**Quiz 4:**

1. **Which of the following is true about virtual functions in C++.**

|  |  |
| --- | --- |
| **A** | Virtual functions are functions that can be overridden in derived class with the same signature. |
| **B** | Virtual functions enable run-time polymorphism in a inheritance hierarchy. |
| **C** | If a function is 'virtual' in the base class, the most-derived class's implementation of the function is called according to the actual type of the object referred to, regardless of the declared type of the pointer or reference. In non-virtual functions, the functions are called according to the type of reference or pointer. |
| **D** | All of the above |

1. **Predict output of the following program**

*#include<iostream>*

*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;*

*}*

**Output:**

|  |  |
| --- | --- |
| **A** | In Base  In base |
| **B** | In Base  In Derived |
| **C** | In Derived  In Derived |
| **D** | In Derived  In Base |

1. **Predict output of the following program**

*#include<iostream>*

*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, b;*

*Derived d;*

*bp = &d;*

*bp->show();*

*bp = &b;*

*bp->show();*

*return 0;*

*}*

**Output:**

|  |  |
| --- | --- |
| **A** | In Base  In base |
| **B** | In Base  In Derived |
| **C** | In Derived  In Derived |
| **D** | In Derived  In Base |

1. **Which of the following is true about pure virtual functions?**

1) Their implementation is not known in a class where they are declared.

2) If a class has a pure virtual function, then the class becomes abstract class and an instance of this class cannot be created.

**Output:**

|  |  |
| --- | --- |
| **A** | Both (1) and (2) |
| **B** | Only (1) |
| **C** | Only (2) |
| **D** | Neither 1 nor 2 |

1. **Predict output of the following program**

*#include<iostream>*

*using namespace std;*

*class Base*

*{*

*public:*

*virtual void show() = 0;*

*};*

*int main(void)*

*{*

*Base b;*

*Base \*bp;*

*return 0;*

*}*

**Output:**

|  |  |
| --- | --- |
| **A** | There are compiler errors in lines "Base b;" and "Base bp;" |
| **B** | There is compiler error in line "Base b;" |
| **C** | There is compiler error in line "Base bp;" |
| **D** | No compiler Error |

1. **Predict output of the following program**

*#include<iostream>*

*using namespace std;*

*class Base*

*{*

*public:*

*virtual void show() = 0;*

*};*

*class Derived : public Base { };*

*int main(void)*

*{*

*Derived q;*

*return 0;*

*}*

**Output:**

|  |  |
| --- | --- |
| **A** | Compiler Error: there cannot be an empty derived class |
| **B** | Compiler Error: Derived is abstract |
| **C** | No compiler Error |
|  |  |

1. **Predict output of the following program**

*#include<iostream>*

*using namespace std;*

*class Base*

*{*

*public:*

*virtual void show() = 0;*

*};*

*class Derived: public Base*

*{*

*public:*

*void show() { cout<<"In Derived \n"; }*

*};*

*int main(void)*

*{*

*Derived d;*

*Base &br = d;*

*br.show();*

*return 0;*

*}*

**Output:**

|  |  |
| --- | --- |
| **A** | Compiler Error in line "Base &br = d;" |
| **B** | Empty Output |
| **C** | In Derived |
|  |  |

1. **Can a constructor be virtual? Will the following program compile?**

|  |
| --- |
| *#include <iostream>*  *using namespace std;*  *class Base {*  *public:*  *virtual Base() {}*  *};*  *int main() {*  *return 0;*  *}* |

**Output:**

|  |  |
| --- | --- |
| **A** | Yes |
| **B** | No |

1. **Can a destructor be virtual? Will the following program compile?**

|  |
| --- |
| *#include <iostream>*  *using namespace std;*  *class Base {*  *public:*  *virtual ~Base() {}*  *};*  *int main() {*  *return 0;*  *}* |

**Output:**

|  |  |
| --- | --- |
| **A** | Yes |
| **B** | No |

1. **Predict output of the following program**

|  |
| --- |
| *#include<iostream>*  *using namespace std;*  *class Base  {*  *public:*  *Base()    { cout<<"Constructor: Base"<<endl; }*  *virtual ~Base()   { cout<<"Destructor : Base"<<endl; }*  *};*  *class Derived: public Base {*  *public:*  *Derived()   { cout<<"Constructor: Derived"<<endl; }*  *~Derived()  { cout<<"Destructor : Derived"<<endl; }*  *};*  *int main()  {*  *Base \*Var = new Derived();*  *delete Var;*  *return 0;*  *}* |

**Output:**

|  |  |
| --- | --- |
| **A** | Constructor: Base  Constructor: Derived  Destructor : Derived  Destructor : Base |
| **B** | Constructor: Base  Constructor: Derived  Destructor : Base |
| **C** | Constructor: Base  Constructor: Derived  Destructor : Derived |
| **D** | Constructor: Derived  Destructor : Derived |

1. **Can static functions be virtual? Will the following program compile?**

|  |
| --- |
| #*include<iostream>*  *using namespace std;*    *class Test*  *{*  *public:*  *virtual static void fun()  { }*  *};* |

**Output:**

|  |  |
| --- | --- |
| **A** | Yes |
| **B** | No |

1. **Predict output of the following program**

*#include <iostream>*

*using namespace std;*

*class A*

*{*

*public:*

*virtual void fun() { cout << "A::fun() "; }*

*};*

*class B: public A*

*{*

*public:*

*void fun() { cout << "B::fun() "; }*

*};*

*class C: public B*

*{*

*public:*

*void fun() { cout << "C::fun() "; }*

*};*

*int main()*

*{*

*B \*bp = new C;*

*bp->fun();*

*return 0;*

*}*

**Output:**

|  |  |
| --- | --- |
| **A** | A::fun() |
| **B** | B::fun() |
| **C** | C::fun() |
|  |  |

1. **Predict output of the following program**

*#include<iostream>*

*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->Base::show();  // Note the use of scope resolution here*

*return 0;*

*}*

**Output:**

|  |  |
| --- | --- |
| **A** | In Base |
| **B** | In Derived |
| **C** | Compiler Error |
| **D** | Runtime Error |

**Result:**

1. D

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

3. D: Initially base pointer points to a derived class object. Later it points to base class object

4. A

5. B: Since Base has a pure virtual function, it becomes an abstract class and an instance of it cannot be created. So there is an error in line "Base b". Note that there is no error in line "Base \*bp;". We can have pointers or references of abstract classes.

6. B: If we don't override the pure virtual function in derived class, then derived class also becomes abstract class.

7. C:

8. B: There is nothing like Virtual Constructor. Making constructors virtual doesn't make sense as constructor is responsible for creating an object and it can’t be delegated to any other object by virtual keyword means.

9. A:

10. A: Since the destructor is virtual, the derived class destructor is called which in turn calls base class destructor.

11. B: Static functions are class specific and may not be called on objects. Virtual functions are called according to the pointed or referred object.

12. C: The important thing to note here is B::fun() is virtual even if we have not uses virtual keyword with it. When a class has a virtual function, functions with same signature in all descendant classes automatically become virtual. We don't need to use virtual keyword in declaration of fun() in B and C. They are anyways virtual.

13. B: A base class function can be accessed with scope resolution operator even if the function is virtual.