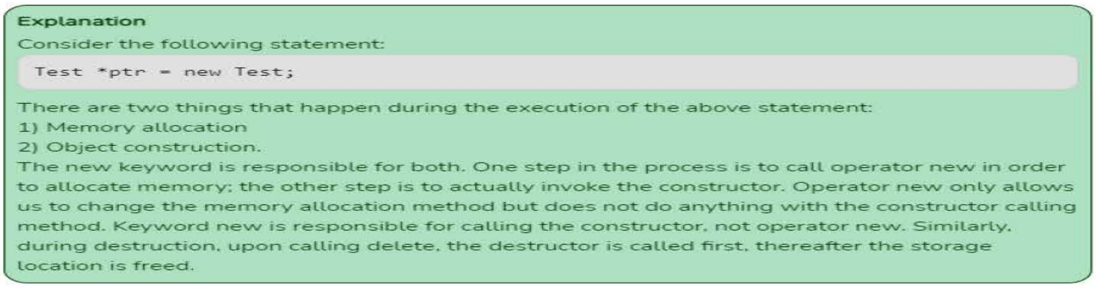


A
```
new called
Constructor called
delete called
Destructor called
```

✓
```
new called
Constructor called
Destructor called
delete called
```

C
```
Constructor called
new called
Destructor called
delete called
```

D
```
Constructor called
new called
delete called
Destructor called
```



Explanation

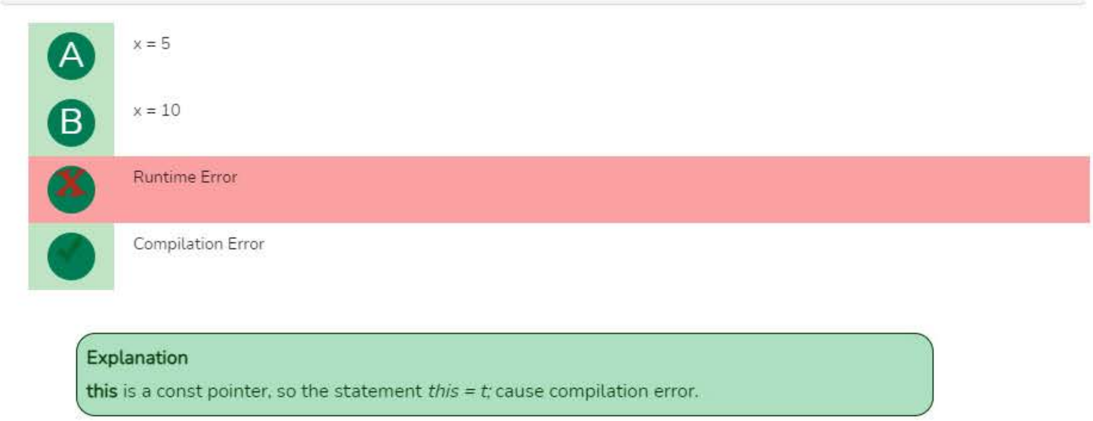
Consider the following statement:

```
Test *ptr = new Test;
```

There are two things that happen during the execution of the above statement:
1) Memory allocation
2) Object construction.
The new keyword is responsible for both. One step in the process is to call operator new in order to allocate memory; the other step is to actually invoke the constructor. Operator new only allows us to change the memory allocation method but does not do anything with the constructor calling method. Keyword new is responsible for calling the constructor, not operator new. Similarly, during destruction, upon calling delete, the destructor is called first, thereafter the storage location is freed.

Question 14 [5 Marks]
Choose the correct output from the options given below:

```
#include <bits/stdc++.h>
using namespace std;

class Test
{
  private:
         int x;
  public:
         Test(int x = 0) { this->x = x; }
         void change(Test *t) { this = t; }
         void print() { cout << "x = " << x << endl; }
};

int main()
{
  Test obj(5);
  Test *ptr = new Test(10);
  obj.change(ptr);
  obj.print();
  return 0;
}
```



A x = 5

B x = 10

✗ Runtime Error

✓ Compilation Error

Explanation
this is a const pointer, so the statement *this = t;* cause compilation error.

Overview Learn Problems Quiz

Back To Course

Learn

Quiz

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20
21 22 23 24
25 26 27 28
29 30

Question 15 [5 Marks]

Which of the following is true about **this** pointer?

A It is passed as a hidden argument to all function calls.

✓ It is passed as a hidden argument to all non-static function calls.

C It is passed as a hidden argument to all static functions.

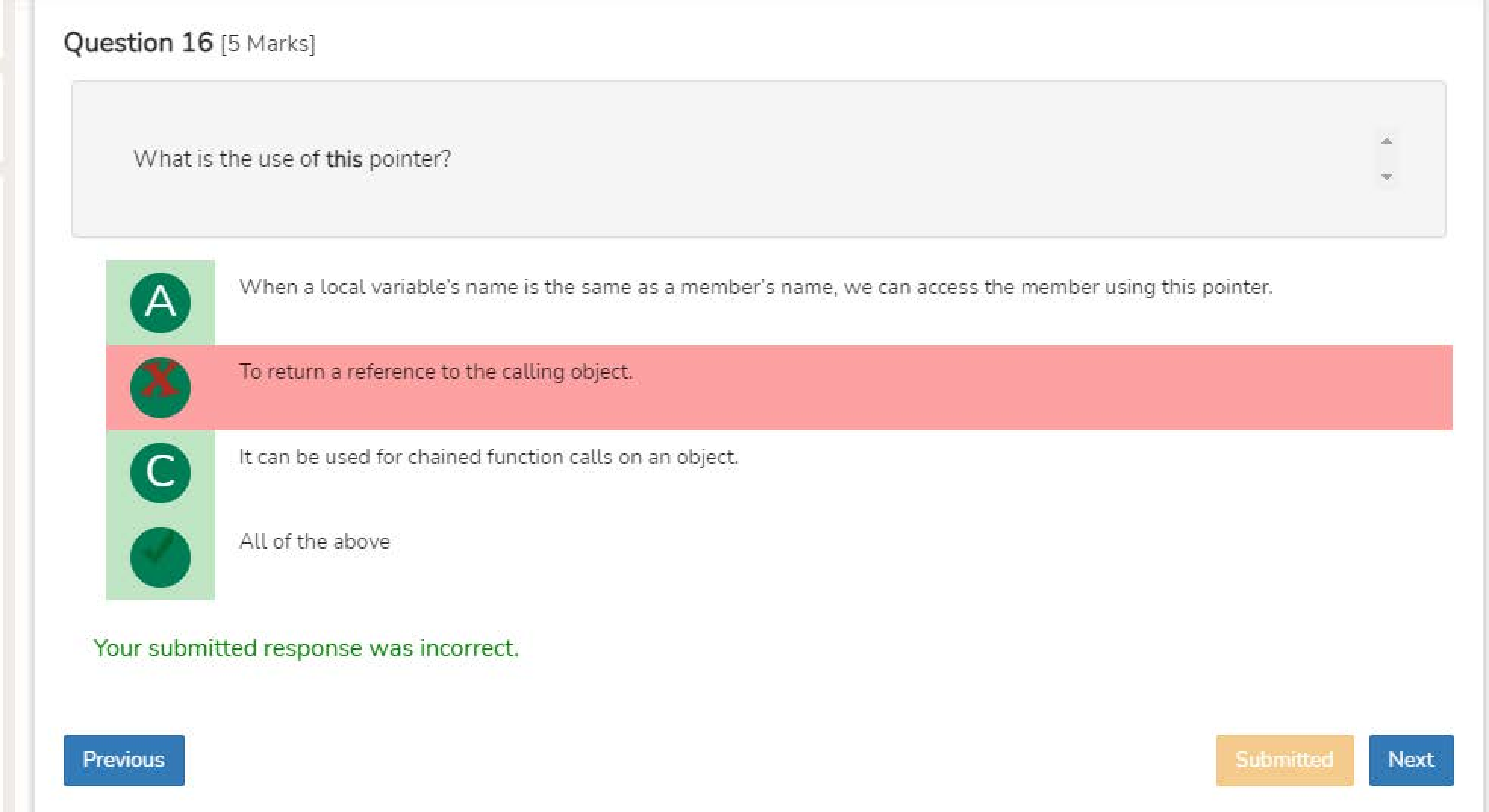
D None of the above

Explanation
The *this* pointer is passed as a hidden argument to all non-static member function calls and is available as a local variable within the body of all non-static functions. *this* pointer is a constant pointer that holds the memory address of the current object. *this* pointer is not available in static member functions as static member functions can be called without any object (with class-name).

Your submitted response was correct.

Previous Submitted Next

Report An Issue
If you are facing any issue on this page. Please let us know.



Question 16 [5 Marks]

What is the use of **this** pointer?

A When a local variable's name is the same as a member's name, we can access the member using this pointer.

✗ To return a reference to the calling object.

C It can be used for chained function calls on an object.

✓ All of the above

Your submitted response was incorrect.

Back To Course

Learn

Quiz

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20
21 22 23 24
25 26 27 28
29 30

LIVE BATCHES

Overview Learn Problems Quiz

Question 17 [5 Marks]

```
#include <bits/stdc++.h>
using namespace std;

int fun(int x = 0, int y = 0, int z)
{
        return (x + y + z);
}

int main()
{
   cout << fun(10);
   return 0;
}
```

A 10

✗ 0

C 20

✓ Compilation Error

Explanation

All default arguments must be the rightmost arguments. Hence, the above program produces a compilation error. The correct version of the above program works fine and produces 10 as output:

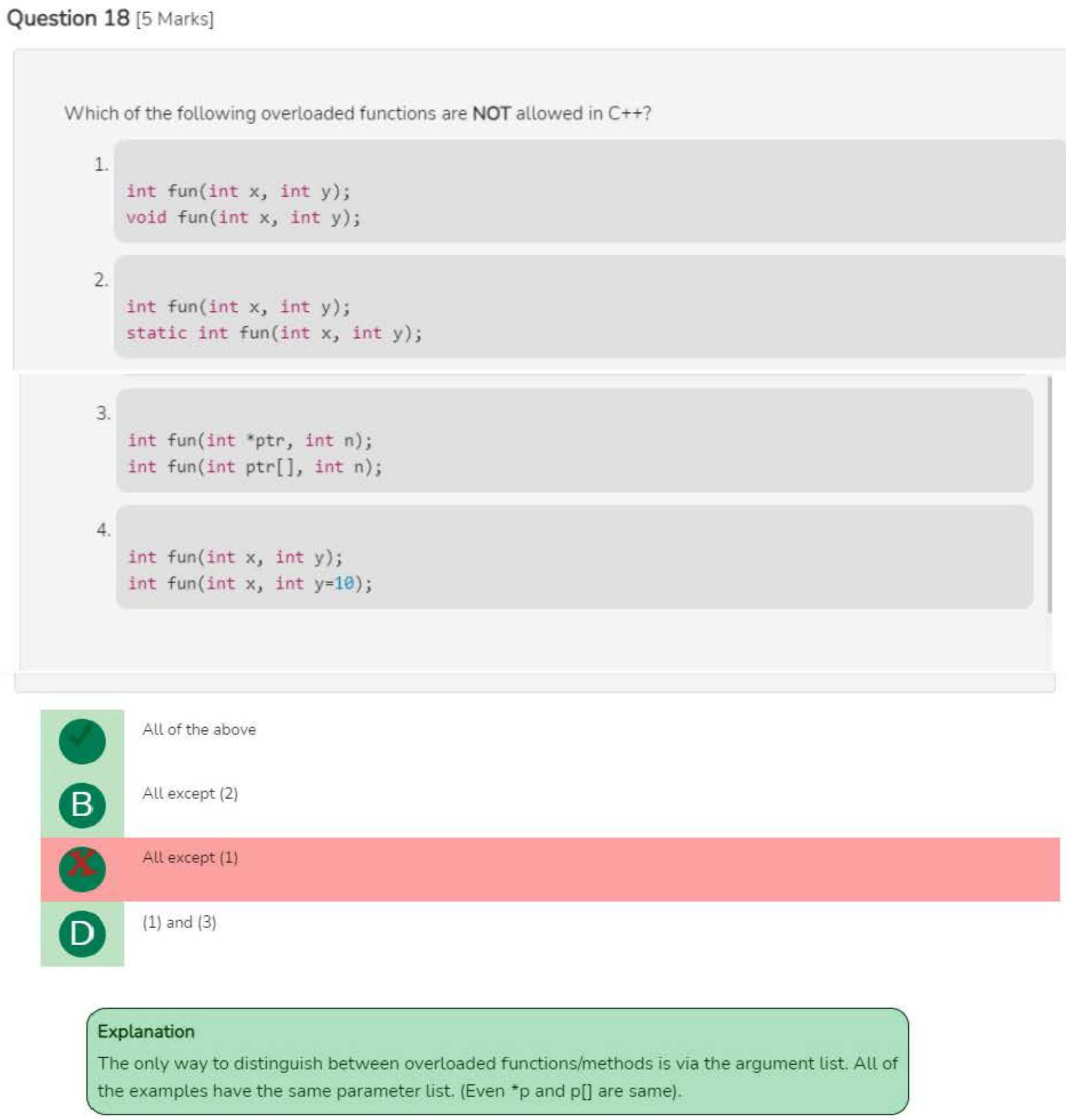
```
#include <bits/stdc++.h>
using namespace std;

int fun(int x, int y = 0, int z = 0)
{
        return (x + y + z);
}

int main()
{
   cout << fun(10);
   return 0;
}
```

Your submitted response was incorrect.

Previous Submitted Next

Report An Issue

If you are facing any issue on this page. Please let us know.



Question 18 [5 Marks]

Which of the following overloaded functions are **NOT** allowed in C++?

1.
```
int fun(int x, int y);
void fun(int x, int y);
```

2.
```
int fun(int x, int y);
static int fun(int x, int y);
```

3.
```
int fun(int *ptr, int n);
int fun(int ptr[], int n);
```

4.
```
int fun(int x, int y);
int fun(int x, int y=10);
```

✓ All of the above

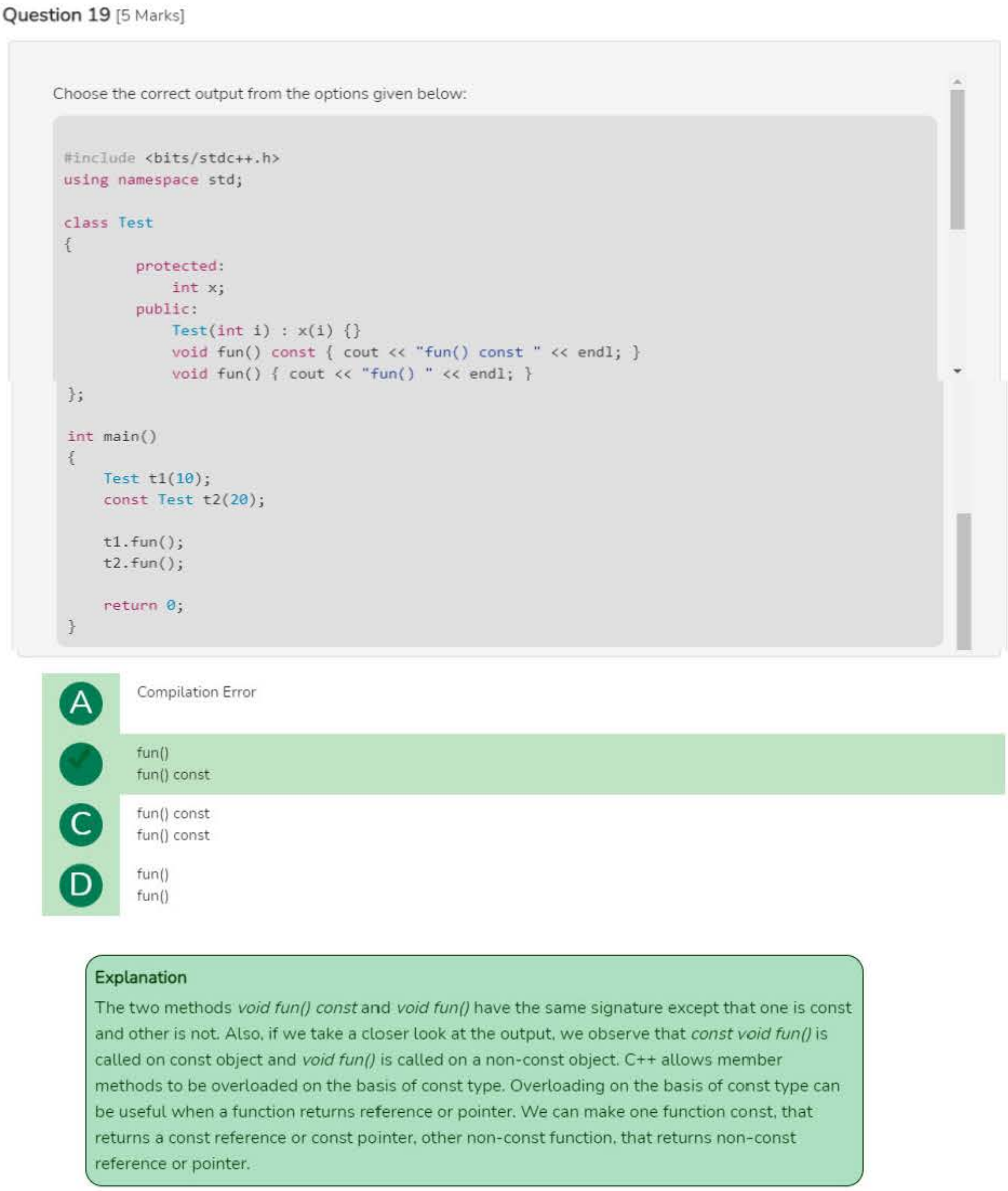
B All except (2)

✗ All except (1)

D (1) and (3)

Explanation

The only way to distinguish between overloaded functions/methods is via the argument list. All of the examples have the same parameter list. (Even *p and p[] are same).



Question 19 [5 Marks]

Choose the correct output from the options given below:

```
#include <bits/stdc++.h>
using namespace std;

class Test
{
        protected:
            int x;
        public:
            Test(int i) : x(i) {}
            void fun() const { cout << "fun() const " << endl; }
            void fun() { cout << "fun() " << endl; }
};

int main()
{
    Test t1(10);
    const Test t2(20);

    t1.fun();
    t2.fun();

    return 0;
}
```

A Compilation Error

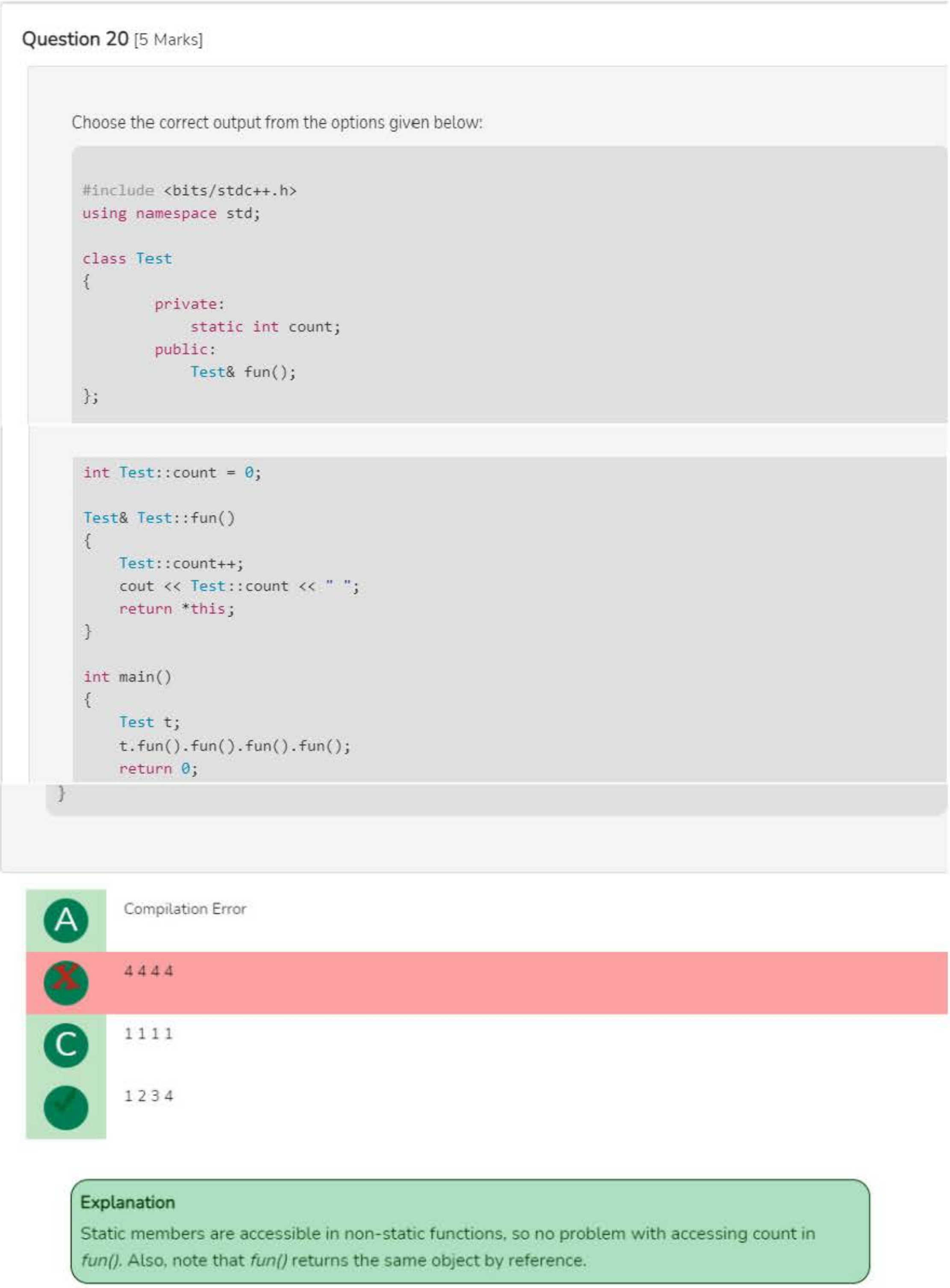
✓ fun()
fun() const

C fun() const
fun() const

D fun()
fun()

Explanation

The two methods *void fun() const* and *void fun()* have the same signature except that one is const and other is not. Also, if we take a closer look at the output, we observe that *const void fun()* is called on const object and *void fun()* is called on a non-const object. C++ allows member methods to be overloaded on the basis of const type. Overloading on the basis of const type can be useful when a function returns reference or pointer. We can make one function const, that returns a const reference or const pointer, other non-const function, that returns non-const reference or pointer.



Question 20 [5 Marks]

Choose the correct output from the options given below:

```
#include <bits/stdc++.h>
using namespace std;

class Test
{
        private:
            static int count;
        public:
            Test& fun();
};
```

```
int Test::count = 0;

Test& Test::fun()
{
    Test::count++;
    cout << Test::count << " ";
    return *this;
}

int main()
{
    Test t;
    t.fun().fun().fun().fun();
    return 0;
}
```

A Compilation Error

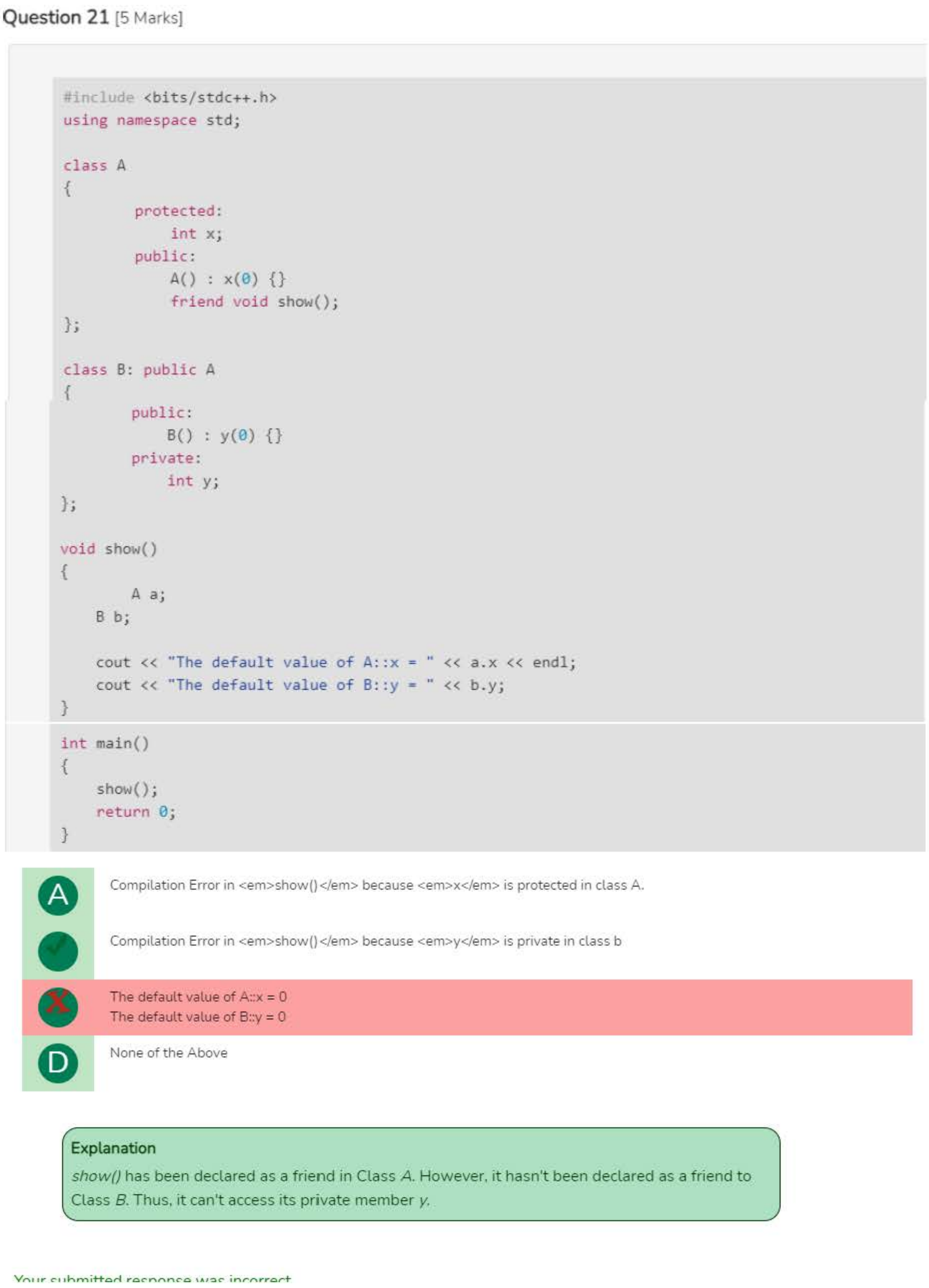
✗ 4 4 4 4

C 1 1 1 1

✓ 1 2 3 4

Explanation

Static members are accessible in non-static functions, so no problem with accessing count in *fun()*. Also, note that *fun()* returns the same object by reference.



Question 21 [5 Marks]

```
#include <bits/stdc++.h>
using namespace std;

class A
{
        protected:
            int x;
        public:
            A() : x(0) {}
            friend void show();
};

class B: public A
{
        public:
            B() : y(0) {}
        private:
            int y;
};

void show()
{
        A a;
    B b;

    cout << "The default value of A::x = " << a.x << endl;
    cout << "The default value of B::y = " << b.y;
}

int main()
{
    show();
    return 0;
}
```

A Compilation Error in <em>show()</em> because <em>x</em> is protected in class A.

✓ Compilation Error in <em>show()</em> because <em>y</em> is private in class b

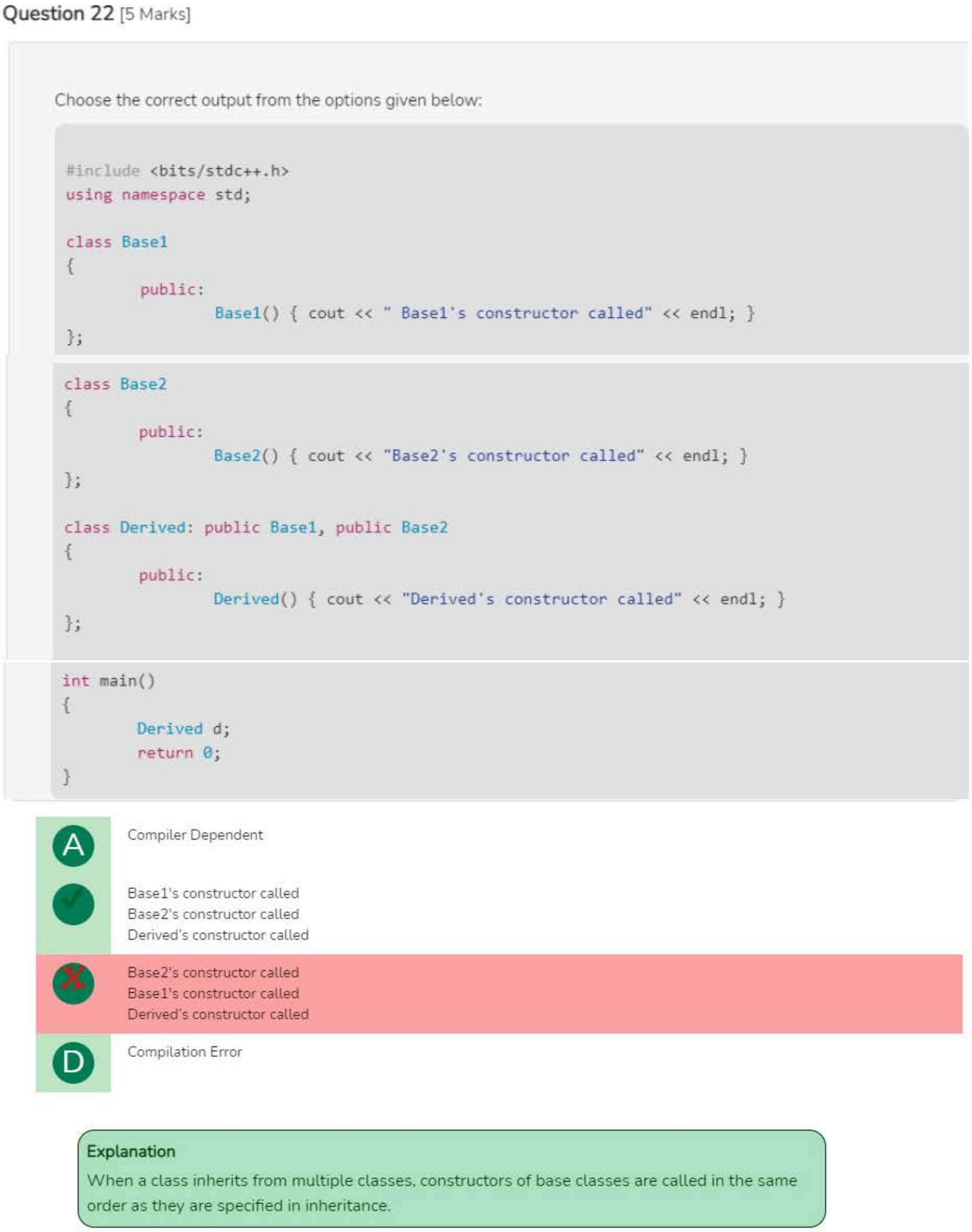
✗ The default value of A::x = 0
The default value of B::y = 0

D None of the Above

Explanation

show() has been declared as a friend in Class *A*. However, it hasn't been declared as a friend to Class *B*. Thus, it can't access its private member *y*.

Your submitted response was incorrect



Question 22 [5 Marks]

Choose the correct output from the options given below:

```
#include <bits/stdc++.h>
using namespace std;

class Base1
{
        public:
                Base1() { cout << " Base1's constructor called" << endl; }
};

class Base2
{
        public:
                Base2() { cout << "Base2's constructor called" << endl; }
};

class Derived: public Base1, public Base2
{
        public:
                Derived() { cout << "Derived's constructor called" << endl; }
};

int main()
{
        Derived d;
        return 0;
}
```

A Compiler Dependent

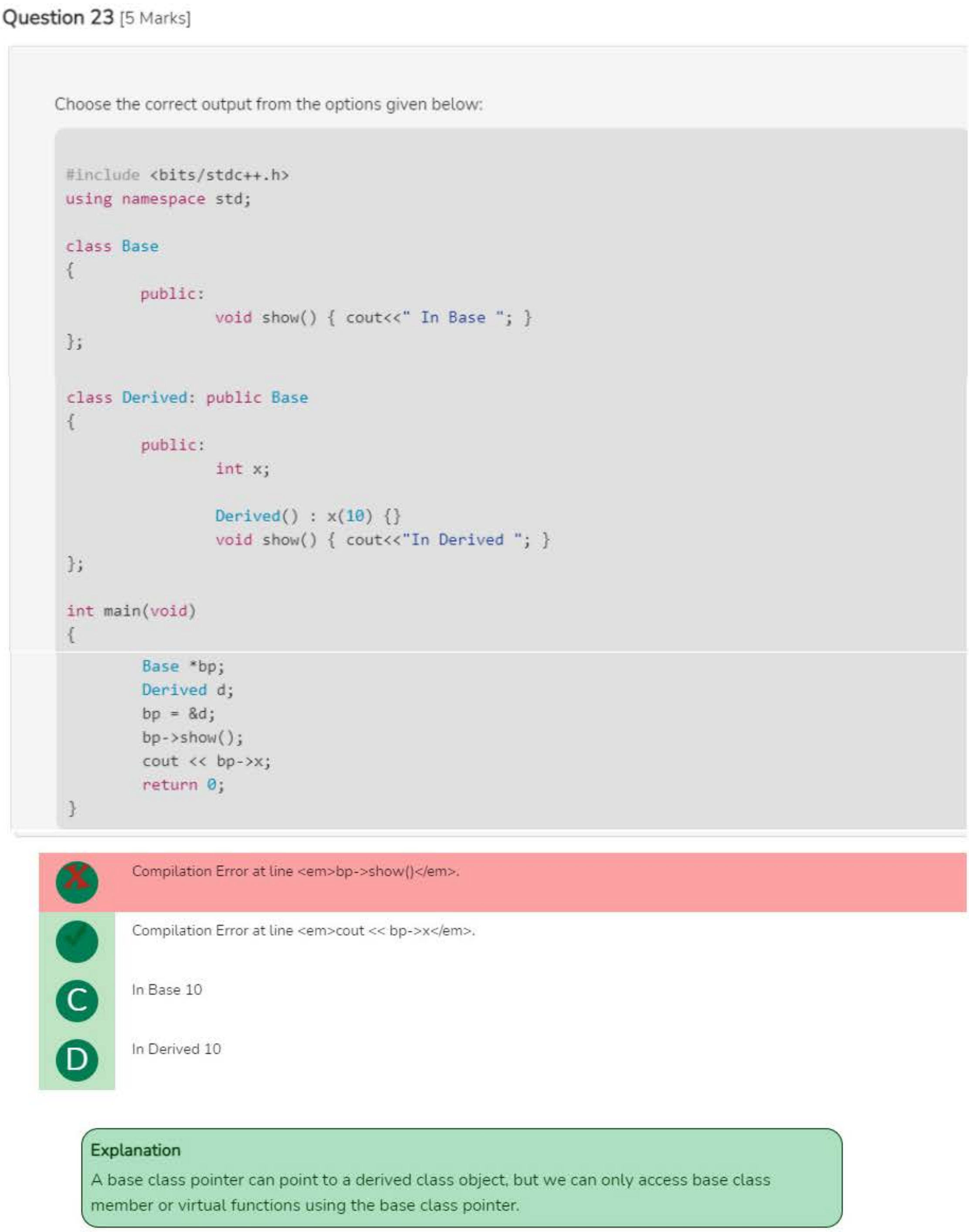
✓ Base1's constructor called
Base2's constructor called
Derived's constructor called

✗ Base2's constructor called
Base1's constructor called
Derived's constructor called

D Compilation Error

Explanation

When a class inherits from multiple classes, constructors of base classes are called in the same order as they are specified in inheritance.



Question 23 [5 Marks]

Choose the correct output from the options given below:

```
#include <bits/stdc++.h>
using namespace std;

class Base
{
        public:
                void show() { cout<<" In Base "; }
};

class Derived: public Base
{
        public:
                int x;

                Derived() : x(10) {}
                void show() { cout<<"In Derived "; }
};

int main(void)
{
        Base *bp;
        Derived d;
        bp = &d;
        bp->show();
        cout << bp->x;
        return 0;
}
```

✗ Compilation Error at line <em>bp->show()</em>.

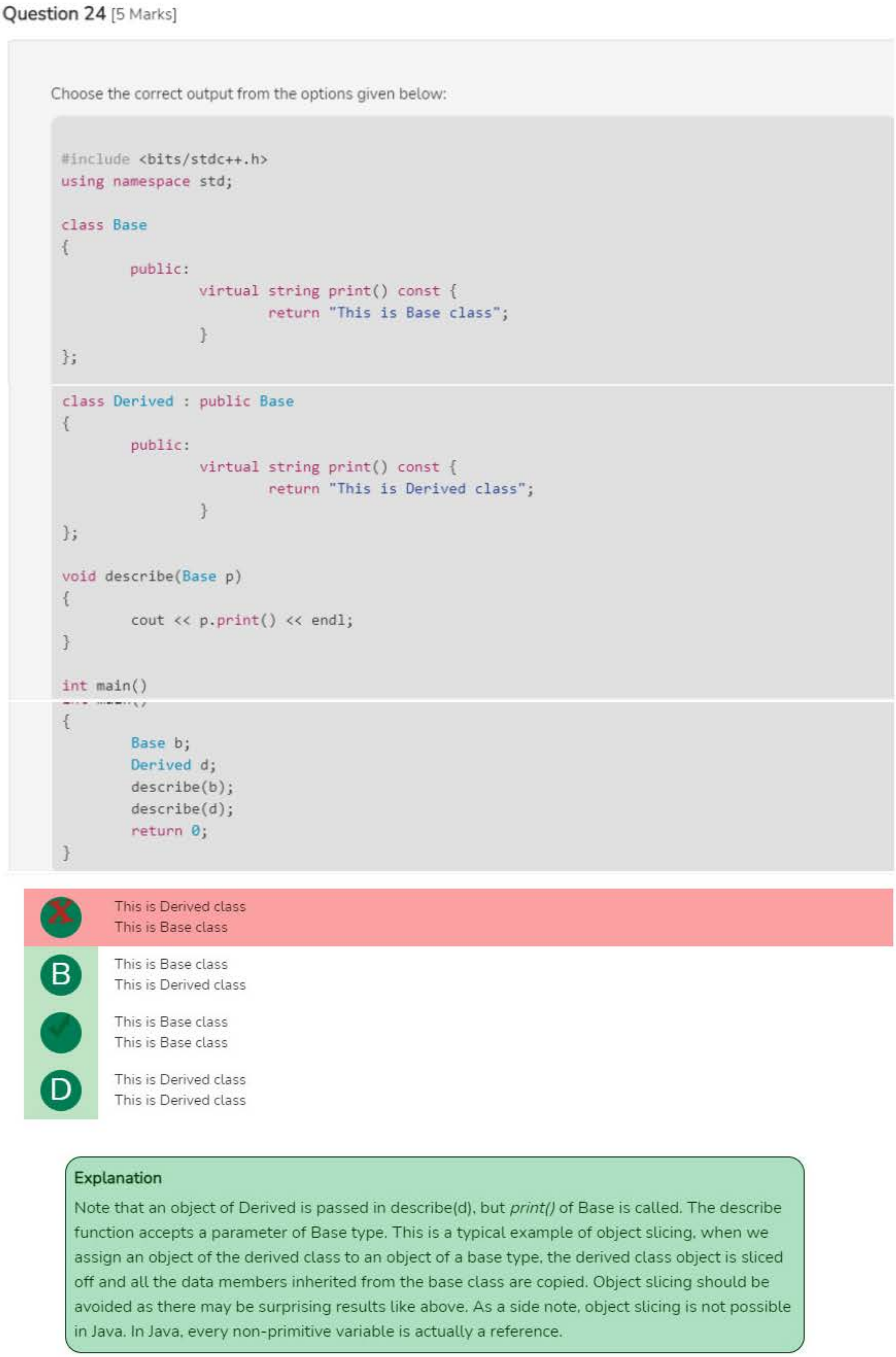
✓ Compilation Error at line <em>cout << bp->x</em>.

C In Base 10

D In Derived 10

Explanation

A base class pointer can point to a derived class object, but we can only access base class member or virtual functions using the base class pointer.



Question 24 [5 Marks]

Choose the correct output from the options given below:

```
#include <bits/stdc++.h>
using namespace std;

class Base
{
        public:
                virtual string print() const {
                        return "This is Base class";
                }
};

class Derived : public Base
{
        public:
                virtual string print() const {
                        return "This is Derived class";
                }
};

void describe(Base p)
{
        cout << p.print() << endl;
}

int main()
{
        Base b;
        Derived d;
        describe(b);
        describe(d);
        return 0;
}
```

✗ This is Derived class
This is Base class

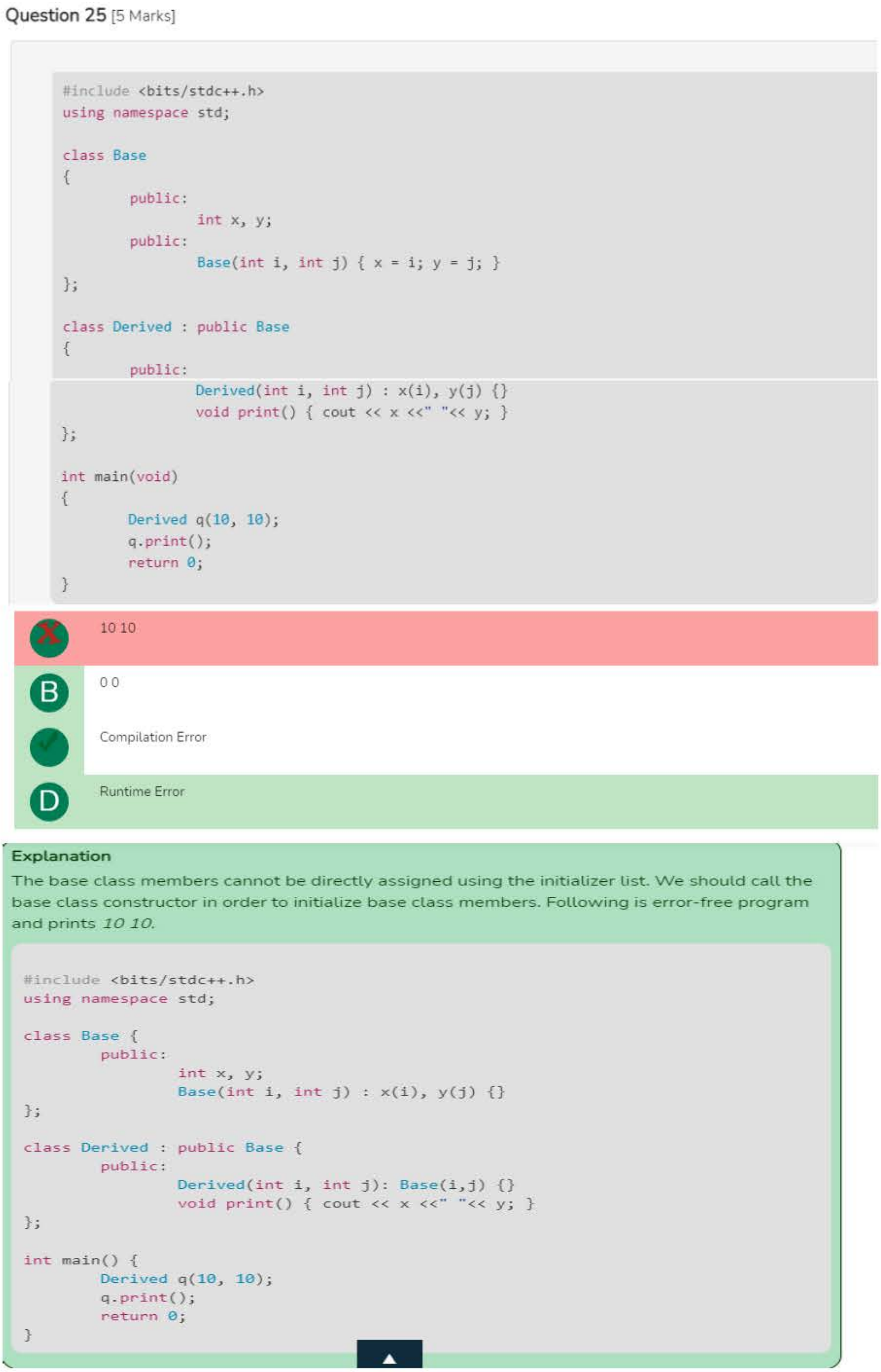
B This is Base class
This is Derived class

✓ This is Base class
This is Base class

D This is Derived class
This is Derived class

Explanation

Note that an object of Derived is passed in describe(d), but *print()* of Base is called. The describe function accepts a parameter of Base type. This is a typical example of object slicing, when we assign an object of the derived class to an object of a base type, the derived class object is sliced off and all the data members inherited from the base class are copied. Object slicing should be avoided as there may be surprising results like above. As a side note, object slicing is not possible in Java. In Java, every non-primitive variable is actually a reference.



Question 25 [5 Marks]

```
#include <bits/stdc++.h>
using namespace std;

class Base
{
        public:
                int x, y;
        public:
                Base(int i, int j) { x = i; y = j; }
};

class Derived : public Base
{
        public:
                Derived(int i, int j) : x(i), y(j) {}
                void print() { cout << x <<" "<< y; }
};

int main(void)
{
        Derived q(10, 10);
        q.print();
        return 0;
}
```

✗ 10 10

B 0 0

✓ Compilation Error

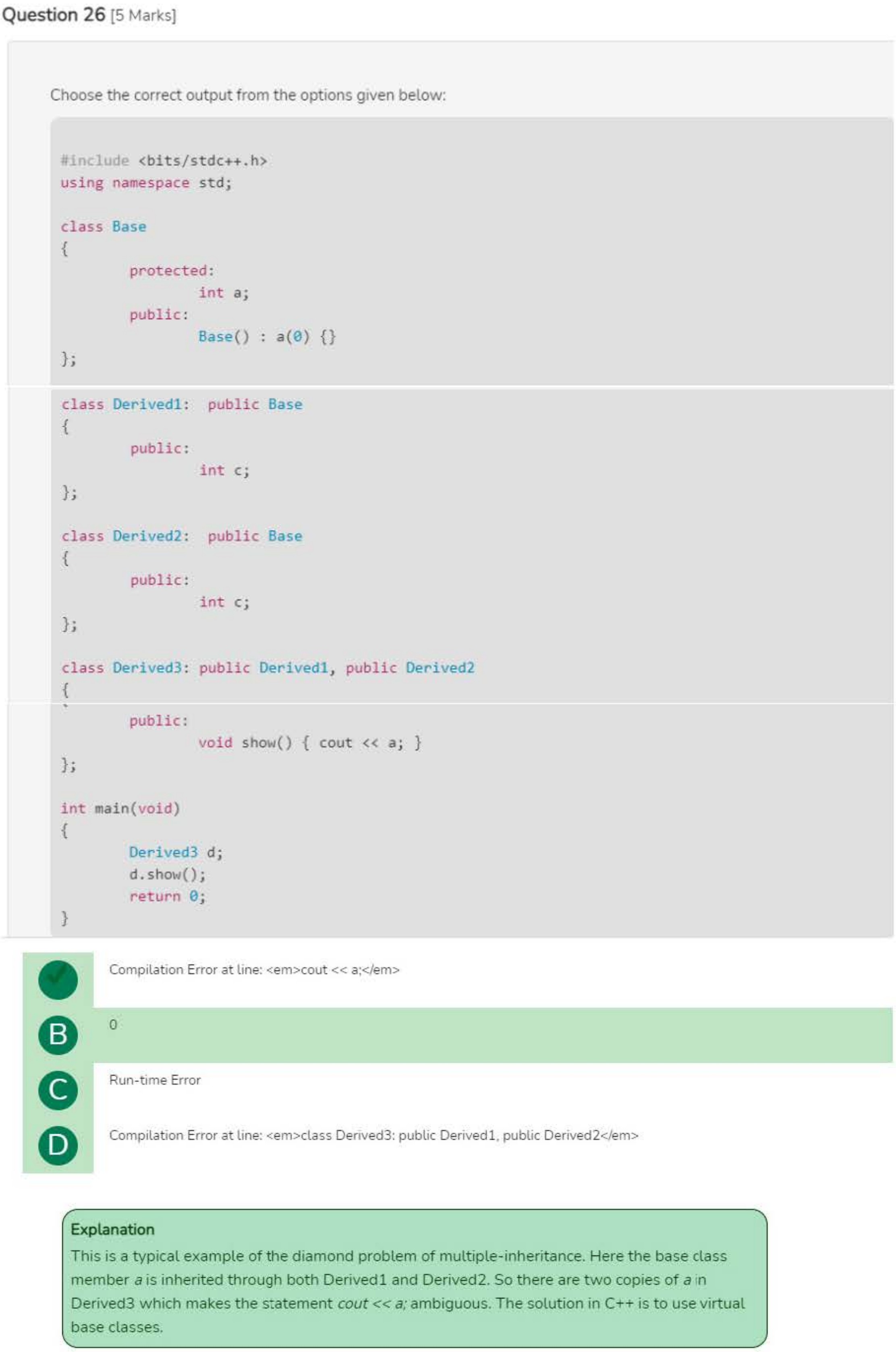
D Runtime Error

Explanation

The base class members cannot be directly assigned using the initializer list. We should call the base class constructor in order to initialize base class members. Following is error-free program and prints *10 10.*

```
#include <bits/stdc++.h>
using namespace std;

class Base {
        public:
                int x, y;
                Base(int i, int j) : x(i), y(j) {}
};

class Derived : public Base {
        public:
                Derived(int i, int j): Base(i,j) {}
                void print() { cout << x <<" "<< y; }
};

int main() {
        Derived q(10, 10);
        q.print();
        return 0;
}
```

▲



Question 26 [5 Marks]

Choose the correct output from the options given below:

```
#include <bits/stdc++.h>
using namespace std;

class Base
{
        protected:
                int a;
        public:
                Base() : a(0) {}
};

class Derived1:  public Base
{
        public:
                int c;
};

class Derived2:  public Base
{
        public:
                int c;
};

class Derived3: public Derived1, public Derived2
{
        public:
                void show() { cout << a; }
};

int main(void)
{
        Derived3 d;
        d.show();
        return 0;
}
```

✓ Compilation Error at line: <em>cout << a;</em>

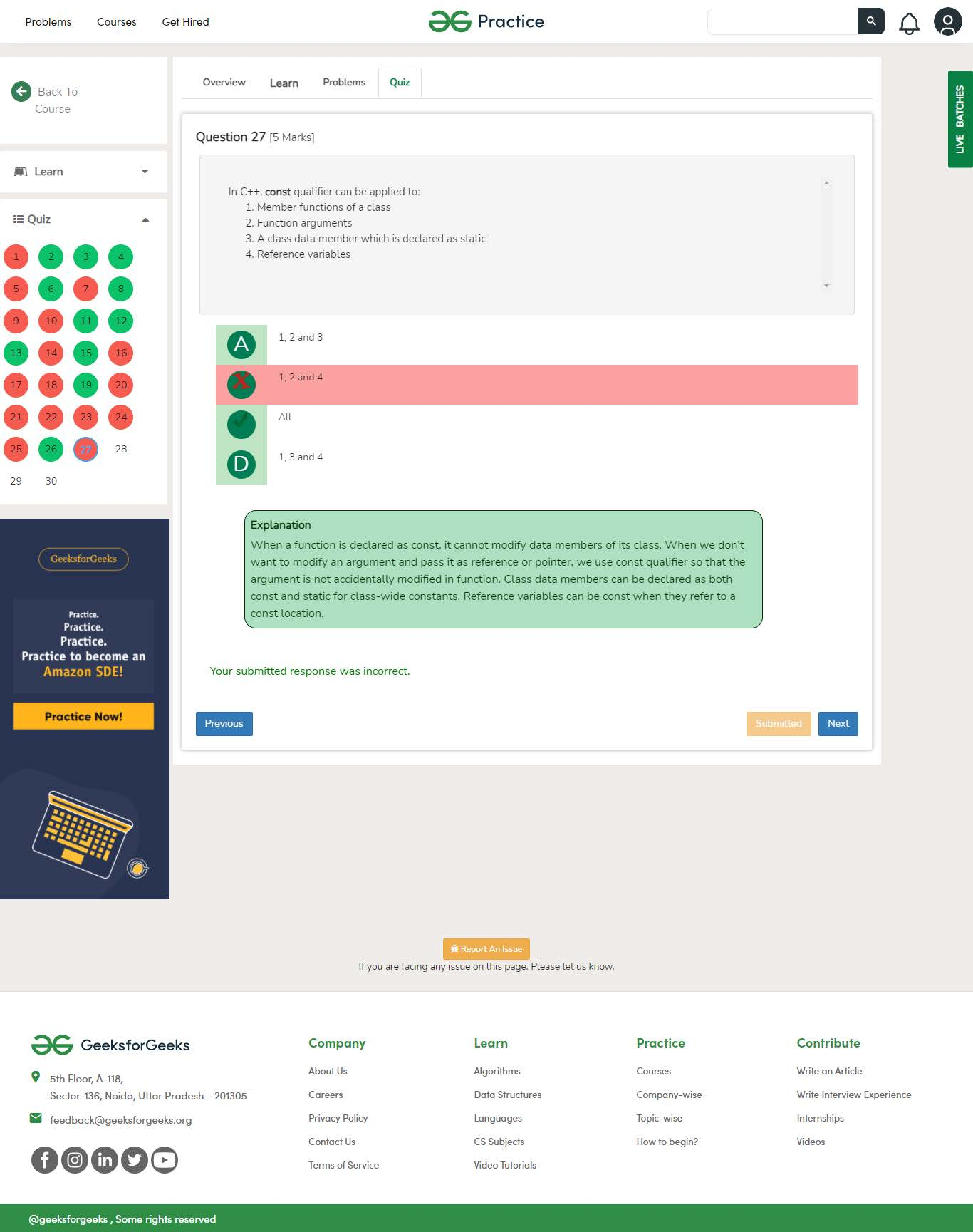
B 0

C Run-time Error

D Compilation Error at line: <em>class Derived3: public Derived1, public Derived2</em>

Explanation

This is a typical example of the diamond problem of multiple-inheritance. Here the base class member *a* is inherited through both Derived1 and Derived2. So there are two copies of *a* in Derived3 which makes the statement *cout << a;* ambiguous. The solution in C++ is to use virtual base classes.



Question 27 [5 Marks]

In C++, **const** qualifier can be applied to:

1. Member functions of a class
2. Function arguments
3. A class data member which is declared as static
4. Reference variables

A 1, 2 and 3

✗ 1, 2 and 4

✓ All

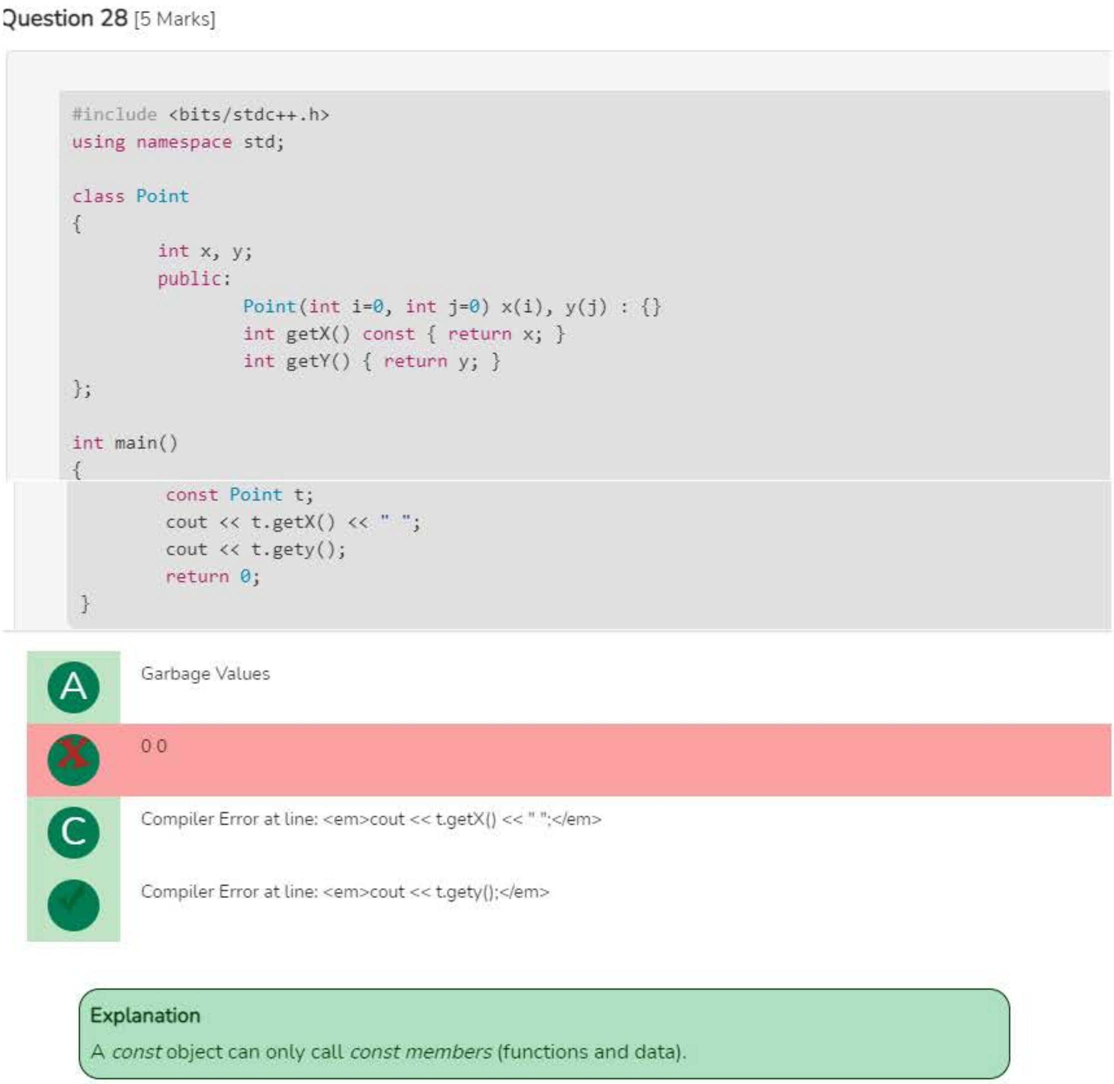
D 1, 3 and 4

Explanation

When a function is declared as const, it cannot modify data members of its class. When we don't want to modify an argument and pass it as reference or pointer, we use const qualifier so that the argument is not accidentally modified in function. Class data members can be declared as both const and static for class-wide constants. Reference variables can be const when they refer to a const location.

Your submitted response was incorrect.

If you are facing any issue on this page. Please let us know.



Question 28 [5 Marks]

```
#include <bits/stdc++.h>
using namespace std;

class Point
{
        int x, y;
        public:
                Point(int i=0, int j=0) x(i), y(j) : {}
                int getX() const { return x; }
                int getY() { return y; }
};

int main()
{
        const Point t;
        cout << t.getX() << " ";
        cout << t.gety();
        return 0;
}
```

A Garbage Values

✗ 0 0

C Compiler Error at line: <em>cout << t.getX() << " ";</em>

✓ Compiler Error at line: <em>cout << t.gety();</em>

Explanation

A *const* object can only call *const members* (functions and data).

Question 29 [5 Marks]

```
using namespace std;

int main()
{
    const char* p = "12345";
    const char **q = &p;
    *q = "abcde";
    const char *s = ++p;
    p = "XYZWVU";
    cout << *++s;

    return 0;
}
```

A Compilation Error

✓ c

✗ b

D Garbage Value

Explanation

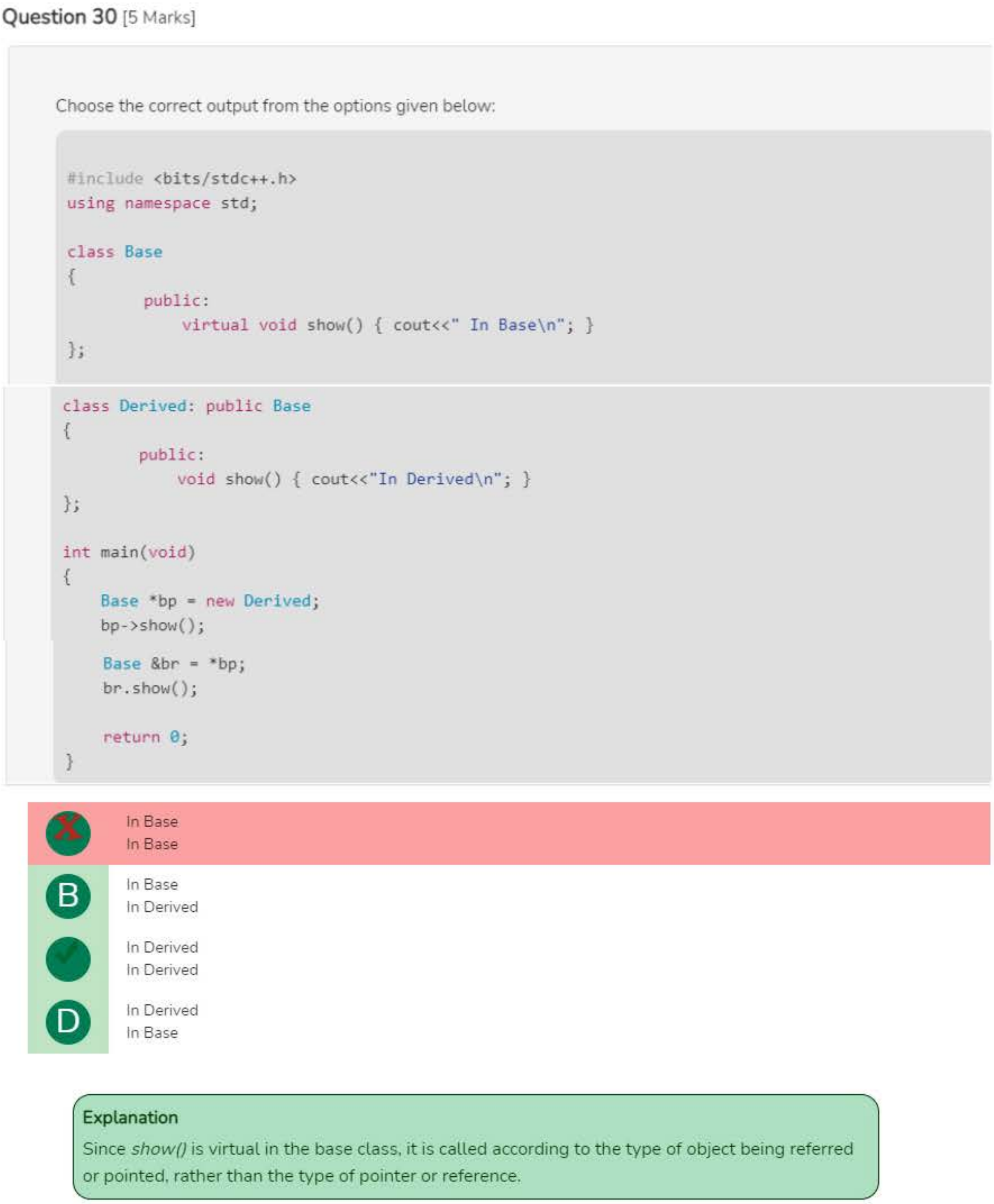
The output is **c**. *const char* p = "12345";* declares a pointer to a constant. So we can't assign something else to **p*, but we can assign a new value to *p*. *const char **q = &p;* declares a pointer to a pointer. We can't assign something else to ***q*, but we can assign new values to *q* and **q*. **q = "abcde";* changes *p* to point to *abcde*. *const char *s = ++p;* assigns address of literal *bcde* to *s*. Again **s* can't be assigned a new value, but *s* can be changed. The statement *cout << *++s;* changes *s* to *"cde"* and the first character at s is printed.

Your submitted response was incorrect.

Previous Submitted Next

Report An Issue

If you are facing any issue on this page. Please let us know.



Question 30 [5 Marks]

Choose the correct output from the options given below:

```
#include <bits/stdc++.h>
using namespace std;

class Base
{
        public:
            virtual void show() { cout<<" In Base\n"; }
};
class Derived: public Base
{
        public:
            void show() { cout<<"In Derived\n"; }
};

int main(void)
{
    Base *bp = new Derived;
    bp->show();

    Base &br = *bp;
    br.show();

    return 0;
}
```

✗ In Base
In Base

B In Base
In Derived

✓ In Derived
In Derived

D In Derived
In Base

Explanation

Since *show()* is virtual in the base class, it is called according to the type of object being referred or pointed, rather than the type of pointer or reference.