**Simple**

1. The copy constructor would take a parameter by reference only
   1. **True** b. False
2. All method invocations in C++ by default exhibit late binding
   1. True b. **False**
3. To get polymorphism for a class you have to mark your methods as
   1. Static b. **Virtual**

c. Pure virtual d. Final

1. The default access scope for a method in a C++ class is
   1. **Private** b. Public

c. Protected d. Default

1. Where does memory get allocated for a static data members of a class
   1. Code/text b. Stack

c. Heap d. **Data**

1. If a dynamic cast fails
   1. It throws an exception b. **Returns a null value**

c. Converts to desired type d. Can never say

1. The operator used for getting the type\_info object is
   1. Typeof b. **Typeid**

c. Type d. Typeinfo

1. Serialization is the process of
   1. Converting bytes to objects
   2. **Converting objects to bytes**
   3. Converting bytes to classes
   4. Converting classes to bytes
2. A constructor can be marked as virtual
   1. True b. **False**
3. For the following allocation which would be the proper deallocation? int \*p = new int[5]
   1. Free(p) b.Delete p

c. **Delete [] p** d. None of the above

1. Template classes can be inherited
   1. **True** b. False
2. Which is the proper prototype for overloading the “>>” operator for a class like Cpoint
   1. istream operator>>(istream, CPoint);
   2. istream operator>>(istream&, CPoint);
   3. istream& operator>>(istream&, CPoint);
   4. **istream& operator>>(istream&, CPoint&);**
3. Namespaces
   1. Provide a logical grouping of objects
   2. **Provide a logical grouping of classes**
   3. Provide a physical grouping of objects
   4. Provide a physical grouping of classes
4. A class in C++ would be assumed as abstract if it has at least one virtual method
   1. True b. **False**

15) Which of the following is the base class of C++ steam class hierarchy?

1. istream
2. iostream
3. stream
4. **ios**
5. ostream

16) class Foo   
{   
   int i;   
};

In the above sample, what is the member access specifier of the member data "i"?

1. default
2. virtual
3. protected
4. **private**
5. public

17) References are allocated memory

1. **False**
2. True

18) Which of the following is the default namespace of C++?

1. iostream
2. standard
3. **std**
4. stdio

19) If ptr is a pointer to array of objects, then delete ptr and delete [] ptr both are same

1. **False**
2. True

20) What operator is prepended onto the member function name to indicate that the function is a destructor?

1. &
2. \*
3. **~**
4. ::
5. -

21) int function(char c = 'd');

Which one of the following is demonstrated by the sample code above?

1. **A default function parameter**
2. A virtual member function
3. A template function
4. A string assignment
5. A member function definition

22) Which one of the following statements is true about constructors and destructors?

1. Both explicitly declared constructors and explicitly declared destructors are required in a class.
2. Neither constructors nor destructors can take parameters.
3. In a given class, constructors are always required, but destructors are not.
4. **Constructors can take parameters, but destructors cannot.**
5. It is illegal to define either a constructor or a destructor as virtual.

23) A const object can access only const function

1. **True**
2. False

24) Select correct statement/s for destructor

1. **Destructor is called when object goes out of scope**
2. By default destructor is not provided by compiler
3. **Destructor can not be overloaded**
4. In case of inheritance base class destructor is called before derived class
5. **Destructors can be virtual**

25) class IntArrayRc : public IntArray;

What does the sequence of tokens ": public IntArray;" in the code above indicate?

1. It is the indicator that IntArray is derived from IntArrayRc class.
2. It is a scope resolution operator that states that IntArrayRc is a sub-class.
3. It is a scope resolution operator that states that IntArray is a super class.
4. **It is the indicator that IntArrayRc is derived from IntArray base class.**
5. It is the indicator for enforcing overloading of the IntArrayRc class from any IntArray class.

**Hard**

26) Copy constructor is called in case…

* 1. When an object is initialized using another object
  2. When object is passed to a function and collected in another object
  3. When object is returned from a function and collected in another object
  4. **All of the above**

27) class X {   
 int   i;   
  
protected:   
 float f;   
  
public:   
 char  c;   
};   
  
class Y : protected X { };

Referring to the sample code above, which one of the following data members are accessible from class Y?

1. c only
2. **f and c only**
3. i and c only
4. i and f only
5. i, f, and c

28) class A {   
public:   
   A();   
   void ~A();

}

class B : public A { };

What is WRONG with the class declarations above?

1. Class B must explicitly define a constructor.
2. **The destructor in "A" cannot have a void return type.**
3. Nothing is wrong with the code above.
4. Class B must define a destructor
5. "A" must provide a copy constructor in order for it to be used as a base class.

29) ) Given following class template

#include <iostream.h>

template<class t1,class t2>

class myclass

{

};

Write a statement which will direct a compiler to

1. generate this class for double and char respectively.
2. Create object of this class “m1” on stack.

\_\_**myclass<double,char> m1**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

30) what is the output ?

#include <iostream.h>

class base

{

public:

base()

{

cout<<"base def.\n";

disp();

}

void disp()

{

cout<<"\nbase disp\n";

}

};

class sub:public base

{

public:

sub()

{

cout<<"sub def\n";

}

void disp()

{

cout<<endl<<"in sub disp\n";

}

};

void main()

{

sub();

}

a) compilation error b) output “in sub def in base def in base disp” c) output “in base def in sub def in sub disp” **d) output “in base def in base disp in sub def”**

31) what is the output ?

#include <iostream.h>

class base

{

public:

base()

{

cout<<"base def.\n";

disp();

}

virtual void disp()=0;

};

class sub:public base

{

public:

sub()

{

cout<<"sub def\n";

}

void disp()

{

cout<<endl<<"in sub disp\n";

}

};

void main()

{

base \*b=new sub;

}

a**) linker error** b) compilation error c) output “in base def in sub def in sub disp” d) runtime error

32) what is the output?

#include<iostream.h>

class myclass

{

public:

void myclass()

{

cout<<endl<<"in myclass def\n";

}

myclass(int k)

{

cout<<endl<<"in param const\n";

}

};

void main()

{

myclass m1, m2(30);

}

a) output “ in param const “ b) output “in myclass def in param const” c) **compilation error** d) runtime error

33) what is the output ?

#include<iostream.h>

class base

{

public:

base()

{

cout<<"\nbase def\n";

}

void disp()

{

cout<<"base disp\n";

}

};

class sub:public base

{

public:

sub()

{

cout<<"sub def\n";

sub::disp();

}

};

void main()

{

sub s;

}

a) output “base def sub def” b) compilation error c) output “base def base disp sub def “ **d) output “base def sub def base disp “** e) compilation error “disp not available in sub”

**Medium**

* + - 1. The statements

int a=5;

cout<<"First"<<(a<<2)<<"Second";

Output will be

1. First52Second
2. Second25First
3. **First20Second**
4. An error message.

35) the following program segment

int a =10;

int const &b=a;

a=11

printf(“%d%d”,a,b);

* 1. Results in compile time error
  2. Results in run time error
  3. **1 1 1 1**
  4. None of the above.

36) what happens to the automatic objects that have been constructed in a try block when that block throws an exception ?

1. only throws exception
2. **Destructors are called for each of the objects**
3. same as for other variables.
4. None of the above.

37) In C++ programs the operation of the assignment operator and that of the copy constructor are

1. **similar except that the copy constructor creates a new object**
2. different except that they both copy member data.
3. both (1) and (2)
4. None of the above.

38) when two or more objects are derived from a common base class, u can prevent multiple copies of the base class from being present in an object derived from those objects by declaring base class when it is inherited.

1. public

2. protected

**3. virtual**

4. private

39) which one support unknown data types in a single framework ?

1. inheritance

2. virtual functions

3. abstract base class

**4. templates.**

40) #include<iostream.h>

class first

{

int a;

virtual void fun(){}

};

What is the size of the class ? (assume 16 bit architecture)

1. 1 byte
2. 2 byte
3. 3 byte
4. **4 byte**

41) int f()

{

int i=12;

int &r=i;

r+=r/4;

int \*p=&r;

\*p+=r;

return i;

}

Referring to the sample code above , what is the return value of the function “f()” ?

1. 15
2. **30**
3. 24
4. 12

42) what is the output ?

#include<stdio.h>

void main()

{

int x=4;

printf("%d",printf("%d%d",x,x) );

}

1. Garbage
2. **4,4,2**
3. 2,2,4
4. compile time error

43) what is the output?

#include<iostream.h>

class obj

{

public:

obj()

{

cout<<"in";

}

~obj()

{

cout<<"out";

}

};

void main()

{

obj A,B;

{

obj D;

}

obj E;

}

1. in in in in out out out out
2. **in in in out in out out out**
3. in in out out in in out out
4. in in out out in out in out

44) what is the output ?

#include<iostream.h>

void main()

{

int a=20;

int &n=a;

n=a++;

a=n++;

cout<<a<<"\t"<<n<<endl;

}

1. 20 20
2. 20 21
3. 21 22
4. **22 22**

45) what is the output ?

#include<iostream.h>

void main()

{

int arr[]={10,20,30,40,50};

int x,\*ptr1=arr,\*ptr2=&arr[3];

x=ptr2-ptr1;

cout<<x<<endl;

}

1. 6
2. **3**
3. compile time error
4. runtime error

46) #include<iostream.h>

class Base

{

int static i;

public:

Base()

{

}

};

class Sub1:public virtual Base

{

};

class Sub2:public Base

{

};

class Multi:public Sub1,public Sub2

{

};

void main()

{

Multi m;

}

In the above program, how many times Base class constructor will be called ?

1. 1
2. **2**
3. 3
4. None

47) what will happen

#include<iostream.h>

class name

{

public:

name()

{

cout<<endl<<"in def con\n";

}

name(name n)

{

cout<<endl<<"in copy con\n";

}

};

void main()

{

name n1;

name n2(n1);

}

1. output infinite “in copy con”
2. output “in def const in copy con”;
3. **compile error**
4. run time error.

48) Identify following a) const int \* ptr; \_\_\_**pointer to constant**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

b) int const \* str; \_\_\_\_ **pointer to constant** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

49) When child class object is assigned to parent class object it is called as \_**Object Slicing** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

50) True / False . We can’t do anything in source when converting from user defined to primitive type. **False**.