**Quiz 03**

----------------------------------------------------------------

# Choose correct variant

#include<iostream>

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;

}

**Output:**

|  |  |
| --- | --- |
| **A** | Compiler Dependent |
| **B** | Base1′s constructor called  Base2′s constructor called  Derived’s constructor called |
| **C** | Base2′s constructor called  Base1′s constructor called  Derived’s constructor called |
| **D** | Compiler Error |

# Choose correct variant

Assume that an integer takes 4 bytes and there is no alignment in following classes, predict the output.

#include<iostream>

using namespace std;

class base {

int arr[10];

};

class b1: public base { };

class b2: public base { };

class derived: public b1, public b2 {};

int main(void)

{

cout << sizeof(derived);

return 0;

}

**Output:**

|  |  |
| --- | --- |
| **A** | 40 |
| **B** | 80 |
| **C** | 0 |
| **D** | 4 |

# Choose correct variant

#include<iostream>

using namespace std;

class P {

public:

void print() { cout <<" Inside P"; }

};

class Q : public P {

public:

void print() { cout <<" Inside Q"; }

};

class R: public Q { };

int main(void)

{

R r;

r.print();

return 0;

}

**Output:**

|  |  |
| --- | --- |
| **A** | Inside P |
| **B** | Inside Q |
| **C** | Compiler Error: Ambiguous call to print() |

# Choose correct variant

#include<iostream>

using namespace std;

class Base {

private:

int i, j;

public:

Base(int \_i = 0, int \_j = 0): i(\_i), j(\_j) { }

};

class Derived: public Base {

public:

void show(){

cout<<" i = "<<i<<" j = "<<j;

}

};

int main(void) {

Derived d;

d.show();

return 0;

}

**Output:**

|  |  |
| --- | --- |
| **A** | i = 0 j = 0 |
| **B** | Compiler Error: i and j are private in Base |
| **C** | Compiler Error: Could not call constructor of Base |

# Choose correct variant

#include<iostream>

using namespace std;

class Base

{

public:

int fun() { cout << "Base::fun() called"; }

int fun(int i) { cout << "Base::fun(int i) called"; }

};

class Derived: public Base

{

public:

int fun() { cout << "Derived::fun() called"; }

};

int main()

{

Derived d;

d.fun(5);

return 0;

}

**Output:**

|  |  |
| --- | --- |
| **A** | Base::fun(int i) called |
| **B** | Derived::fun() called |
| **C** | Base::fun() called |
| **D** | Compiler Error of no matching function for call to 'Derived::fun(int)' |

# Choose correct variant

#include<iostream>

using namespace std;

class Base {

public:

int fun() { cout << "Base::fun() called"; }

int fun(int i) { cout << "Base::fun(int i) called"; }

};

class Derived: public Base {

public:

int fun() { cout << "Derived::fun() called"; }

};

int main() {

Derived d;

d.Base::fun(5);

return 0;

}

**Output:**

|  |  |
| --- | --- |
| **A** | Compiler Error |
| **B** | Base::fun(int i) called |

# Choose correct variant

#include<iostream>

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;

}

**Output:**

|  |  |
| --- | --- |
| **A** | 10 10 |
| **B** | Compiler Error of class 'Derived' does not have any field named 'x' |
| **C** | 0 0 |

# Choose correct variant

#include<iostream>

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 DerivedDerived: public Derived1, public Derived2

{

public:

void show() {cout << a;}

};

int main(void)

{

DerivedDerived d;

d.show();

return 0;

}

**Output:**

|  |  |
| --- | --- |
| **A** | Compiler Error: reference to 'a' is ambiguous |
| **B** | 0 |
| **C** | A garbage value |

# Choose correct variant

#include<iostream>

using namespace std;

class Base1

{

public:

char c;

};

class Base2

{

public:

int c;

};

class Derived: public Base1, public Base2

{

public:

void show() { cout << c; }

};

int main(void)

{

Derived d;

d.show();

return 0;

}

**Output:**

|  |  |
| --- | --- |
| **A** | Compiler Error : reference to 'c' is ambiguous |
| **B** | Garbage Value |
| **C** | 0 |

# 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 an 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 |

# 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 |

# **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 |

# **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 |

# **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 |

# **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 |