**C++ Programming Questions and Answers – Derived Classes**

1. Where is the derived class is derived from?  
a) derived  
b) base  
c) both derived & base  
d) class

Answer: b  
Explanation: Because derived inherits functions and variables from base.

2. Pick out the correct statement.  
a) A derived class’s constructor cannot explicitly invokes its base class’s constructor  
b) A derived class’s destructor cannot invoke its base class’s destructor  
c) A derived class’s destructor can invoke its base class’s destructor  
d) A derived class’s destructor can invoke its base & derived class’s destructor

Answer: b  
Explanation: Destructors are automatically invoked when an object goes out of scope or when a dynamically allocated object is deleted. Inheritance does not change this behavior. This is the reason a derived destructor cannot invoke its base class destructor.

3. Which of the following can derived class inherit?  
a) members  
b) functions  
c) both members & functions  
d) classes

Answer: c  
Explanation: Both data members and member functions are inherited by derived class in C++.

4. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. class A
4. {
5. public:
6. A(int n )
7. {
8. cout << n;
9. }
10. };
11. class B: public A
12. {
13. public:
14. B(int n, double d)
15. : A(n)
16. {
17. cout << d;
18. }
19. };
20. class C: public B
21. {
22. public:
23. C(int n, double d, char ch)
24. : B(n, d)
25. {
26. cout <<ch;
27. }
28. };
29. int main()
30. {
31. C c(5, 4.3, 'R');
32. return 0;
33. }

a) 54.3R  
b) R4.35  
c) 4.3R5  
d) R2.6

Answer: a  
Explanation: In this program, We are passing the value and manipulating by using the derived class.  
Output:

$ g++ der.cpp

$ a.out

54.3R

5. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. class BaseClass
4. {
5. protected:
6. int i;
7. public:
8. BaseClass(int x)
9. {
10. i = x;
11. }
12. ~BaseClass()
13. {
14. }
15. };
16. class DerivedClass: public BaseClass
17. {
18. int j;
19. public:
20. DerivedClass(int x, int y): BaseClass(y)
21. {
22. j = x;
23. }
24. ~DerivedClass()
25. {
26. }
27. void show()
28. {
29. cout << i << " " << j << endl;
30. }
31. };
32. int main()
33. {
34. DerivedClass ob(3, 4);
35. ob.show();
36. return 0;
37. }

a) 3 4  
b) 4 3  
c) 4  
d) 3

Answer: b  
Explanation: In this program, We are passing the values and assigning it to i and j and we are printing it.  
Output:

$ g++ der1.cpp

$ a.out

4 3

6. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. class Base
4. {
5. public:
6. int m;
7. Base(int n=0)
8. : m(n)
9. {
10. cout << "Base" << endl;
11. }
12. };
13. class Derived: public Base
14. {
15. public:
16. double d;
17. Derived(double de = 0.0)
18. : d(de)
19. {
20. cout << "Derived" << endl;
21. }
22. };
23. int main()
24. {
25. cout << "Instantiating Base" << endl;
26. Base cBase;
27. cout << "Instantiating Derived" << endl;
28. Derived cDerived;
29. return 0;
30. }

a) Instantiating Base

Base

Instantiating Derived

Base

Derived

b) Instantiating Base

Instantiating Derived

Base

Derived

c) Instantiating Base

Base

Instantiating Derived

Base

d) Instantiating Base

Answer: a  
Explanation: In this program, We are printing the execution order of the program.  
Output:

$ g++ der2.cpp

$ a.out

Instantiating Base

Base

Instantiating Derived

Base

Derived

7. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. class Parent
4. {
5. public:
6. Parent (void)
7. {
8. cout << "Parent()**\n**";
9. }
10. Parent (int i)
11. {
12. cout << "Parent("<< i << ")**\n**";
13. };
14. Parent (void)
15. {
16. cout << "~Parent()**\n**";
17. };
18. };
19. class Child1 : public Parent { };
20. class Child2 : public Parent
21. {
22. public:
23. Child2 (void)
24. {
25. cout << "Child2()**\n**";
26. }
27. Child2 (int i) : Parent (i)
28. {
29. cout << "Child2(" << i << ")**\n**";
30. }
31. ~Child2 (void)
32. {
33. cout << "~Child2()**\n**";
34. }
35. };
36. int main (void)
37. {
38. Child1 a;
39. Child2 b;
40. Child2 c(42);
41. return 0;
42. }

a)Parent()

Parent()

Child2()

Parent(42)

Child2(42)

~Child2()

~Parent()

~Child2()

~Parent()

~Parent()

b) error  
c) runtime error  
d) Parent(42)

Answer: b  
Explanation: In this program, We got an error in overloading because we didn’t invoke the destructor of parent.

8. What will be the output of the following C++ code?

1. #include<iostream>
2. using namespace std;
3. class X
4. {
5. int m;
6. public:
7. X() : m(10)
8. {
9. }
10. X(int mm): m(mm)
11. {
12. }
13. int getm()
14. {
15. return m;
16. }
17. };
18. class Y : public X
19. {
20. int n;
21. public:
22. Y(int nn) : n(nn) {}
23. int getn() { return n; }
24. };
25. int main()
26. {
27. Y yobj( 100 );
28. cout << yobj.getm() << " " << yobj.getn() << endl;
29. }

a) 10 100  
b) 100 10  
c) 10 10  
d) 100 100

Answer: a  
Explanation: In this program, We are passing the value and getting the result by derived class.  
Output:

$ g++ der5.cpp

$ a.out

10 100

9. Which operator is used to declare the destructor?  
a) #  
b) ~  
c) @  
d) $

Answer: b  
Explanation: tilde(~) is used to declare destructor of a class.

10. Which constructor will initialize the base class data member?  
a) derived class  
b) base class  
c) class  
d) derived & base class

Answer: b  
Explanation: Because it is having the proper data set to initialize, Otherwise it will throw an error.

# C++ Programming Questions and Answers – Abstract Classes – 1

1. Which class is used to design the base class?  
a) abstract class  
b) derived class  
c) base class  
d) derived & base class

Answer: a  
Explanation: Abstract class is used to design base class because functions of abstract class can be overridden in

derived class hence derived class from same base class can have common method with different implementation, hence forcing encapsulation.

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

Answer: b  
Explanation: For making a method as pure virtual function, We have to append ‘=0’ to the class or method.

3. Which is also called as abstract class?  
a) virtual function  
b) pure virtual function  
c) derived class  
d) base class

Answer: b  
Explanation: Classes that contain at least one pure virtual function are called as abstract base classes.

4. What will be the output of the following C++ code?

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

Answer: d  
Explanation: In this program, We are calculating the area of rectangle and  
triangle by using abstract class.  
Output:

$ g++ abs.cpp

$ a.out

2010

5. What will be the output of the following C++ code?

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) 15

Answer: c  
Explanation: In this program, We are displaying the data from the two classes by using abstract class.  
Output:

$ g++ abs1.cpp

$ a.out

5 5

6. What will be the output of the following C++ code?

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) ubunt esome

Answer: c  
Explanation: In this program, We are combining the two statements from two classes and printing it by using abstract class.  
Output:

$ g++ abs3.cpp

$ a.out

ubuntu is awesome

7. What will be the output of the following C++ code?

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) 215

Answer: b  
Explanation: In this program, We are executing these based on the condition given in array. So it is printing as 212.  
Output:

$ g++ abs4.cpp

$ a.out

212

8. 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) Function which does not have any return type & own definition

Answer: a  
Explanation: As the name itself implies, it have to depend on other class only.

9. 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) We can make an instance of an abstract derived class

Answer: a  
Explanation: We cannot make an instance of an abstract base class.

10. Where does the abstract class is used?  
a) base class only  
b) derived class  
c) both derived & base class  
d) virtual class

Answer: a  
Explanation: As base class only as it helps in encapsulation of similar functioning of derived classes.

# C++ Programming Questions and Answers – Abstract Classes – 2

1. What is an abstract class in C++?  
a) Class specifically used as a base class with atleast one virtual functions  
b) Class specifically used as a base class with atleast one pure virtual functions  
c) Class from which any class is derived  
d) Any Class in C++ is an abstract class

Answer: b  
Explanation: An abstract class is defined as a class which is specifically used as a base class. An abstract class should have atleast one pure virtual function.

2. What is a pure virtual function in C++?  
a) A virtual function defined in a base class  
b) A virtual function declared in a base class  
c) Any function in a class  
d) A function without definition in a base class

Answer: b  
Explanation: Pure virtual function is a virtual function which has no definition/implementation in the base class.

3. Which is the correct syntax of defining a pure virtual function?  
a) pure virtual return\_type func();  
b) virtual return\_type func() pure;  
c) virtual return\_type func() = 0;  
d) virtual return\_type func();

Answer: c  
Explanation: virtual return\_type function\_name(parameters) = 0; where {=0} is called pure specifier.

4. Which is the correct statement about pure virtual functions?  
a) They should be defined inside a base class  
b) Pure keyword should be used to declare a pure virtual function  
c) Pure virtual function is implemented in derived classes  
d) Pure virtual function cannot implemented in derived classes

Answer: c  
Explanation: A pure virtual function does not have a definition corresponding to base class. All derived class may or may not have an implementation of a pure virtual function. there is no pure keyword in C++.

5. Pick the correct statement.  
a) Pure virtual functions and virtual functions are the same  
b) Both Pure virtual function and virtual function have an implementation in the base class  
c) Pure virtual function has no implementation in the base class whereas virtual function may have an implementation in the base class  
d) The base class has no pure virtual function

Answer: c  
Explanation: Pure virtual function has no implementation in the base class whereas virtual function may have an implementation in the base class. The base class has at least one pure virtual function.

6. What will be the output of the following C++ code?

#include <iostream>

#include <string>

using namespace std;

class A

{

int a;

public:

virtual void func() = 0;

};

class B: public A

{

public:

void func(){

cout<<"Class B"<<endl;

}

};

int main(int argc, char const \*argv[])

{

B b;

b.func();

return 0;

}

a) Class B  
b) Error  
c) Segmentation fault  
d) No output

Answer: a  
Explanation: The program is correct so no error occurs hence the program runs successfully and b is calling is func() function therefore “Class B” is printed.

7. What will be the output of the following C++ code?

#include <iostream>

#include <string>

using namespace std;

class A

{

int a;

public:

virtual void func() = 0;

};

class B: public A

{

public:

void func(){

cout<<"Class B"<<endl;

}

};

int main(int argc, char const \*argv[])

{

A a;

a.func();

return 0;

}

a) Class B  
b) Error  
c) Segmentation fault  
d) No output

Answer: b  
Explanation: The C++ does allows to declare a normal object for an astract class therefore the program throws an error as we are trying to declare an object of abstract class.

8. What will be the output of the following C++ code?

#include <iostream>

#include <string>

using namespace std;

class A

{

int a;

public:

virtual void func() = 0;

};

class B: public A

{

public:

void func(){

cout<<"Class B"<<endl;

}

};

int main(int argc, char const \*argv[])

{

A \*a;

a->func();

return 0;

}

a) Class B  
b) Error  
c) Segmentation fault  
d) No output

Answer: c  
Explanation: As we are allowed to declare a pointer object of an abstract so the program does not give any compilation error but as there is no definition of func() function corresponding to class A, therefore, the program gives segmentation fault as it is not able to call such function from class A.

# C++ Programming Questions and Answers – Design of Class Hierarchies

1. Which interface determines how your class will be used by another program?  
a) public  
b) private  
c) protected  
d) void

Answer: a  
Explanation: If we invoked the interface as public means, We can access the program from other programs also.

2. Pick out the correct statement about the override.  
a) Overriding refers to a derived class function that has the same name and signature as a base class virtual function  
b) Overriding has different names  
c) Overriding refers to a derived class  
d) Overriding has different names & it refers to a derived class

Answer: a  
Explanation: Overriding refers to a derived class function that has the same name and signature as a base class virtual function.

3. How many ways of reusing are there in the class hierarchy?  
a) 1  
b) 2  
c) 3  
d) 4

Answer: b  
Explanation: Class hierarchies promote reuse in two ways. They are code sharing and interface sharing.

4. Which of the following statements regarding abstract base classes, concrete derived classes, and standalone classes in C++ is correct?  
a) Abstract base classes cannot be instantiated  
b) Concrete derived classes must override all methods of the abstract base class  
c) Standalone classes cannot inherit from abstract base classes  
d) Standalone classes cannot have member functions

Answer: a  
Explanation: Abstract base classes contain one or more pure virtual functions and cannot be instantiated. They serve as base classes for other classes.

5. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. class BaseClass
4. {
5. int i;
6. public:
7. void setInt(int n);
8. int getInt();
9. };
10. class DerivedClass : public BaseClass
11. {
12. int j;
13. public:
14. void setJ(int n);
15. int mul();
16. };
17. void BaseClass::setInt(int n)
18. {
19. i = n;
20. }
21. int BaseClass::getInt()
22. {
23. return i;
24. }
25. void DerivedClass::setJ(int n)
26. {
27. j = n;
28. }
29. int DerivedClass::mul()
30. {
31. return j \* getInt();
32. }
33. int main()
34. {
35. DerivedClass ob;
36. ob.setInt(10);
37. ob.setJ(4);
38. cout << ob.mul();
39. return 0;
40. }

a) 10  
b) 4  
c) 40  
d) 30

Answer: c  
Explanation: In this program, We are multiplying the value 10 and 4 by using inheritance.  
Output:

$ g++ des.cpp

$ a.out

40

6. Pick out the correct statement about multiple inheritances.  
a) Deriving a class from one direct base class  
b) Deriving a class from more than one direct base class  
c) Deriving a class from more than one direct derived class  
d) Deriving a class from more than one direct derivedbase class

Answer: b  
Explanation: In multiple inheritances, We are able to derive a class from more than one base class.

7. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. class BaseClass
4. {
5. int x;
6. public:
7. void setx(int n)
8. {
9. x = n;
10. }
11. void showx()
12. {
13. cout << x ;
14. }
15. };
16. class DerivedClass : private BaseClass
17. {
18. int y;
19. public:
20. void setxy(int n, int m)
21. {
22. setx(n);
23. y = m;
24. }
25. void showxy()
26. {
27. showx();
28. cout << y << '**\n**';
29. }
30. };
31. int main()
32. {
33. DerivedClass ob;
34. ob.setxy(10, 20);
35. ob.showxy();
36. return 0;
37. }

a) 10  
b) 20  
c) 1020  
d) 1120

Answer: c  
Explanation: In this program, We are passing the values from the main class and printing it on the inherited classes.  
Output:

$ g++ des2.cpp

$ a.out

1020

8. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. class BaseClass
4. {
5. public:
6. virtual void myFunction()
7. {
8. cout << "1";
9. }
10. };
11. class DerivedClass1 : public BaseClass
12. {
13. public:
14. void myFunction()
15. {
16. cout << "2";
17. }
18. };
19. class DerivedClass2 : public DerivedClass1
20. {
21. public:
22. void myFunction()
23. {
24. cout << "3";
25. }
26. };
27. int main()
28. {
29. BaseClass \*p;
30. BaseClass ob;
31. DerivedClass1 derivedObject1;
32. DerivedClass2 derivedObject2;
33. p = &ob;
34. p -> myFunction();
35. p = &derivedObject1;
36. p -> myFunction();
37. p = &derivedObject2;
38. p -> myFunction();
39. return 0;
40. }

a) 123  
b) 12  
c) 213  
d) 321

Answer: a  
Explanation: We are passing the objects and executing them in a certain order and we are printing the program flow.  
Output:

$ g++ des3.cpp

$ a.out

123

9. What does inheritance allow you to do?  
a) create a class  
b) create a hierarchy of classes  
c) access methods  
d) create a method

Answer: b  
Explanation: Inheritance helps in creating hierarchy of classes by making connections between different classes in which one is called base class and other is class derived class.

10. What is the syntax of inheritance of class?  
a) class name  
b) class name: access specifier  
c) class name: access specifier class name  
d) access specifier class name  
Answer: c  
Explanation: Syntax is:  
class Class\_Name: Access\_Specifier Base\_Class\_Name

example:  
class A{};  
class B: public A{};

# C++ Programming Questions and Answers – Class Hierarchies and Abstract Classes

1. Which of the following statements regarding absolute and concrete classes in C++ is correct?  
a) Absolute classes cannot have concrete subclasses  
b) Concrete classes cannot have abstract methods  
c) Absolute classes cannot have member variables  
d) Concrete classes cannot have constructors

Answer: b  
Explanation: Concrete classes provide implementations for all their member functions, so they cannot have abstract methods.

2. What is meant by polymorphism?  
a) class having many forms

b) class having only single form  
c) class having two forms  
d) class having four forms

Answer: a  
Explanation: Polymorphism is literally meant class having many forms.

3. How many types of inheritance are there in c++?  
a) 2  
b) 3  
c) 4  
d) 5

Answer: d  
Explanation: There are five types of inheritance in c++. They are single, Multiple, Hierarchical, Multilevel, Hybrid.

4. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. class stu
4. {
5. protected:
6. int rno;
7. public:
8. void get\_no(int a)
9. {
10. rno = a;
11. }
12. void put\_no(void)
13. {
14. }
15. };
16. class test:public stu
17. {
18. protected:
19. float part1,part2;
20. public:
21. void get\_mark(float x, float y)
22. {
23. part1 = x;
24. part2 = y;
25. }
26. void put\_marks()
27. {
28. }
29. };
30. class sports
31. {
32. protected:
33. float score;
34. public:
35. void getscore(float s)
36. {
37. score = s;
38. }
39. void putscore(void)
40. {
41. }
42. };
43. class result: public test, public sports
44. {
45. float total;
46. public:
47. void display(void);
48. };
49. void result::display(void)
50. {
51. total = part1 + part2 + score;
52. put\_no();
53. put\_marks();
54. putscore();
55. cout << "Total Score=" << total << "**\n**";
56. }
57. int main()
58. {
59. result stu;
60. stu.get\_no(123);
61. stu.get\_mark(27.5, 33.0);
62. stu.getscore(6.0);
63. stu.display();
64. return 0;
65. }

a) 66.5  
b) 64.5  
c) 62.5  
d) 60.5

Answer: a  
Explanation: In this program, We are passing the values by using different methods and totaling the marks to get the result.  
Output:

$ g++ class.cpp

$ a.out

Total Score=66.5

5. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. class poly
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. };
13. class Coutput
14. {
15. public:
16. void output(int i);
17. };
18. void Coutput::output(int i)
19. {
20. cout << i;
21. }
22. class rect:public poly, public Coutput
23. {
24. public:
25. int area()
26. {
27. return(width \* height);
28. }
29. };
30. class tri:public poly, public Coutput
31. {
32. public:
33. int area()
34. {
35. return(width \* height / 2);
36. }
37. };
38. int main()
39. {
40. rect rect;
41. tri trgl;
42. rect.set\_values(3, 4);
43. trgl.set\_values(4, 5);
44. rect.output(rect.area());
45. trgl.output(trgl.area());
46. return 0;
47. }

a) 1212  
b) 1210  
c) 1010  
d) 1250

Answer: b  
Explanation: In this program, We are calculating the area of rectangle and triangle by using multilevel inheritance.

$ g++ class1.cpp

$ a.out

1210

6. What is meant by container ship?  
a) class contains objects of other class types as its members  
b) class contains objects of other class types as its objects  
c) class contains objects of other class types as its members 7 also objects  
d) class contains objects of other class types as its members 9 also objects

Answer: a  
Explanation: Container ship is a class contains objects of other class types as its members.

7. How many types of the constructor are there in C++?  
a) 1  
b) 2  
c) 3  
d) 4

Answer: c  
Explanation: There are three types of constructor in C++. They are the Default constructor, Parameterized constructor, Copy constructor.

8. How many constructors can present in a class?  
a) 1  
b) 2  
c) 3  
d) multiple

Answer: d  
Explanation: There can be multiple constructors of the same class, provided they have different signatures.

9. What should be the name of the constructor?  
a) same as the object  
b) same as the member  
c) same as the class  
d) same as the function

Answer: c  
Explanation: Constructor name should be same as the class name.

10. What does derived class does not inherit from the base class?  
a) constructor and destructor  
b) friends  
c) operator = () members  
d) all of the mentioned

Answer: d  
Explanation: The derived class inherits everything from the base class except the given things.

# C++ Programming Questions and Answers – Simple String Template

1. What is a template?  
a) A template is a formula for creating a generic class  
b) A template is used to manipulate the class  
c) A template is used for creating the attributes  
d) A template is used to delete the class

Answer: a  
Explanation: Templates are used for creating generic classes to handle different types in single classes.

2. Pick out the correct statement about string template.  
a) It is used to replace a string  
b) It is used to replace a string with another string at runtime  
c) It is used to delete a string  
d) It is used to create a string

Answer: b  
Explanation: Every string template is used to replace the string with another string at runtime.

3. How to declare a template?  
a) tem  
b) temp  
c) template<>  
d) temp()

Answer: c  
Explanation: template<> syntax is used.  
An example for calculating max of two ints, floats, doubles, or any other number type where T indicates the type of the parameters passes.  
template <typename T>  
T max(T a, T b){  
return a > b? a : b;  
}

4. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. template <class T>
4. inline T square(T x)
5. {
6. T result;
7. result = x \* x;
8. return result;
9. };
10. template <>
11. string square<string>(string ss)
12. {
13. return (ss+ss);
14. };
15. int main()
16. {
17. int i = 4, ii;
18. string ww("A");
19. ii = square<int>(i);
20. cout << i << ii;
21. cout << square<string>(ww) << endl;
22. }

a) 416AA  
b) 164AA  
c) AA416  
d) AA41A

Answer: a  
Explanation: In this program, We are using two template to calculate the square and to find the addition.  
Output:

$ g++ tem.cpp

$ a.out

416AA

5. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. template <typename T, typename U>
4. void squareAndPrint(T x, U y)
5. {
6. cout << x << x \* x << endl;
7. cout << y << " " << y \* y << endl;
8. };
9. int main()
10. {
11. int ii = 2;
12. float jj = 2.1;
13. squareAndPrint<int, float>(ii, jj);
14. }

a) 23

2.1 4.41

b) 24

2.1 4.41

c)24

2.1 3.41

d) 2.1 3.41

Answer: b  
Explanation: In this multiple templated types, We are passing two values of different types and producing the result.  
Output:

$ g++ tem1.cpp

$ a.out

24

2.1 4.41

6. What will be the output of the following C++ code?

1. #include <iostream>
2. #include <string>
3. using namespace std;
4. template<typename T>
5. void print\_mydata(T output)
6. {
7. cout << output << endl;
8. }
9. int main()
10. {
11. double d = 5.5;
12. string s("Hello World");
13. print\_mydata( d );
14. print\_mydata( s );
15. return 0;
16. }

a) 5.5

Hello World

b) 5.5  
c) Hello World  
d) Hello

Answer: a  
Explanation: In this program, We are passing the value to the template and printing it in the template.  
Output:

$ g++ tem2.cpp

$ a.out

5.5

Hello World

7. How many types of templates are there in c++?  
a) 1  
b) 2  
c) 3  
d) 4

Answer: b  
Explanation: There are two types of templates. They are function template and class template.

8. Which are done by compiler for templates?  
a) type-safe  
b) portability  
c) code elimination  
d) prototype

Answer: a  
Explanation: The compiler can determine at compile time whether the type associated with a template definition can perform all of the functions required by that template definition.

9. What may be the name of the parameter that the template should take?  
a) same as template  
b) same as class  
c) same as function  
d) same as member

Answer: a  
Explanation: The name of the parameter that the template should take same as the template.

10. How many parameters are legal for non-type template?  
a) 1  
b) 2  
c) 3  
d) 4

Answer: d  
Explanation: The following are legal for non-type template parameters: integral or enumeration type, Pointer to object or pointer to function, Reference to object or reference to function, Pointer to member.

# C++ Programming Questions and Answers – Function Templates – 1

1. What is a function template?  
a) creating a function without having to specify the exact type  
b) creating a function with having an exact type  
c) creating a function without having blank spaces  
d) creating a function without class

Answer: a  
Explanation: Function template is used to create a function without having to specify the exact type.

2. Which is used to describe the function using placeholder types?  
a) template parameters  
b) template type parameters  
c) template type  
d) type parameters

Answer: b  
Explanation: During runtime, We can choose the appropriate type for the function and it is called as template type parameters.

3. Pick out the correct statement.  
a) you only need to write one function, and it will work with many different types  
b) it will take a long time to execute  
c) duplicate code is increased  
d) it will take a long time to execute & duplicate code is increased

Answer: a  
Explanation: Because of template type parameters, It will work with many types and saves a lot of time.

4. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. template<typename type>
4. type Max(type Var1, type Var2)
5. {
6. return Var1 > Var2 ? Var1:Var2;
7. }
8. int main()
9. {
10. int p;
11. p = Max(100, 200);
12. cout << p << endl;
13. return 0;
14. }

a) 100  
b) 200  
c) 300  
d) 100200

Answer: b  
Explanation: In this program, We are returning the maximum value by using function template.  
Output:

$ g++ ftemp.cpp

$ a.out

200

5. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. template<typename type>
4. class Test
5. {
6. public:
7. Test()
8. {
9. };
10. ~Test()
11. {
12. };
13. type Funct1(type Var1)
14. {
15. return Var1;
16. }
17. type Funct2(type Var2)
18. {
19. return Var2;
20. }
21. };
22. int main()
23. {
24. Test<int> Var1;
25. Test<float> Var2;
26. cout << Var1.Funct1(200) << endl;
27. cout << Var2.Funct2(3.123) << endl;
28. return 0;
29. }

a) 200

3.123

b) 3.123

200

c) 200  
d) 3.123

Answer: a  
Explanation: In this program, We are passing the values and getting it back from template. And we are using the constructor and destructor for the function template.  
Output:

$ g++ ftemp1.cpp

$ a.out

200

3.123

6. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. template<typename type>
4. class TestVirt
5. {
6. public:
7. virtual type TestFunct(type Var1)
8. {
9. return Var1 \* 2;
10. }
11. };
12. int main()
13. {
14. TestVirt<int> Var1;
15. cout << Var1.TestFunct(100) << endl;
16. return 0;
17. }

a) 100  
b) 200  
c) 50  
d) 150

Answer: b  
Explanation: In this program, We are using class to pass the value and then we are manipulating it.  
Output:

$ g++ ftemp3.cpp

$ a.out

200

7. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. template<typename T>
4. inline T square(T x)
5. {
6. T result;
7. result = x \* x;
8. return result;
9. };
10. int main()
11. {
12. int i, ii;
13. float x, xx;
14. double y, yy;
15. i = 2;
16. x = 2.2;
17. y = 2.2;
18. ii = square(i);
19. cout << i << " " << ii << endl;
20. yy = square(y);
21. cout << y << " " << yy << endl;
22. }

a) 2 4

2.2 4.84

b) 2 4  
c) error  
d) 3 6

Answer: a  
Explanation: In this program, We are passing the values and calculating the square of the value by using the function template.  
Output:

$ g++ ftemp4.cpp

$ a.out

2 4

2.2 4.84

8. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. template<typename T>
4. void loopIt(T x)
5. {
6. int count = 3;
7. T val[count];
8. for (int ii=0; ii < count; ii++)
9. {
10. val[ii] = x++;
11. cout << val[ii] << endl;
12. }
13. };
14. int main()
15. {
16. float xx = 2.1;
17. loopIt(xx);
18. }

a) 2.1  
b) 3.1  
c)2.1

3.1

4.1

d) 3.2

Answer: c  
Explanation: In this program, We are using the for loop to increment the value by 1 in the function template.  
Output:

$ g++ ftemp5.cpp

$ a.out

2.1

3.1

4.1

9. What can be passed by non-type template parameters during compile time?  
a) int  
b) float  
c) constant expression  
d) string

Answer: c  
Explanation: Non-type template parameters provide the ability to pass a constant expression at compile time. The constant expression may also be an address of a function, object or static class member.

10. From where does the template class derived?  
a) regular non-templated C++ class  
b) templated class  
c) regular non-templated C++ class or templated class  
d) main function

Answer: c  
Explanation: Class derived template is derived from regular non-templated C++ class or templated class.

# C++ Programming Questions and Answers – Function Templates – 2

1. What are Templates in C++?  
a) A feature that allows the programmer to write generic programs  
b) A feature that allows the programmer to write specific codes for a problem  
c) A feature that allows the programmer to make program modular  
d) A feature that does not add any power to the language

Answer: a  
Explanation: Templates are features in C++ that allows the programmer to write generic programs. for example, making the same function to take different types of arguments and perform the same action on them without specifying the type in the argument list.

2. In how many ways templates concept can be used?  
a) 1  
b) 2  
c) 3  
d) 4

Answer: b  
Explanation: Template concept can be used in two different ways. They are function templates used with functions and class templates used with classes.

3. What is the difference between normal function and template function?  
a) The normal function works with any data types whereas template function works with specific types only  
b) Template function works with any data types whereas normal function works with specific types only  
c) Unlike a normal function, the template function accepts a single parameter  
d) Unlike the template function, the normal function accepts more than one parameters

Answer: b  
Explanation: As a template feature allows you to write generic programs. therefore a template function works with any type of data whereas normal function works with the specific types mentioned while writing a program. Both normal and template function accepts any number of parameters.

4. Templates simulate which of the following feature?  
a) Polymorphism  
b) Abstraction  
c) Encapsulation  
d) Inheritance

Answer: a  
Explanation: Template function helps in writing functions that work with different types of parameters which is what polymorphism means i.e. using same function prototype to perform the same operations on different types of parameters.

5. Which keyword is used for the template?  
a) Template  
b) template  
c) Temp  
d) temp

Answer: b  
Explanation: C++ uses template reserved keyword for defining templates.

6. What is the correct syntax of defining function template/template functions?  
a) template <class T> void(T a){cout<<a;}  
b) Template <class T> void(T a){cout<<a;}  
c) template <T> void(T a){cout<<a;}  
d) Template <T> void(T a){cout<<a;}

Answer: a  
Explanation: Starts with keyword template and then <class VAR>, then use VAR as type anywhere in the function below.

7. What does this template function indicates?

==================

template<class T>

void func(T a)

{

cout<<a;

}

==================

a) A function taking a single generic parameter and returning a generic type  
b) A function taking a single generic parameter and returning nothing  
c) A function taking single int parameter and returning a generic type  
d) A function taking a single generic parameter and returning a specific non-void type

Answer: b  
Explanation: As the return type of function is void therefore function is not returning anything. Now as the function is taking a template T as its argument which is a general type, therefore, it is accepting a single general type argument.

8. What does this template function indicates?

==================

template<class T>

T func(T a)

{

cout<<a;

}

==================

a) A function taking a single generic parameter and returning a generic type  
b) A function taking a single generic parameter and returning nothing  
c) A function taking single int parameter and returning a generic type  
d) A function taking a single generic parameter and returning a specific non-void type

Answer: a  
Explanation: As the return type of function is template T, therefore, the function is returning a general type. Now as the function is taking a template T as its argument which is a general type, therefore, it is accepting a single general type argument.

9. What does this template function indicates?

==================

template<class T, class U>

U func(T a, U b)

{

cout<<a<<"**\t**"<<b;

}

==================

a) A function taking a single generic parameter and returning a generic type which may be different from argument type  
b) A function taking a single generic parameter and returning a generic type which must be different from argument type  
c) A function taking a single generic parameter and returning a generic type which must have the same type as argument type  
d) A function taking a single generic parameter and returning a specific non-void type

Answer: a  
Explanation: As the return type of function is template U, therefore, the function is returning a general type. Now as the function is taking a template T as its argument which is a general type, therefore, it is accepting a single general type argument. But as U and T are different therefore return type and argument type may be the same or different. Same if U and T both have the same type and different if both have different types.

10. What will be the output of the following C++ code?

#include <iostream>

#include <string>

#include <cstdlib>

using namespace std;

template<class T>

T func(T a)

{

cout<<a;

return a;

}

template<class U>

void func(U a)

{

cout<<a;

}

int main(int argc, char const \*argv[])

{

int a = 5;

int b = func(a);

return 0;

}

a) 5  
b) 55  
c) Error  
d) Segmentation fault

Answer: c  
Explanation: Template function cannot be overloaded as done in this program. They can be overloaded with normal functions or other templates.

11. What will be the output of the following C++ code?

#include <iostream>

#include <string>

#include <cstdlib>

using namespace std;

template<class T>

T func(T a)

{

cout<<a;

return a;

}

int func(int a)

{

cout<<a;

}

int main(int argc, char const \*argv[])

{

int a = 5;

int b = func(a);

float c = func(5.5);

return 0;

}

a) 5  
b) 55.5  
c) Error  
d) Segmentation fault

Answer: b  
Explanation: C++ allows to overload template functions with normal functions therefore the program does not gives any error. When program is executed then int b = func(a); calls the normal function whereas float c = func(5.5) calls the template function.

12. What will be the output of the following C++ code?

#include <iostream>

#include <string>

#include <cstdlib>

using namespace std;

template<class T>

T func(T a)

{

return a;

}

template<class T, class U>

T func(U a)

{

return (T)a;

}

int main(int argc, char const \*argv[])

{

int a = 5;

int b = func(a);

int c = func(5.5);

cout<<b<<c<<endl;

return 0;

}

a) 5  
b) 55  
c) Error  
d) Segmentation fault

Answer: b  
Explanation: C++ allows such overlaoding of template functions therefore the program does not gives any error. The first template function returns the same parameter, whereas second template function returns type casted value.

13. Write a template function which takes two numbers, can be integers/real, and an operator as a character ‘+’/’-‘ or as boolean values true/false(representing ‘+’/’-‘ respectively) and returns the results accordingly.  
a)

template<class T, class U>

T func(T a, T b, U c)

{

if(c == '+' || c){

return a+b;

}

else if(c == '-' || !c){

return a-b;

}

}

b)

template<class T>

T func(T a, T b, T c)

{

if(c == '+' || c){

return a+b;

}

else if(c == '-' || !c){

return a-b;

}

}

c)

template<class T, class U>

U func(T a, T b, U c)

{

if(c == '+' || !c){

return a+b;

}

else if(c == '-' || c){

return a-b;

}

}

d)

template<class T, class U>

U func(T a, T b, U c)

{

if(c == '+' || c){

return a+b;

}

else if(c == '-' || !c){

return a-b;

}

}

Answer: a  
Explanation: We need two templates one for handling numbers and other for operators. Here T handles numbers and U handles operators. T can take any number type, whereas U can take either char or bool type. The function takes numbers and operators and produces the result. The return type should be a number, therefore, the return type is T.

14. What type can be used to replace templates from this function?

template<class T, class U>

T func(T a, T b, U c)

{

if(c == '+' || c){

return a+b;

}

else if(c == '-' || !c){

return a-b;

}

}

a) replace templates T, U with auto keyword  
b) replace templates T, U with generic keyword  
c) replace templates T, U with temp keyword  
d) replace templates T, U with GEN\_TEMP keyword

Answer: a  
Explanation: C++ allows use of auto keyword which helps in binding types dynamically. The modified without template version of above program is given below:

========================================

auto func(auto a, auto b, auto c)

{

if(c == '+' || c){

return a+b;

}

else if(c == '-' || !c){

return a-b;

}

}

========================================

15. What is the syntax of an explicit call for a template? Assume the given template function.

template<class T, class U>

void func(T a, U b)

{

cout<<a<<"**\t**"<<b<<endl;

}

a) func<int,char>(3,’a’);  
b) func(3,’a’)<int,char>;  
c) <int,char>func(3,’a’);  
d) func(<int>3,<char>’a’);

Answer: a  
Explanation: This is the correct method of explicit template function call.

**C++ Programming Questions and Answers – Class Templates**

1. What is the syntax of class template?  
a) template <paramaters> class declaration  
b) Template <paramaters> class declaration  
c) temp <paramaters> class declaration  
d) Temp <paramaters> class declaration

Answer: a  
Explanation: Syntax involves template keyword followed by list of parameters in angular brackets and then class declaration. As follows template <paramaters> class declaration;

2. What will be the output of the following C++ code?

#include <iostream>

#include <string>

#include <cstdlib>

using namespace std;

template<class T>

class A

{

public:

A(){

cout<<"Created**\n**";

}

~A(){

cout<<"Destroyed**\n**";

}

};

int main(int argc, char const \*argv[])

{

A a;

return 0;

}

a)Created

Destroyed

b)Destroyed

Created

c) Compile-time error  
d) Run-time error

Answer: c  
Explanation: As class A is a template class and a template class during object declaration requires template arguments, therefore A a; gives an error.

3. What will be the output of the following C++ code?

#include <iostream>

#include <string>

#include <cstdlib>

using namespace std;

template<class T>

class A

{

public:

A(){

cout<<"Created**\n**";

}

~A(){

cout<<"Destroyed**\n**";

}

};

int main(int argc, char const \*argv[])

{

A <int>a;

return 0;

}

a)Created

Destroyed

b)Destroyed

Created

c) Compile-time error  
d) Run-time error

Answer: a  
Explanation: In this program object of template class has an argument, therefore, the program does not give any error. Hence the program runs perfectly and the output is produced.  
Output:

$ ./a.out

Created

Destroyed

4. What will be the output of the following C++ code?

#include <iostream>

#include <string>

#include <cstdlib>

using namespace std;

template<class T>

class A

{

public:

A(){

cout<<"Created**\n**";

}

~A(){

cout<<"Destroyed**\n**";

}

};

int main(int argc, char const \*argv[])

{

A <int>a1;

A <char>a2;

A <float>a3;

return 0;

}

a)Created

Destroyed

Created

Destroyed

Created

Destroyed

b)Created

Created

Created

Destroyed

Destroyed

Destroyed

c)Destroyed

Created

Destroyed

Created

Destroyed

Created

d)Destroyed

Destroyed

Destroyed

Created

Created

Created

Answer: b  
Explanation: All the objects are first created and then all are destroyed when the scope of all the objects are destroyed i.e. at the end of the main function. That’s why first all the created are printed and then after that, all the destroyed are printed.

5. What will be the output of the following C++ code?

#include <iostream>

#include <string>

#include <cstdlib>

using namespace std;

template<class T>

class A

{

public:

T func(T a, T b){

return a/b;

}

};

int main(int argc, char const \*argv[])

{

A <int>a1;

cout<<a1.func(3,2)<<endl;

cout<<a1.func(3.0,2.0)<<endl;

return 0;

}

a)1

1

b)1

1.5

c)1.5

1

d)1.5

1.5

Answer: a  
Explanation: As a1 object is defined with <int> as the template parameter therefore all the numbers passed to the func() are converted to integers and as integer division of 3 by 2 is 1 therefore the output is 1.

6. What will be the output of the following C++ code?

#include <iostream>

#include <string>

#include <cstdlib>

using namespace std;

template<class T>

class A

{

public:

T func(T a, T b){

return a/b;

}

};

int main(int argc, char const \*argv[])

{

A <float>a1;

cout<<a1.func(3,2)<<endl;

cout<<a1.func(3.0,2.0)<<endl;

return 0;

}

a)1

1

b)1

1.5

c)1.5

1

d)1.5

1.5

Answer: d  
Explanation: As a1 object is defined with <float> as the template parameter therefore all the numbers passed to the func() are converted to decimal floating point numbers and as floating division of 3 by 2 is 1.5 therefore the output is 1.5 for both 1st and 2nd call of the function.

7. What will be the output of the following C++ code?

#include <iostream>

#include <string>

#include <cstdlib>

using namespace std;

template<class T>

class A

{

public:

T func(T a, T b){

return a/b;

}

};

int main(int argc, char const \*argv[])

{

A <char>a1;

cout<<a1.func(65,1)<<endl;

cout<<a1.func(65.28,1.1)<<endl;

return 0;

}

a)A

A

b)65

65

c)A

65

d)65

A

Answer: a  
Explanation: As a1 object is defined with as the template parameter therefore all the numbers passed to the func() are converted to characters integers. So when 1st func() call is done with 65 and 1 then after dividing 65 by 1 gives 65 for which char equivalent is ‘A’ hence ‘A’ is printed similarly in 2nd case 65.28/1.1 = 65 integer which represents ‘A’ in char type therefore ‘A’ is printed.

8. What will be the output of the following C++ code?

#include <iostream>

#include <string>

#include <cstdlib>

using namespace std;

template<class T>

class A

{

T a;

public:

A(){}

~A(){}

};

int main(int argc, char const \*argv[])

{

A <char>a1;

A <int>a2;

A <double>a3;

cout<<sizeof(a1)<<endl;

cout<<sizeof(a2)<<endl;

cout<<sizeof(a3)<<endl;

return 0;

}

a)1

4

8

b)4

1

8

c)1

1

1

d)4

4

4

Answer: a  
Explanation: class A has a variable of type template type therefore the type of the class depends on the template type assigned. Therefore for char template the class size is 1. For int template type size is 4 and for double type size is 8.

9. How the template class is different from the normal class?  
a) Template class generate objects of classes based on the template type  
b) Template class saves system memory  
c) Template class helps in making genetic classes  
d) All of the mentioned

Answer: d  
Explanation: Size of the object of template class varies depending on the type of template parameter passed to the class. Due to which each object occupies different memories on system hence saving extra memories. Template class also helps in making generic classes.

10. What will be the output of the following C++ code?

#include <iostream>

#include <string>

#include <cstdlib>

using namespace std;

template<class T, class U = char>

class A

{

T a;

U b;

public:

A(T a\_val, char b\_val = '$'){

this->a = a\_val;

this->b = b\_val;

}

void print(){

cout<<a<<' '<<b<<endl;

}

};

int main(int argc, char const \*argv[])

{

A <int, int> a1(5,10);

A <int> a2(5);

A <float> a3(10.0);

return 0;

}

a)5 10

5 $

10 $

b) nothing  
c) Error  
d) Segmentation fault

Answer: b  
Explanation: The program is correct therefore no error or segmentation fault but as no print() function is called using any object therefore nothing is printed on the output console.

11. What will be the output of the following C++ code?

#include <iostream>

#include <string>

#include <cstdlib>

using namespace std;

template<class T, class U = char>

class A

{

T a;

U b;

public:

A(T a\_val, char b\_val = '$'){

this->a = a\_val;

this->b = b\_val;

}

void print(){

cout<<a<<' '<<b<<endl;

}

};

int main(int argc, char const \*argv[])

{

A <int, int> a1(5,10);

A <int> a2(5);

A <float> a3(10.0);

a1.print();

a2.print();

a3.print();

return 0;

}

a)5 10

5 $

10 $

b) Nothing  
c) Error  
d) Segmentation fault

Answer: a  
Explanation: As we have defined the default templates so when a1 is initilized with 5 and 10 as we have specified <int,int> as the parameters. For a2 5 and $ is initialized as we have only passed <int> as the template parameter. Similar goes for a3. Hence the output is printed as mentioned.

$ ./a.out

5 10

5 $

10 $

12. How many template parameters are allowed in template classes?  
a) 1  
b) 2  
c) 3  
d) one or more

Answer: d  
Explanation: Just like normal parameters we can pass more than one or more template parameters to a template class.

# C++ Programming Questions and Answers – Template Arguments to Specify Policy Usage

1. What is meant by the template parameter?  
a) It can be used to pass a type as an argument  
b) It can be used to evaluate a type  
c) It can of no return type  
d) It can be used to delete a type

Answer: a  
Explanation: A template parameter is a special kind of parameter that can be used to pass a type as argument.

2. Which keyword can be used in template?  
a) class  
b) typename  
c) both class & typename  
d) function

Answer: c  
Explanation: Both keywords can be used as shown below:  
template <class T> function declaration;  
template <typename T> function declaration;

3. What is the validity of template parameters?  
a) inside that block only  
b) inside the class  
c) whole program  
d) inside the main class

Answer: a  
Explanation: Template parameters are valid inside a block only i.e. they have block scope.

4. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. template <class T, int N>
4. class mysequence
5. {
6. T memblock [N];
7. public:
8. void setmember (int x, T value);
9. T getmember (int x);
10. };
11. template <class T, int N>
12. void mysequence<T,N> :: setmember (int x, T value)
13. {
14. memblock[x] = value;
15. }
16. template <class T, int N>
17. T mysequence<T,N> :: getmember (int x)
18. {
19. return memblock[x];
20. }
21. int main ()
22. {
23. mysequence <int, 5> myints;
24. mysequence <double, 5> myfloats;
25. myints.setmember (0, 100);
26. myfloats.setmember (3, 3.1416);
27. cout << myints.getmember(0) << '**\n**';
28. cout << myfloats.getmember(3) << '**\n**';
29. return 0;
30. }

a) 100  
b) 3.1416  
c)100

3.1416

d) 4.14

Answer: c  
Explanation: In this program, We are printing the integer in the first function and float in the second function.  
Output:

$ g++ farg.cpp

$ a.out

100

3.1416

5. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. template <class T>
4. T max (T& a, T& b)
5. {
6. return (a>b?a:b);
7. }
8. int main ()
9. {
10. int i = 5, j = 6, k;
11. long l = 10, m = 5, n;
12. k = max(i, j);
13. n = max(l, m);
14. cout << k << endl;
15. cout << n << endl;
16. return 0;
17. }

a) 6  
b)6

10

c)5

10

d) 5

Answer: b  
Explanation: In this program, We are using the ternary operator on the template function.  
Output:

$ g++ farg.cpp

$ a.out

6

10

6. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. template <class type>
4. class Test
5. {
6. public:
7. Test()
8. {
9. };
10. ~Test()
11. {
12. };
13. type Funct1(type Var1)
14. {
15. return Var1;
16. }
17. type Funct2(type Var2)
18. {
19. return Var2;
20. }
21. };
22. int main()
23. {
24. Test<int> Var1;
25. Test<double> Var2;
26. cout << Var1.Funct1(200);
27. cout << Var2.Funct2(3.123);
28. return 0;
29. }

a) 100  
b) 200  
c) 3.123  
d) 2003.123

Answer: d  
Explanation: In this program, We are passing the value and returning it from template.  
Output:

$ g++ farg3.cpp

$ a.out

2003.123

7. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. template <typename T, int count>
4. void loopIt(T x)
5. {
6. T val[count];
7. for(int ii = 0; ii < count; ii++)
8. {
9. val[ii] = x++;
10. cout << val[ii] << endl;
11. }
12. };
13. int main()
14. {
15. float xx = 2.1;
16. loopIt<float, 3>(xx);
17. }

a) 2.1  
b) 3.1  
c) 2.1

3.1

4.1

d) 4.1

Answer: c  
Explanation: In this program, We are using the non-type template parameter to increment the value in the function template.  
Output:

$ g++ farg4.cpp

$ a.out

2.1

3.1

4.1

8. Why we use :: template-template parameter?  
a) binding  
b) rebinding  
c) both binding & rebinding  
d) reusing

Answer: c  
Explanation: It is used to adapt a policy into binary ones.

9. Which parameter is legal for non-type template?  
a) pointer to member  
b) object  
c) class  
d) baseclass

Answer: a  
Explanation: The following are legal for non-type template parameters:integral or enumeration type, Pointer to object or pointer to function, Reference to object or reference to function, Pointer to member.

10. Which of the things does not require instantiation?  
a) functions  
b) non virtual member function  
c) member class  
d) all of the mentioned

Answer: d  
Explanation: The compiler does not generate definitions for functions, non virtual member functions, class or member class because it does not require instantiation.

# C++ Programming MCQ – Specialization

1. What is meant by template specialization?  
a) It will have certain data types to be fixed  
b) It will make certain data types to be dynamic  
c) Certain data types are invalid  
d) It will make all data types to be dynamic

Answer: a  
Explanation: In the template specialization, it will make the template to be specific for some data types.

2. Which is similar to template specialization?  
a) template  
b) function overloading  
c) function template overloading  
d) overloading

Answer: c  
Explanation: function template overloading is similar to template specialization.

3. Which is called on allocating the memory for the array of objects?  
a) destructor  
b) constructor  
c) method  
d) class

Answer: b  
Explanation: When you allocate memory for an array of objects, the default constructor must be called to construct each object. If no default constructor exists, you’re stuck needing a list of pointers to objects.

4. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. template <class T>
4. inline T square(T x)
5. {
6. T result;
7. result = x \* x;
8. return result;
9. };
10. template <>
11. string square<string>(string ss)
12. {
13. return (ss+ss);
14. };
15. int main()
16. {
17. int i = 2, ii;
18. string ww("A");
19. ii = square<int>(i);
20. cout << i << ": " << ii;
21. cout << square<string>(ww) << ":" << endl;
22. }

a) 2: 4AA:  
b) 2:4  
c) AA  
d) 2:4A

Answer: a  
Explanation: Template specialization is used when a different and specific implementation is to be used for a specific data type. In this program, We are using integer and character.  
Output:

$ g++ spec.cpp

$ a.out

2: 4AA:

5. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. template <typename T = float, int count = 3>
4. T multIt(T x)
5. {
6. for(int ii = 0; ii < count; ii++)
7. {
8. x = x \* x;
9. }
10. return x;
11. };
12. int main()
13. {
14. float xx = 2.1;
15. cout << xx << ": " << multIt<>(xx) << endl;
16. }

a) 2.1  
b) 378.228  
c) 2.1: 378.228  
d) 376

Answer: c  
Explanation: In this program, We specify the type in the template function. We need to compile this program by adding -std=c++0x.  
Output:

$ g++ -std=c++0x spec1.cpp

$ a.out

2.1: 378.228

6. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. template <class T>
4. class XYZ
5. {
6. public:
7. void putPri();
8. static T ipub;
9. private:
10. static T ipri;
11. };
12. template <class T>
13. void XYZ<T>::putPri()
14. {
15. cout << ipri++ << endl;
16. }
17. template <class T> T XYZ<T>::ipub = 1;
18. template <class T> T XYZ<T>::ipri = 1.2;
19. int main()
20. {
21. XYZ<int> a;
22. XYZ<float> b;
23. a.putPri();
24. cout << a.ipub << endl;
25. b.putPri();
26. }

a) 1  
b) 1.2  
c) 1

1.2

d)1

1

1.2

Answer: d  
Explanation: In this program, We are passing the value of specified type and printing it by specialization.  
Output:

$ g++ spec2.cpp

$ a.out

1

1

1.2

7. What will be the output of the following C++ code?

1. #include <iostream>
2. #include <string>
3. #include <cstring>
4. using namespace std;
5. template <class type>
6. type MyMax(const type Var1, const type Var2)
7. {
8. cout << "no specialization";
9. return Var1 < Var2 ? Var2 : Var1;
10. }
11. template <>
12. const char \*MyMax(const char \*Var1, const char \*Var2)
13. {
14. return (strcmp(Var1, Var2)<0) ? Var2 : Var1;
15. }
16. int main()
17. {
18. string Str1 = "class", Str2 = "template";
19. const char \*Var3 = "class";
20. const char \*Var4 = "template";
21. const char \*q = MyMax(Var3, Var4);
22. cout << q << endl;
23. return 0;
24. }

a) template  
b) class  
c) no specialization  
d) templateclass

Answer: a  
Explanation: In this program, We are computing the result in the specialized block of the program.  
Output:

$ g++ spec3.cpp

$ a.out

template

8. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. template<class T = float, int i = 5> class A
4. {
5. public:
6. A();
7. int value;
8. };
9. template<> class A<>
10. {
11. public: A();
12. };
13. template<> class A<double, 10>
14. {
15. public: A();
16. };
17. template<class T, int i> A<T, i>::A() : value(i)
18. {
19. cout << value;
20. }
21. A<>::A()
22. {
23. cout << "default";
24. }
25. A<double, 10>::A()
26. {
27. cout << "10" << endl;
28. }
29. int main()
30. {
31. A<int, 6> x;
32. A<> y;
33. A<double, 10> z;
34. }

a) 6  
b) 10  
c) 6default10  
d) 6default

Answer: c  
Explanation: In this program, We are defining three templates and specializing it and passing the values to it and printing it.  
Output:

$ g++ spec5.cpp

$ a.out

6default10

9. How many types of specialization are there in c++?  
a) 1  
b) 2  
c) 3  
d) 4

Answer: b  
Explanation: There are two types of specialization. They are full specialization and partial specialization.

10. What is another name of full specialization?  
a) explicit specialization  
b) implicit specialization  
c) function overloading template  
d) overloading template

Answer: a  
Explanation: explicit specialization is another name of full specialization.

**C++ Programming Questions and Answers – Derivation and Templates**

1. Which is dependant on template parameter?  
a) base class  
b) abstract class  
c) method  
d) static class

Answer: a  
Explanation: Base class is dependant on template parameter.

2. Which value is placed in the base class?  
a) derived values  
b) default type values  
c) both default type & derived values  
d) null value

Answer: b  
Explanation: We can place the default type values in a base class and overriding some of them through derivation.

3. How many bits of memory needed for internal representation of class?  
a) 1  
b) 2  
c) 4  
d) no memory needed

Answer: d  
Explanation: classes that contain only type members, nonvirtual function members, and static data members do not require memory at run time.

4. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. class class0
4. {
5. public:
6. virtual ~class0(){}
7. protected:
8. char p;
9. public:
10. char getChar();
11. };
12. class class1 : public class0
13. {
14. public:
15. void printChar();
16. };
17. void class1::printChar()
18. {
19. cout << "True" << endl;
20. }
21. int main()
22. {
23. class1 c;
24. c.printChar();
25. return 1;
26. }

a) True  
b) error  
c) no output  
d) runtime error

Answer: a  
Explanation: In this program, We are passing the values and inheriting it to the other class and printing the result.

$ g++ dert.cpp

$ a.out

True

5. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. template<typename T>class clsTemplate
4. {
5. public:
6. T value;
7. clsTemplate(T i)
8. {
9. this->value = i;
10. }
11. void test()
12. {
13. cout << value << endl;
14. }
15. };
16. class clsChild : public clsTemplate<char>
17. {
18. public:
19. clsChild(): clsTemplate<char>( 0 )
20. {
21. }
22. clsChild(char c): clsTemplate<char>( c )
23. {
24. }
25. void test2()
26. {
27. test();
28. }
29. };
30. int main()
31. {
32. clsTemplate <int> a( 42 );
33. clsChild b( 'A' );
34. a.test();
35. b.test();
36. return 0;
37. }

a) 42  
b) A  
c)42

A

d)A

42

Answer: c  
Explanation: In this program, We are passing the values by using the template inheritance and printing it.  
Output:

$ g++ dert.cpp

$ a.out

42

A

 6. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. template <class T>
4. class A
5. {
6. public:
7. A(int a): x(a) {}
8. protected:
9. int x;
10. };
11. template <class T>
12. class B: public A<char>
13. {
14. public:
15. B(): A<char>::A(100)
16. {
17. cout << x \* 2 << endl;
18. }
19. };
20. int main()
21. {
22. B<char> test;
23. return 0;
24. }

a) 100  
b) 200  
c) error  
d) runtime error

Answer: b  
Explanation: In this program, We are passing the values and manipulating it by using the template inheritance.  
Output:

$ g++ dert2.cpp

$ a.out

200

7. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. template <class type>
4. class Test
5. {
6. public:
7. Test();
8. ~Test();
9. type Data(type);
10. };
11. template <class type>
12. type Test<type>::Data(type Var0)
13. {
14. return Var0;
15. }
16. template <class type>
17. Test<type>::Test()
18. {
19. }
20. template <class type>
21. Test<type>::~Test()
22. {
23. }
24. int main(void)
25. {
26. Test<char> Var3;
27. cout << Var3.Data('K') << endl;
28. return 0;
29. }

a) k  
b) l  
c) error  
d) runtime error

Answer: a  
Explanation: In this program, We are passing the values and printing it by using template inheritance.  
Output:

$ g++ dert3.cpp

$ a.out

k

8. What will be the output of the following C++ code?

1. #include <iostream>
2. using namespace std;
3. class Base
4. {
5. public:
6. Base ( )
7. {
8. cout << "1" << endl;
9. }
10. ~Base ( )
11. {
12. cout << "2" << endl;
13. }
14. };
15. class Derived : public Base
16. {
17. public:
18. Derived ( )
19. {
20. cout << "3" << endl;
21. }
22. ~Derived ( )
23. {
24. cout << "4" << endl;
25. }
26. };
27. int main( )
28. {
29. Derived x;
30. }

a) 1234  
b) 4321  
c) 1423  
d) 1342

Answer: d  
Explanation: In this program, We are printing the order of execution of constructor and destructor in the class.  
Output:

$ g++ dert4.cpp

$ a.out

1342

9. How many kinds of entities are directly parameterized in c++?  
a) 1  
b) 2  
c) 3  
d) 4

Answer: c  
Explanation: C++ allows us to parameterize directly three kinds of entities through templates: types, constants, and templates.

10. How many kinds of parameters are there in C++?  
a) 1  
b) 2  
c) 3  
d) 5

Answer: c  
Explanation: There are three kinds of parameters are there in C++. They are type, non-type, template.

**C++ Programming Questions and Answers – Standard Template Library**

1. What is the Standard Template Library?  
a) Set of C++ template classes to provide common programming data structures and functions  
b) Set of C++ classes  
c) Set of Template functions used for easy data structures implementation  
d) Set of Template data structures only

Answer: a  
Explanation: STL expanded as Standard Template Library is set of C++ template classes to provide common programming data structures and functions.

2. Pick the correct statement.  
a) STL is a generalized library  
b) Components of STL are parameterized  
c) STL uses the concept of templates classes and functions to achieve generalized implementation  
d) All of the mentioned

Answer: d  
Explanation: STL is a generalized library and components of STL are parameterized. STL uses the concept of templates classes and function to achieve generalized implementation.

3. How many components STL has?  
a) 1  
b) 2  
c) 3  
d) 4

Answer: d  
Explanation: STL has four components namely Algorithms, Containers, Functors and Iterators.

4. What are the containers?  
a) Containers store objects and data  
b) Containers stores all the algorithms  
c) Containers contain overloaded functions  
d) Containers contain set of Iterators

Answer: a  
Explanation: Containers is a component of STL which stores objects and data.

5. In how many categories, containers are divided?  
a) 1  
b) 2  
c) 3  
d) 4

Answer: d  
Explanation: Containers are divided into 4 categories namely Sequence Containers, Associative Containers, Unordered Associative Containers and Container Adaptors.

6. What are the Sequence Containers?  
a) Containers that implements data structures which can be accessed sequentially  
b) Containers that implements sorted data structures for fast search in O(logn)  
c) Containers that implements unsorted(hashed) data structures for quick search in O(1)  
d) Containers that implements data structures which can be accessed non-sequentially

Answer: a  
Explanation: Sequence Containers is the subset of Containers that implements data structures which can be accessed sequentially.

7. How many Sequence Containers are provided by C++?  
a) 2  
b) 3  
c) 4  
d) 5

Answer: d  
Explanation: C++ provides 5 types of Sequence Containers namely array, vector, deque, forward\_list and list.

8. What are the Associative Containers?  
a) Containers that implements data structures which can be accessed sequentially  
b) Containers that implements sorted data structures for fast search in O(logn)  
c) Containers that implements unsorted(hashed) data structures for quick search in O(1)  
d) Containers that implements data structures which can be accessed non-sequentially

Answer: b  
Explanation: Associative Containers is the subset of Containers that implements sorted data structures for fast search in O(logn).

9. How many Associative Containers are provided by C++?  
a) 2  
b) 3  
c) 4  
d) 5

Answer: c  
Explanation: C++ provides 4 types of Associative Containers namely Set, Map, multiset and multimap.

10. What are Unordered Associative Containers?  
a) Containers that implements data structures which can be accessed sequentially  
b) Containers that implements sorted data structures for fast search in O(logn)  
c) Containers that implements unsorted(hashed) data structures for quick search in O(1)  
d) Containers that implements data structures which can be accessed non-sequentially

Answer: c  
Explanation: Unordered Associative Containers is the subset of Containers that implements unsorted(hashed) data structures for quick search in O(1) amortized O(n) Worst case complexity.

11. What are Container Adaptors?  
a) Containers that implements data structures which can be accessed sequentially  
b) Containers that implements sorted data structures for fast search in O(logn)  
c) Containers that implements unsorted(hashed) data structures for quick search in O(1)  
d) Containers that provide a different interface for sequential containers

Answer: d  
Explanation: Container Adaptors is the subset of Containers that provides a different interface for sequential containers.

12. How many Container Adaptors are provided by C++?  
a) 2  
b) 3  
c) 4  
d) 5

Answer: b  
Explanation: C++ provides 3 types of Container Adaptors namely Stack, Queue and Priority Queues.

13. What are Iterators?  
a) Iterators are used to iterate over C-like arrays  
b) Iterators are used to iterate over pointers  
c) Iterators are used to point memory addresses of STL containers  
d) Iterators are used to iterate over functions

Answer: c  
Explanation: In C++, Iterators are provided to iterate over the STL containers.

14. How many types of Iterators are provided by C++?  
a) 2  
b) 3  
c) 4  
d) 5

Answer: d  
Explanation: There are five types of Iterators provided by C++ namely Input Iterators, Output Iterators, Forward Iterators, Bi-directional Iterators and Random-access Iterators.

15. Which header file is used for Iterators?  
a) <iterator>  
b) <algorithm>  
c) <iter>  
d) <loopIter>

Answer: a  
Explanation: Iterators are present inside the <iterator> header file so this header file is needed to use Iterators.

**C++ Programming MCQ – Templates**

1. Which of the following is used for generic programming?  
a) Virtual functions  
b) Modules  
c) Templates  
d) Abstract Classes

Answer: c  
Explanation: Templates are used for generic programming. They help in making generic functions and classes hence achieving the generic codes.

2. Which of the following is correct about templates?  
a) It is a type of compile time polymorphism  
b) It allows the programmer to write one code for all data types  
c) Helps in generic programming  
d) All of the mentioned

Answer: d  
Explanation: Templates are used for generic programming hence allowing to write a single function for all data types. It is a type of compile time polymorphism.

3. What will be the output of the following C++ code?

#include <iostream>

using namespace std;

template <typename T>

void fun(const T&x)

{

static int count = 0;

cout << "x = " << x << " count = " << count;

++count;

return;

}

int main()

{

fun<int> (1);

cout << endl;

fun<int>(1);

cout << endl;

fun<double>(1.1);

cout << endl;

return 0;

}

a)x = 1 count = 0

x = 1 count = 1

x = 1.1 count = 0

b)x = 1 count = 0

x = 1.0 count = 1.0

x = 1.1 count = 0

c)x = 1.0 count = 0.0

x = 1 count = 1

x = 1.1 count = 0

d)x = 1.0 count = 0.0

x = 1.0 count = 1.0

x = 1.1 count = 0.0

Answer: a  
Explanation: As template is a type of polymorphism so count becomes 1 for int type because we have called the function for int twice but as we have called it only once for double therefore value of count is 0 for double i.e. for last call.

4. What will be the output of the following C++ function?

#include <iostream>

using namespace std;

template <typename T>

T max(T x, T y)

{

return (x > y)? x : y;

}

int main()

{

cout << max(3, 7) << std::endl;

cout << max(3.0, 7.0) << std::endl;

cout << max(3, 7.0) << std::endl;

return 0;

}

a)7

7.0

7.0

b)7

7

7

c)7.0

7.0

7.0

d) Error

Answer: d  
Explanation: Here in the template function as both the argument have same type T but in third called of max function we are using 2 types int and float while calling the function hence the program gives error.

5. What will be the output of the following C++ code?

#include <iostream>

using namespace std;

template <class T>

class Test

{

private:

T val;

public:

static int count;

Test() {

count++;

}

};

template<class T>

int Test<T>::count = 0;

int main()

{

Test<int> a;

Test<int> b;

Test<double> c;

cout << Test<int>::count << endl;

cout << Test<double>::count << endl;

return 0;

}

a)2

1

b)1

2

c)1

1

d)2

2

Answer: a  
Explanation: In this as we have first used int to define the objects of class Test so count is increased to 2 for int types and declared the double object once therefore the constructor is called only once hence value of count is 1.

6. What will be the output of the following C++ code?

#include<iostream>

#include<stdlib.h>

using namespace std;

template<class T, class U>

class A

{

T x;

U y;

};

int main()

{

A<char, char> a;

A<int, int> b;

cout << sizeof(a) << endl;

cout << sizeof(b) << endl;

return 0;

}

a)2

8

b)8

2

c)1

4

d)4

1

Answer: a  
Explanation: In this we have made class with two template arguments which are used in defining one variable of each type inside the class. So the total size of the class will depend on the type of U and T. So when we have defined the object with <char,char> type then the class have two variable of char hence making the size of class = 2 which in case of <int,int> = 8.

7. What will be the output of the following C++ code?

#include<iostream>

#include<stdlib.h>

using namespace std;

template<class T, class U, class V=double>

class A

{

T x;

U y;

V z;

};

int main()

{

A<int, int> a;

A<double, double> b;

cout << sizeof(a) << endl;

cout << sizeof(b) << endl;

return 0;

}

a)16

24

b)24

16

c) Error  
d) Segmentation fault

Answer: a  
Explanation: In this case the class has three variables of type T, U and V = double by default. Now as we have defined objects with types <int,int> so its size is 4+4+8 = 16 and second object of <double,double> which leads to 8+8+8 = 24.

8. What will be the output of the following C++ code?

#include <iostream>

using namespace std;

template <class T, int max>

int arrMin(T arr[], int n)

{

int m = max;

for (int i = 0; i < n; i++)

if (arr[i] < m)

m = arr[i];

return m;

}

int main()

{

int arr1[] = {10, 20, 15, 12};

int n1 = sizeof(arr1)/sizeof(arr1[0]);

char arr2[] = {1, 2, 3};

int n2 = sizeof(arr2)/sizeof(arr2[0]);

cout << arrMin<int, 10000>(arr1, n1) << endl;

cout << arrMin<char, 256>(arr2, n2);

return 0;

}

a)10

1

b)12

2

c)10000

256

d) Error

Answer: a  
Explanation: In this, we are passing arr and its size in the template function. Now during first call value of m becomes 10 as it is the minimum in the array. In second call the value of m becomes 1 which is minimum in 2nd array.

9. What will be the output of the following C++ code?

#include <iostream>

using namespace std;

template <class T>

T max (T &a, T &b)

{

cout << "Template Called ";

return (a > b)? a : b;

}

template <>

int max <int> (int &a, int &b)

{

cout << "Called ";

return (a > b)? a : b;

}

int main ()

{

int a = 10, b = 20;

cout << max <int> (a, b);

}

a) Template Called 20  
b) Called 20  
c) Error  
d) Segmentation fault

Answer: b  
Explanation: For T = int we have created a separate definition for the above template function. Hence the call using int takes the newly defined function.

10. What will be the output of the following C++ code?

#include <iostream>

using namespace std;

template<int n>

struct funStruct

{

static const int val = 2\*funStruct<n-1>::val;

};

template<>

struct funStruct<0>

{

static const int val = 1 ;

};

int main()

{

cout << funStruct<10>::val << endl;

return 0;

}

a) 1  
b) 1024  
c) Error  
d) Segmentation fault

Answer: b  
Explanation: The above call for struct will call the first struct for n > 0 and second one when n = 0. Therefore when value of n = 10 the until n becomes 0 first struct is called so we will call 2\*2\*2…10 times\*1 which will give the result 210 = 1024.