1. What is meant by multiple inheritance?

a) Deriving a base class from derived class

b) Deriving a derived class from base class

c) Deriving a derived class from more than one base class

d) None of the mentioned

2. Which symbol is used to create multiple inheritance?

a) Dot

b) Comma

c) Dollar

d) None of the mentioned

3. Which of the following advantages we lose by using multiple inheritance?

a) Dynamic binding

b) Polymorphism

c) Both Dynamic binding & Polymorphism

d) None of the mentioned

4. What is the output of this program?

#include <iostream>

using namespace std;

class polygon

{

protected:

int width, height;

public:

void set\_values (int a, int b)

{

width = a; height = b;}

};

class output1

{

public:

void output (int i);

};

void output1::output (int i)

{

cout << i << endl;

}

class rectangle: public polygon, public output1

{

public:

int area ()

{

return (width \* height);

}

};

class triangle: public polygon, public output1

{

public:

int area ()

{

return (width \* height / 2);

}

};

int main ()

{

rectangle rect;

triangle trgl;

rect.set\_values (4, 5);

trgl.set\_values (4, 5);

rect.output (rect.area());

trgl.output (trgl.area());

return 0;

}

a) 20

b) 10

c) 20

10

5. What is the output of this program?

1. #include <iostream>
2. using namespace std;
3. class Base
4. {
5. public:
6. virtual void print() const = 0;
7. };
8. class DerivedOne : public Base
9. {
10. public:
11. void print() const
12. {
13. cout << "DerivedOne**\n**";
14. }
15. };
16. class DerivedTwo : public Base
17. {
18. public:
19. void print() const
20. {
21. cout << "DerivedTwo**\n**";
22. }
23. };
24. class Multiple : public DerivedOne, public DerivedTwo
25. {
26. public:
27. void print() const
28. {
29. DerivedTwo :: print();
30. }
31. };
32. int main()
33. {
34. int i;
35. Multiple both;
36. DerivedOne one;
37. DerivedTwo two;
38. Base \*array[ 3 ];
39. array[ 0 ] = &both;
40. array[ 1 ] = &one;
41. array[ 2 ] = &two;
42. array[ i ] -> print();
43. return 0;
44. }

a) DerivedOne  
b) DerivedTwo  
c) Error  
d) None of the mentioned

6. What is the output of this program?

1. #include <iostream>
2. using namespace std;
3. class student
4. {
5. public:
6. int rno , m1 , m2 ;
7. void get()
8. {
9. rno = 15, m1 = 10, m2 = 10;
10. }
11. };
12. class sports
13. {
14. public:
15. int sm;
16. void getsm()
17. {
18. sm = 10;
19. }
20. };
21. class statement:public student,public sports
22. {
23. int tot,avg;
24. public:
25. void display()
26. {
27. tot = (m1 + m2 + sm);
28. avg = tot / 3;
29. cout << tot;
30. cout << avg;
31. }
32. };
33. int main()
34. {
35. statement obj;
36. obj.get();
37. obj.getsm();
38. obj.display();
39. }

a) 3100  
b) 3010  
c) 2010  
d) 1010

7. What is the output of this program?

1. #include <iostream>
2. using namespace std;
3. struct a
4. {
5. int count;
6. };
7. struct b
8. {
9. int\* value;
10. };
11. struct c : public a, public b
12. {
13. };
14. int main()
15. {
16. c\* p = new c;
17. p->value = 0;
18. cout << "Inherited";
19. return 0;
20. }

a) Inherited  
b) Error  
c) Runtime error  
d) None of the mentioned

8. What is the output of this program?

1. #include <iostream>
2. using namespace std;
3. class Base1
4. {
5. protected:
6. int SampleDataOne;
7. public:
8. Base1()
9. {
10. SampleDataOne = 100;
11. }
12. ~Base1()
13. {
14. }
15. int SampleFunctOne()
16. {
17. return SampleDataOne;
18. }
19. };
20. class Base2
21. {
22. protected:
23. int SampleDataTwo;
24. public:
25. Base2()
26. {
27. SampleDataTwo = 200;
28. }
29. ~Base2()
30. {
31. }
32. int SampleFunctTwo()
33. {
34. return SampleDataTwo;
35. }
36. };
37. class Derived1 : public Base1, public Base2
38. {
39. int MyData;
40. public:
41. Derived1()
42. {
43. MyData = 300;
44. }
45. ~Derived1()
46. {
47. }
48. int MyFunct()
49. {
50. return (MyData + SampleDataOne + SampleDataTwo);
51. }
52. };
53. int main()
54. {
55. Base1 SampleObjOne;
56. Base2 SampleObjTwo;
57. Derived1 SampleObjThree;
58. cout << SampleObjThree.Base1 :: SampleFunctOne() << endl;
59. cout << SampleObjThree.Base2 :: SampleFunctTwo() << endl;
60. return 0;
61. }

a) 100  
b) 200  
c) Both 100 & 200  
d) None of the mentioned

9. Which design patterns benefit from the multiple inheritances?  
a) Adapter and observer pattern  
b) Code pattern  
c) Glue pattern  
d) None of the mentioned  
View Answer

10. What are the things are inherited from the base class?  
a) Constructor and its destructor  
b) Operator=() members  
c) Friends  
d) All of the mentioned

.

11. Which class is used to design the base class?  
a) abstract class  
b) derived class  
c) base class  
d) none of the mentioned

12. Which is used to create a pure virtual function ?  
a) $  
b) =0  
c) &  
d) !

13. Which is also called as abstract class?  
a) virtual function  
b) pure virtual function  
c) derived class  
d) none of the mentioned

14. What is the output of this program?

1. #include <iostream>
2. using namespace std;
3. class p
4. {
5. protected:
6. int width, height;
7. public:
8. void set\_values (int a, int b)
9. {
10. width = a; height = b;
11. }
12. virtual int area (void) = 0;
13. };
14. class r: public p
15. {
16. public:
17. int area (void)
18. {
19. return (width \* height);
20. }
21. };
22. class t: public p
23. {
24. public:
25. int area (void)
26. {
27. return (width \* height / 2);
28. }
29. };
30. int main ()
31. {
32. r rect;
33. t trgl;
34. p \* ppoly1 = &rect;
35. p \* ppoly2 = &trgl;
36. ppoly1->set\_values (4, 5);
37. ppoly2->set\_values (4, 5);
38. cout << ppoly1 -> area() ;
39. cout << ppoly2 -> area();
40. return 0;
41. }

a) 1020  
b) 20  
c) 10  
d) 2010

15. What is the output of this program?

1. #include <iostream>
2. using namespace std;
3. class MyInterface
4. {
5. public:
6. virtual void Display() = 0;
7. };
8. class Class1 : public MyInterface
9. {
10. public:
11. void Display()
12. {
13. int a = 5;
14. cout << a;
15. }
16. };
17. class Class2 : public MyInterface
18. {
19. public:
20. void Display()
21. {
22. cout <<" 5" << endl;
23. }
24. };
25. int main()
26. {
27. Class1 obj1;
28. obj1.Display();
29. Class2 obj2;
30. obj2.Display();
31. return 0;
32. }

a) 5  
b) 10  
c) 5 5  
d) None of the mentioned

16. What is the output of this program?

1. #include <iostream>
2. using namespace std;
3. class sample
4. {
5. public:
6. virtual void example() = 0;
7. };
8. class Ex1:public sample
9. {
10. public:
11. void example()
12. {
13. cout << "ubuntu";
14. }
15. };
16. class Ex2:public sample
17. {
18. public:
19. void example()
20. {
21. cout << " is awesome";
22. }
23. };
24. int main()
25. {
26. sample\* arra[2];
27. Ex1 e1;
28. Ex2 e2;
29. arra[0]=&e1;
30. arra[1]=&e2;
31. arra[0]->example();
32. arra[1]->example();
33. }

a) ubuntu  
b) is awesome  
c) ubuntu is awesome  
d) none of the mentioned

17. What is the output of this program?

1. #include <iostream>
2. using namespace std;
3. class Base
4. {
5. public:
6. virtual void print() const = 0;
7. };
8. class DerivedOne : virtual public Base
9. {
10. public:
11. void print() const
12. {
13. cout << "1";
14. }
15. };
16. class DerivedTwo : virtual public Base
17. {
18. public:
19. void print() const
20. {
21. cout << "2";
22. }
23. };
24. class Multiple : public DerivedOne, DerivedTwo
25. {
26. public:
27. void print() const
28. {
29. DerivedTwo::print();
30. }
31. };
32. int main()
33. {
34. Multiple both;
35. DerivedOne one;
36. DerivedTwo two;
37. Base \*array[ 3 ];
38. array[ 0 ] = &both;
39. array[ 1 ] = &one;
40. array[ 2 ] = &two;
41. for ( int i = 0; i < 3; i++ )
42. array[ i ] -> print();
43. return 0;
44. }

a) 121  
b) 212  
c) 12  
d) none of the mentioned

18. What is meant by pure virtual function?  
a) Function which does not have definition of its own  
b) Function which does have definition of its own  
c) Function which does not have any return type  
d) None of the mentioned

19. Pick out the correct option.  
a) We cannot make an instance of an abstract base class  
b) We can make an instance of an abstract base class  
c) We can make an instance of an abstract super class  
d) None of the mentioned

20. Where does the abstract class is used?  
a) base class only  
b) derived class  
c) both derived & base class  
d) none of the mentioned

21. 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. |
|  |  |

|  |
| --- |
| 1. #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;  } |

Run on IDE

|  |  |
| --- | --- |
| A | In Base  B) In Base |
|  | In Base  C)In Derived |
|  | In Derived |
|  | D) In Derived  In Base  23. Output of 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;  } |   A  In Base  In Base  B  In Base  In Derived  C  In Derived  In Derived  D  In Derived  In Base |

24. Which of the following is true about pure virtual functions? 1) Their implementation is not provided 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.

|  |  |
| --- | --- |
| A | Both 1 and 2 |
| B | Only 1 |
| C | Only 2 |

D Neither 1 nor 2

|  |
| --- |
| 25. #include<iostream>  using namespace std;    class Base  {  public:      virtual void show() = 0;  };    int main(void)  {      Base b;      Base \*bp;      return 0;  } |

|  |  |
| --- | --- |
| 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 |
|  |  |

26. Predict the output of 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;  } |

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

|  |
| --- |
| 27. #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;  } |

Run on IDE

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

Can a constructor be virtual? Will the following program compile?

|  |
| --- |
| 28. #include <iostream>  using namespace std;  class Base {  public:    virtual Base() {}  };  int main() {     return 0;  } |

Run on IDE

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

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

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

Run on IDE

|  |  |
| --- | --- |
| A | Yes |

B NO

|  |
| --- |
| 30. Predict the output:  #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;  } |

Run on IDE

|  |  |
| --- | --- |
| 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 |