C++ interview Question

1.What is Boyce Codd Normal form?
A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F+ of the form a-> , where a and b is a subset of R, at least one of the following holds:
* a- > b is a trivial functional dependency (b is a subset of a)
* a is a superkey for schema R

2.What is virtual class and friend class?
Friend classes are used when two or more classes are designed to work together and need access to each other's implementation in ways that the rest of the world shouldn't be allowed to have. In other words, they help keep private things private. For instance, it may be desirable for class DatabaseCursor to have more privilege to the internals of class Database than main() has.

3.What is the word you will use when defining a function in base class to allow this function to be a polimorphic function?
virtual

4.What do you mean by binding of data and functions?
Encapsulation.

5.What are 2 ways of exporting a function from a DLL?
1.Taking a reference to the function from the DLL instance.
2. Using the DLL 's Type Library

6.What is the difference between an object and a class?
Classes and objects are separate but related concepts. Every object belongs to a class and every class contains one or more related objects.
– A Class is static. All of the attributes of a class are fixed before, during, and after the execution of a program. The attributes of a class don't change.
– The class to which an object belongs is also (usually) static. If a particular object belongs to a certain class at the time that it is created then it almost certainly will still belong to that class right up until the time that it is destroyed.
– An Object on the other hand has a limited lifespan. Objects are created and eventually destroyed. Also during that lifetime, the attributes of the object may undergo significant change.

7.Suppose that data is an array of 1000 integers. Write a single function call that will sort the 100 elements data [222] through data [321].
quicksort ((data + 222), 100);

8.What is a class?
Class is a user-defined data type in C++. It can be created to solve a particular kind of problem. After creation the user need not know the specifics of the working of a class.

9.What is friend function?
As the name suggests, the function acts as a friend to a class. As a friend of a class, it can access its private and protected members. A friend function is not a member of the class. But it must be listed in the class definition.

10.Which recursive sorting technique always makes recursive calls to sort subarrays that are about half size of the original array?

Mergesort always makes recursive calls to sort subarrays that are about half size of the original array, resulting in O(n log n) time.

11.What is abstraction?
Abstraction is of the process of hiding unwanted details from the user.

12.What are virtual functions?
A virtual function allows derived classes to replace the implementation provided by the base class. The compiler makes sure the replacement is always called whenever the object in question is actually of the derived class, even if the object is accessed by a base pointer rather than a derived pointer. This allows algorithms in the base class to be replaced in the derived class, even if users don't know about the derived class.

13.What is the difference between an external iterator and an internal iterator? Describe an advantage of an external iterator.
An internal iterator is implemented with member functions of the class that has items to step through. .An external iterator is implemented as a separate class that can be "attach" to the object that has items to step through. .An external iterator has the advantage that many difference iterators can be active simultaneously on the same object.

14.What is a scope resolution operator?
A scope resolution operator (::), can be used to define the member functions of a class outside the class.

15.What do you mean by pure virtual functions?
A pure virtual member function is a member function that the base class forces derived classes to provide. Normally these member functions have no implementation. Pure virtual functions are equated to zero.
class Shape { public: virtual void draw() = 0; };

16.What is polymorphism? Explain with an example?
"Poly" means "many" and "morph" means "form". Polymorphism is the ability of an object (or reference) to assume (be replaced by) or become many different forms of object.
Example: function overloading, function overriding, virtual functions. Another example can be a plus '+' sign, used for adding two integers or for using it to concatenate two strings.

17.What's the output of the following program? Why?

```
#include <stdio.h>
main()
{
typedef union
{
int a;
char b[10];
float c;
}
Union;

Union x,y = {100};
x.a = 50;
strcpy(x.b,\"hello\");
x.c = 21.50;

printf(\"Union x : %d %s %f \n\",x.a,x.b,x.c );
printf(\"Union y :%d %s%f \n\",y.a,y.b,y.c);
}
```

Given inputs X, Y, Z and operations | and & (meaning bitwise OR and AND, respectively)
What is output equal to in
output = (X & Y) | (X & Z) | (Y & Z)

18.Why are arrays usually processed with for loop?
The real power of arrays comes from their facility of using an index variable to traverse the array, accessing each element with the same expression a[i]. All the is needed to make this work is a iterated statement in which the variable i serves as a counter, incrementing from 0 to a.length -1. That is exactly what a loop does.

19.What is an HTML tag?
Answer: An HTML tag is a syntactical construct in the HTML language that abbreviates specific instructions to be executed when the HTML script is loaded into a Web browser. It is like a method in Java, a function in C++, a procedure in Pascal, or a subroutine in FORTRAN.

20.Explain which of the following declarations will compile and what will be constant – a pointer or the value pointed at: * const char *
*** char const ***
*** char * const**

Note: Ask the candidate whether the first declaration is pointing to a string or a single character. Both explanations are correct, but if he says that it's a single character pointer, ask why a whole string is initialized as char* in C++. If he says this is a string declaration, ask him to declare a pointer to a single character. Competent candidates should not have problems pointing out why const char* can be both a character and a string declaration, incompetent ones will come up with invalid reasons.

21.You're given a simple code for the class Bank Customer. Write the following functions:
*** Copy constructor**
*** = operator overload**
*** == operator overload**
*** + operator overload (customers' balances should be added up, as an example of joint account between husband and wife)**

Note:Anyone confusing assignment and equality operators should be dismissed from the interview. The applicant might make a mistake of passing by value, not by reference. The candidate might also want to return a pointer, not a new object, from the addition operator. Slightly hint that you'd like the value to be changed outside the function, too, in the first case. Ask him whether the statement customer3 = customer1 + customer2 would work in the second case.

22.What problems might the following macro bring to the application?
#define sq(x) x*x

23.Anything wrong with this code?
T *p = new T[10];
delete p;

Everything is correct, Only the first element of the array will be deleted", The entire array will be deleted, but only the first element destructor will be called.

24.Anything wrong with this code?
T *p = 0;
delete p;

Yes, the program will crash in an attempt to delete a null pointer.

25.How do you decide which integer type to use?
It depends on our requirement. When we are required an integer to be stored in 1 byte (means less than or equal to 255) we use short int, for 2 bytes we use int, for 8 bytes we use long int.

A char is for 1-byte integers, a short is for 2-byte integers, an int is generally a 2-byte or 4-byte integer (though not necessarily), a long is a 4-byte integer, and a long long is a 8-byte integer.

26.What does extern mean in a function declaration?
Using extern in a function declaration we can make a function such that it can used outside the file in which it is defined.

An extern variable, function definition, or declaration also makes the described variable or function usable by the succeeding part of the current source file. This declaration does not replace the definition. The declaration is used to describe the variable that is externally defined.

If a declaration for an identifier already exists at file scope, any extern declaration of the same identifier found within a block refers to that same object. If no other declaration for the identifier exists at file scope, the identifier has external linkage.

27.What can I safely assume about the initial values of variables which are not explicitly initialized?
It depends on complier which may assign any garbage value to a variable if it is not initialized.

28.What is the difference between char a[] = “string”; and char *p = “string”;?
In the first case 6 bytes are allocated to the variable a which is fixed, where as in the second case if *p is assigned to some other value the allocate memory can change.

29.What’s the auto keyword good for?
Answer1
Not much. It declares an object with automatic storage duration. Which means the object will be destroyed at the end of the objects scope. All variables in functions that are not declared as static and not dynamically allocated have automatic storage duration by default.

For example
```
int main()
{
int a; //this is the same as writing “auto int a;”
}
```

Answer2
Local variables occur within a scope; they are “local” to a function. They are often called automatic variables because they automatically come into being when the scope is entered and automatically go away when the scope closes. The keyword auto makes this explicit, but local variables default to auto auto auto auto so it is never necessary to declare something as an auto auto auto auto.

30.What is indirection?
In C when we use variable name to access the value it is direct access. If we use pointer to get the variable value, it is indirection.

31.What is the difference between char a[] = “string”; and char *p = “string”; ?
Answer1
```
a[] = “string”;
char *p = “string”;
```

The difference is this:
p is pointing to a constant string, you can never safely say
```
p[3]=’x’;
```
however you can always say a[3]=’x’;

char a[]=”string”; – character array initialization.
char *p=”string” ; – non-const pointer to a const-string.(this is permitted only in the case of char pointer in C++ to preserve backward compatibility with C.)

Answer2
a[] = “string”;
char *p = “string”;

a[] will have 7 bytes. However, p is only 4 bytes. P is pointing to an adress is either BSS or the data section (depending on which compiler — GNU for the former and CC for the latter).

Answer3
char a[] = “string”;
char *p = “string”;

for char a[]…….using the array notation 7 bytes of storage in the static memory block are taken up, one for each character and one for the terminating nul character.

But, in the pointer notation char *p……the same 7 bytes required, plus N bytes to store the pointer variable “p” (where N depends on the system but is usually a minimum of 2 bytes and can be 4 or more)……

32.How do I declare an array of N pointers to functions returning pointers to functions returning pointers to characters?
Answer1
If you want the code to be even slightly readable, you will use typedefs.
typedef char* (*functiontype_one)(void);
typedef functiontype_one (*functiontype_two)(void);
functiontype_two myarray[N]; //assuming N is a const integral

Answer2
char* (* (*a[N])())()
Here a is that array. And according to question no function will not take any parameter value.

33.What does extern mean in a function declaration?
It tells the compiler that a variable or a function exists, even if the compiler hasn’t yet seen it in the file currently being compiled. This variable or function may be defined in another file or further down in the current file.

34.How do I initialize a pointer to a function?
This is the way to initialize a pointer to a function
```
void fun(int a)
{

}

void main()
{
void (*fp)(int);
fp=fun;
fp(1);

}
```

35.How do you link a C++ program to C functions?
By using the extern “C” linkage specification around the C function declarations.

36.Explain the scope resolution operator.
It permits a program to reference an identifier in the global scope that has been hidden by another identifier with the same name in the local scope.

37.What are the differences between a C++ struct and C++ class?
The default member and base-class access specifier are different.

38.How many ways are there to initialize an int with a constant?
Two.
There are two formats for initializers in C++ as shown in the example that follows. The first format uses the traditional C notation. The second format uses constructor notation.
int foo = 123;
int bar (123);

39.How does throwing and catching exceptions differ from using setjmp and longjmp?
The throw operation calls the destructors for automatic objects instantiated since entry to the try block.

40,What is a default constructor?
Default constructor WITH arguments class B { public: B (int m = 0) : n (m) {} int n; }; int main(int argc, char *argv[]) { B b; return 0; }

41.What is a conversion constructor?
A constructor that accepts one argument of a different type.

42.What is the difference between a copy constructor and an overloaded assignment operator?
A copy constructor constructs a new object by using the content of the argument object. An overloaded assignment operator assigns the contents of an existing object to another existing object of the same class.

43.When should you use multiple inheritance?
There are three acceptable answers: “Never,” “Rarely,” and “When the problem domain cannot be accurately modeled any other way.”

44.Explain the ISA and HASA class relationships. How would you implement each in a class design?
A specialized class “is” a specialization of another class and, therefore, has the ISA relationship with the other class. An Employee ISA Person. This relationship is best implemented with inheritance. Employee is derived from Person. A class may have an instance of another class. For example, an employee “has” a salary, therefore the Employee class has the HASA relationship with the Salary class. This relationship is best implemented by embedding an object of the Salary class in the Employee class.

45.When is a template a better solution than a base class?
When you are designing a generic class to contain or otherwise manage objects of other types, when the format and behavior of those other types are unimportant to their containment or management, and particularly when those other types are unknown (thus, the generosity) to the designer of the container or manager class.

46.What is a mutable member?
One that can be modified by the class even when the object of the class or the member function doing the modification is const.

47.What is an explicit constructor?
A conversion constructor declared with the explicit keyword. The compiler does not use an explicit

constructor to implement an implied conversion of types. It's purpose is reserved explicitly for construction.

48.What is the Standard Template Library (STL)?
A library of container templates approved by the ANSI committee for inclusion in the standard C++ specification.
A programmer who then launches into a discussion of the generic programming model, iterators, allocators, algorithms, and such, has a higher than average understanding of the new technology that STL brings to C++ programming.

49.Describe run-time type identification.
The ability to determine at run time the type of an object by using the typeid operator or the dynamic_cast operator.

50.What problem does the namespace feature solve?
Multiple providers of libraries might use common global identifiers causing a name collision when an application tries to link with two or more such libraries. The namespace feature surrounds a library's external declarations with a unique namespace that eliminates the potential for those collisions. This solution assumes that two library vendors don't use the same namespace identifier, of course.

51.Are there any new intrinsic (built-in) data types?
Yes. The ANSI committee added the bool intrinsic type and its true and false value keywords.

52.Will the following program execute?

```
void main()
{
void *vptr = (void *) malloc(sizeof(void));
vptr++;
}
```

Answer1
It will throw an error, as arithmetic operations cannot be performed on void pointers.

Answer2
It will not build as sizeof cannot be applied to void* (error "Unknown size")

Answer3
How can it execute if it won't even compile? It needs to be int main, not void main. Also, cannot increment a void *.

Answer4
According to gcc compiler it won't show any error, simply it executes. but in general we can't do arthematic operation on void, and gives size of void as 1

Answer5
The program compiles in GNU C while giving a warning for "void main". The program runs without a crash. sizeof(void) is "1? hence when vptr++, the address is incremented by 1.

Answer6
Regarding arguments about GCC, be aware that this is a C++ question, not C. So gcc will compile and execute, g++ cannot. g++ complains that the return type cannot be void and the argument of sizeof() cannot be void. It also reports that ISO C++ forbids incrementing a pointer of type 'void*'.

Answer7
in C++

```
voidp.c: In function `int main()':
voidp.c:4: error: invalid application of `sizeof' to a void type
voidp.c:4: error: `malloc' undeclared (first use this function)
```

voidp.c:4: error: (Each undeclared identifier is reported only once for each function it appears in.)
voidp.c:6: error: ISO C++ forbids incrementing a pointer of type `void*'

But in c, it work without problems

53.void main()
{
char *cptr = 0?2000;
long *lptr = 0?2000;
cptr++;
lptr++;
printf(" %x %x", cptr, lptr);
}
Will it execute or not?
Answer1
For Q2: As above, won't compile because main must return int. Also, 0×2000 cannot be implicitly converted to a pointer (I assume you meant 0×2000 and not 0?2000.)

Answer2
Not Excute.
Compile with VC7 results following errors:
error C2440: 'initializing' : cannot convert from 'int' to 'char *'
error C2440: 'initializing' : cannot convert from 'int' to 'long *'

Not Excute if it is C++, but Excute in C.
The printout:
2001 2004

Answer3
In C++
[$]> g++ point.c
point.c: In function `int main()':
point.c:4: error: invalid conversion from `int' to `char*'
point.c:5: error: invalid conversion from `int' to `long int*'

in C
_____
[$] etc > gcc point.c
point.c: In function `main':
point.c:4: warning: initialization makes pointer from integer without a cast
point.c:5: warning: initialization makes pointer from integer without a cast
[$] etc > ./a.exe
2001 2004

54.What is the difference between Mutex and Binary semaphore?
semaphore is used to synchronize processes. where as mutex is used to provide synchronization between threads running in the same process.

55.In C++, what is the difference between method overloading and method overriding?
Overloading a method (or function) in C++ is the ability for functions of the same name to be defined as long as these methods have different signatures (different set of parameters). Method overriding is the ability of the inherited class rewriting the virtual method of the base class.

56.What methods can be overridden in Java?
In C++ terminology, all public methods in Java are virtual. Therefore, all Java methods can be overwritten in subclasses except those that are declared final, static, and private.

57.What are the defining traits of an object-oriented language?

The defining traits of an object-oriented langauge are:

* encapsulation
* inheritance
* polymorphism

58.Write a program that ask for user input from 5 to 9 then calculate the average

```
int main()
{
int MAX=4;
int total =0;
int average=0;
int numb;
cout<<“Please enter your input from 5 to 9”;
cin>>numb;
if((numb <5)&&(numb>9))
cout<<“please re type your input”;
else
for(i=0;i<=MAX; i++)
{
total = total + numb;
average= total /MAX;
}
cout<<“The average number is”<<average<<endl;

return 0;
}
```

59.Assignment Operator – What is the diffrence between a “assignment operator” and a “copy constructor”?

Answer1.

In assignment operator, you are assigning a value to an existing object. But in copy constructor, you are creating a new object and then assigning a value to that object. For example:

```
complex c1,c2;
c1=c2; //this is assignment
complex c3=c2; //copy constructor
```

Answer2.

A copy constructor is used to initialize a newly declared variable from an existing variable. This makes a deep copy like assignment, but it is somewhat simpler:

There is no need to test to see if it is being initialized from itself.
There is no need to clean up (eg, delete) an existing value (there is none).
A reference to itself is not returned.

60.RTTI – What is RTTI?

Answer1.

RTTI stands for “Run Time Type Identification”. In an inheritance hierarchy, we can find out the exact type of the objet of which it is member. It can be done by using:

1) dynamic id operator
2) typecast operator

Answer2.

RTTI is defined as follows: Run Time Type Information, a facility that allows an object to be queried

at runtime to determine its type. One of the fundamental principles of object technology is polymorphism, which is the ability of an object to dynamically change at runtime.

61.STL Containers – What are the types of STL containers?
There are 3 types of STL containers:

1. Adaptive containers like queue, stack
2. Associative containers like set, map
3. Sequence containers like vector, deque

62.What is the need for a Virtual Destructor ?
Destructors are declared as virtual because if do not declare it as virtual the base class destructor will be called before the derived class destructor and that will lead to memory leak because derived classâ€™s objects will not get freed.Destructors are declared virtual so as to bind objects to the methods at runtime so that appropriate destructor is called.

63.What is “mutable”?
Answer1.
“mutable” is a C++ keyword. When we declare const, none of its data members can change. When we want one of its members to change, we declare it as mutable.

Answer2.
A “mutable” keyword is useful when we want to force a “logical const” data member to have its value modified. A logical const can happen when we declare a data member as non-const, but we have a const member function attempting to modify that data member. For example:

```
class Dummy {
public:
bool isValid() const;
private:
mutable int size_ = 0;
mutable bool validStatus_ = FALSE;
// logical const issue resolved
};

bool Dummy::isValid() const
// data members become bitwise const
{
if (size > 10) {
validStatus_ = TRUE; // fine to assign
size = 0; // fine to assign
}
}
```

Answer2.
“mutable” keyword in C++ is used to specify that the member may be updated or modified even if it is member of constant object. Example:

```
class Animal {
private:
string name;
string food;
mutable int age;
public:
void set_age(int a);
};
```

```
void main() {
const Animal Tiger(â€™Fulffyâ€™,’antelopeâ€™,1);
Tiger.set_age(2);
// the age can be changed since its mutable
}
```

64.Differences of C and C++
Could you write a small program that will compile in C but not in C++ ?
In C, if you can a const variable e.g.
const int i = 2;
you can use this variable in other module as follows
extern const int i;
C compiler will not complain.

But for C++ compiler u must write
extern const int i = 2;
else error would be generated.

65.Bitwise Operations – Given inputs X, Y, Z and operations | and & (meaning bitwise OR and AND, respectively), what is output equal to in?
output = (X & Y) | (X & Z) | (Y & Z);

66.What is a modifier?
A modifier, also called a modifying function is a member function that changes the value of at least one data member. In other words, an operation that modifies the state of an object. Modifiers are also known as ‘mutators’. Example: The function mod is a modifier in the following code snippet:

```
class test
{
int x,y;
public:
test()
{
x=0; y=0;
}
void mod()
{
x=10;
y=15;
}
};
```

67.What is an accessor?
An accessor is a class operation that does not modify the state of an object. The accessor functions need to be declared as const operations

68.Differentiate between a template class and class template.
Template class: A generic definition or a parameterized class not instantiated until the client provides the needed information. It’s jargon for plain templates. Class template: A class template specifies how individual classes can be constructed much like the way a class specifies how individual objects can be constructed. It’s jargon for plain classes.

69.When does a name clash occur?
A name clash occurs when a name is defined in more than one place. For example., two different class libraries could give two different classes the same name. If you try to use many class libraries

at the same time, there is a fair chance that you will be unable to compile or link the program because of name clashes.

70.Define namespace.
It is a feature in C++ to minimize name collisions in the global name space. This namespace keyword assigns a distinct name to a library that allows other libraries to use the same identifier names without creating any name collisions. Furthermore, the compiler uses the namespace signature for differentiating the definitions.

71.What is the use of 'using' declaration. ?
A using declaration makes it possible to use a name from a namespace without the scope operator.

72.What is an Iterator class ?
A class that is used to traverse through the objects maintained by a container class. There are five categories of iterators: input iterators, output iterators, forward iterators, bidirectional iterators, random access. An iterator is an entity that gives access to the contents of a container object without violating encapsulation constraints. Access to the contents is granted on a one-at-a-time basis in order. The order can be storage order (as in lists and queues) or some arbitrary order (as in array indices) or according to some ordering relation (as in an ordered binary tree). The iterator is a construct, which provides an interface that, when called, yields either the next element in the container, or some value denoting the fact that there are no more elements to examine. Iterators hide the details of access to and update of the elements of a container class.
The simplest and safest iterators are those that permit read-only access to the contents of a container class.

73.What is an incomplete type?
Incomplete types refers to pointers in which there is non availability of the implementation of the referenced location or it points to some location whose value is not available for modification.

```
int *i=0x400 // i points to address 400
*i=0; //set the value of memory location pointed by i.
```

Incomplete types are otherwise called uninitialized pointers.

74.What is a dangling pointer?
A dangling pointer arises when you use the address of an object after its lifetime is over. This may occur in situations like returning addresses of the automatic variables from a function or using the address of the memory block after it is freed. The following code snippet shows this:

```
class Sample
{
public:
int *ptr;
Sample(int i)
{
ptr = new int(i);
}

~Sample()
{
delete ptr;
}
void PrintVal()
{
cout << "The value is " << *ptr;
```

```
}
};

void SomeFunc(Sample x)
{
cout << “Say i am in someFunc ” << endl;
}

int main()
{
Sample s1 = 10;
SomeFunc(s1);
s1.PrintVal();
}
```

In the above example when PrintVal() function is called it is called by the pointer that has been freed by the destructor in SomeFunc.

75.Differentiate between the message and method.

Message:

* Objects communicate by sending messages to each other.
* A message is sent to invoke a method.

Method

* Provides response to a message.
* It is an implementation of an operation.

76.What is an adaptor class or Wrapper class?

A class that has no functionality of its own. Its member functions hide the use of a third party software component or an object with the non-compatible interface or a non-object-oriented implementation.

77.What is a Null object?

It is an object of some class whose purpose is to indicate that a real object of that class does not exist. One common use for a null object is a return value from a member function that is supposed to return an object with some specified properties but cannot find such an object.

78.What is class invariant?

A class invariant is a condition that defines all valid states for an object. It is a logical condition to ensure the correct working of a class. Class invariants must hold when an object is created, and they must be preserved under all operations of the class. In particular all class invariants are both preconditions and post-conditions for all operations or member functions of the class.

79.What do you mean by Stack unwinding?

It is a process during exception handling when the destructor is called for all local objects between the place where the exception was thrown and where it is caught.

80.Define precondition and post-condition to a member function.

Precondition: A precondition is a condition that must be true on entry to a member function. A class is used correctly if preconditions are never false. An operation is not responsible for doing anything sensible if its precondition fails to hold. For example, the interface invariants of stack class say nothing about pushing yet another element on a stack that is already full. We say that isful() is a precondition of the push operation. Post-condition: A post-condition is a condition that must be true on exit from a member function if the precondition was valid on entry to that function. A class is implemented correctly if post-conditions are never false. For example, after pushing an element on

the stack, we know that isempty() must necessarily hold. This is a post-condition of the push operation.

81.What are the conditions that have to be met for a condition to be an invariant of the class?
* The condition should hold at the end of every constructor.
* The condition should hold at the end of every mutator (non-const) operation.

82.What are proxy objects?
Objects that stand for other objects are called proxy objects or surrogates.

```
template <class t="">
class Array2D
{
public:
class Array1D
{
public:
T& operator[] (int index);
const T& operator[] (int index)const;
};

Array1D operator[] (int index);
const Array1D operator[] (int index) const;
};
```

The following then becomes legal:

```
Array2D<float>data(10,20);
cout<<data[3][6]; // fine
```

Here data[3] yields an Array1D object and the operator [] invocation on that object yields the float in position(3,6) of the original two dimensional array. Clients of the Array2D class need not be aware of the presence of the Array1D class. Objects of this latter class stand for one-dimensional array objects that, conceptually, do not exist for clients of Array2D. Such clients program as if they were using real, live, two-dimensional arrays. Each Array1D object stands for a one-dimensional array that is absent from a conceptual model used by the clients of Array2D. In the above example, Array1D is a proxy class. Its instances stand for one-dimensional arrays that, conceptually, do not exist.

83.Name some pure object oriented languages.
Smalltalk, Java, Eiffel, Sather.

84.What is an orthogonal base class?
If two base classes have no overlapping methods or data they are said to be independent of, or orthogonal to each other. Orthogonal in the sense means that two classes operate in different dimensions and do not interfere with each other in any way. The same derived class may inherit such classes with no difficulty.

85.What is a node class?
A node class is a class that,
* relies on the base class for services and implementation,
* provides a wider interface to the users than its base class,
* relies primarily on virtual functions in its public interface
* depends on all its direct and indirect base class
* can be understood only in the context of the base class
* can be used as base for further derivation
* can be used to create objects.

A node class is a class that has added new services or functionality beyond the services inherited from its base class.

86.What is a container class? What are the types of container classes?

A container class is a class that is used to hold objects in memory or external storage. A container class acts as a generic holder. A container class has a predefined behavior and a well-known interface. A container class is a supporting class whose purpose is to hide the topology used for maintaining the list of objects in memory. When a container class contains a group of mixed objects, the container is called a heterogeneous container; when the container is holding a group of objects that are all the same, the container is called a homogeneous container.

87.How do you write a function that can reverse a linked-list?

Answer1:

```
void reverselist(void)
{
if(head==0)
return;
if(head-<next==0)
return;
if(head-<next==tail)
{
head-<next = 0;
tail-<next = head;
}
else
{
node* pre = head;
node* cur = head-<next;
node* curnext = cur-<next;
head-<next = 0;
cur-<next = head;

for(; curnext!=0; )
{
cur-<next = pre;
pre = cur;
cur = curnext;
curnext = curnext-<next;
}

curnext-<next = cur;
}
}
```

Answer2:

```
node* reverselist(node* head)
{
if(0==head || 0==head->next)
//if head->next ==0 should return head instead of 0;
return 0;

{
node* prev = head;
```

```
node* curr = head->next;
node* next = curr->next;

for(; next!=0; )
{
curr->next = prev;
prev = curr;
curr = next;
next = next->next;
}
curr->next = prev;

head->next = 0;
head = curr;
}

return head;
}
```

88.What is polymorphism?
Polymorphism is the idea that a base class can be inherited by several classes. A base class pointer can point to its child class and a base class array can store different child class objects.

89.How do you find out if a linked-list has an end? (i.e. the list is not a cycle)
You can find out by using 2 pointers. One of them goes 2 nodes each time. The second one goes at 1 nodes each time. If there is a cycle, the one that goes 2 nodes each time will eventually meet the one that goes slower. If that is the case, then you will know the linked-list is a cycle.

90.How can you tell what shell you are running on UNIX system?
You can do the Echo $RANDOM. It will return a undefined variable if you are from the C-Shell, just a return prompt if you are from the Bourne shell, and a 5 digit random numbers if you are from the Korn shell. You could also do a ps -l and look for the shell with the highest PID.

91.What is Boyce Codd Normal form?
A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F+ of the form a->b, where a and b is a subset of R, at least one of the following holds:

* a->b is a trivial functional dependency (b is a subset of a)
* a is a superkey for schema R

92.What is pure virtual function?
A class is made abstract by declaring one or more of its virtual functions to be pure. A pure virtual function is one with an initializer of = 0 in its declaration

93.Write a Struct Time where integer m, h, s are its members

```
struct Time
{
int m;
int h;
int s;
};
```

94.How do you traverse a Btree in Backward in-order?
Process the node in the right subtree
Process the root
Process the node in the left subtree

95.What is the two main roles of Operating System?
As a resource manager
As a virtual machine

96.In the derived class, which data member of the base class are visible?
In the public and protected sections.

97.Could you tell something about the Unix System Kernel?
The kernel is the heart of the UNIX openrating system, it's reponsible for controlling the computer's resouces and scheduling user jobs so that each one gets its fair share of resources.

98.What are each of the standard files and what are they normally associated with?
They are the standard input file, the standard output file and the standard error file. The first is usually associated with the keyboard, the second and third are usually associated with the terminal screen.

Q1. What is difference between C and C++ ?

Ans:

- C++ is **Multi-Paradigm**(not pure OOP, supports both procedural and object oriented) while C follows procedural style programming.
- In C data security is less, but in C++ you can use modifiers for your class members to make it inaccessible from outside.
- C follows top-down approach (solution is created in step by step manner, like each step is processed into details as we proceed) but C++ follows a bottom-up approach (where base elements are established first and are linked to make complex solutions).
- C++ supports function overloading while C does not support it.
- C++ allows use of functions in structures, but C does not permit that.
- **C++ supports reference variables**(two variables can point to same memory location). C does not support this.
- C does not have a built in exception handling framework, though we can emulate it with other mechanism. **C++ directly supports exception handling**, which makes life of developer easy.

Q2. What is a class?

Ans: [Probably this would be the first question for a Java/c++ technical interview for freshers and sometimes for experienced as well. Try to give examples when you answer this question.]

Class defines a datatype, it's type definition of category of thing(s). But a class actually does not define the data, it just specifies the structure of data. To use them you need to create objects out of the class. Class can be considered as a blueprint of a building, you can not stay inside blueprint of building, you need to construct building(s) out of that plan.

You can create any number of buildings from the blueprint, similarly you can create any number of objects from a class.

```
class Vehicle

{

public:

int numberOfTyres;

double engineCapacity;

void drive(){

// code to drive the car

}

};
```

Q3. What is an Object/Instance?

Ans: Object is the instance of a class, which is concrete. From the above example, we can create instance of class Vehicle as given below

```
Vehicle vehicleObject;
```

We can have different objects of the class Vehicle, for example we can have ***Vehicle*** objects with 2 tyres, 4tyres etc. Similarly different engine capacities as well.

Q4. What do you mean by C++ access specifiers ?

Ans: [Questions regarding access specifiers are common not just in c++ interview but for other object oriented language interviews as well.]

Access specifiers are used to define how the members (functions and variables) can be accessed outside the class. There are three access specifiers defined which are ***public, private, and protected***

- **private:**
  Members declared as private are accessible only with in the same class and they cannot be accessed outside the class they are declared.
- **public:**
  Members declared as public are accessible from any where.
- **protected:**
  Members declared as protected can not be accessed from outside the class except a child class. This access specifier has significance in the context of inheritance.

Q5. What are the basics concepts of OOPS?

Ans:

- **Classes and Objects**

  Refer Questions 2 and 3 for the concepts about classes and objects

- **Encapsulation**

  Encapsulation is the mechanism by which data and associated operations/methods are bound together and thus hide the data from outside world. It's also called data hiding. In c++, encapsulation achieved using the access specifiers (private, public and protected). Data members will be declared as private (thus protecting from direct access from outside) and public methods will be provided to access these data. Consider the below class

  ```
  class Person

  {

  private:

  int age;

  public:

  int getAge(){

  return age;

  }
  ```

```
int setAge(int value){

if(value > 0){

age = value;

}

}

};
```

In the class ***Person***, access to the data field age is protected by declaring it as *private* and providing ***public*** access methods. What would have happened if there was no access methods and the field age was ***public***? Anybody who has a ***Person*** object can set an invalid value (negative or very large value) for the age field. So by encapsulation we can preventing direct access from outside, and thus have complete control, protection and integrity of the data.

- **Data abstraction**

  Data abstraction refers to hiding the internal implementations and show only the necessary details to the outside world. In C++ data abstraction is implemented using interfaces and abstract classes.

```
class Stack

{

public:

virtual void push(int)=0;

virtual int pop()=0;

};

class MyStack : public Stack
```

```
{

private:

int arrayToHoldData[]; //Holds the data from stack

public:

void push(int) {

// implement push operation using array

}

int pop()

{

// implement pop operation using array

}

};
```

In the above example, the outside world only need to know about the ***Stack*** class and its ***push, pop*** operations. Internally stack can be implemented using arrays or linked lists or queues or anything that you can think of. This means, as long as the push and pop method performs the operations work as expected, you have the freedom to change the internal implementation with out affecting other applications that use your Stack class.

- **Inheritance**

  Inheritance allows one class to inherit properties of another class. In other words, inheritance allows one class to be defined in terms of another class.

```
class SymmetricShape

{
```

```
public:
int getSize()
{
return size;
}
void setSize(int w)
{
size = w;
}
protected:
int size;
};
// Derived class
class Square: public SymmetricShape
{
public:
int getArea()
{
```

```
return (size * size);

}

};
```

In the above example, class ***Square*** inherits the properties and methods of class ***SymmetricShape***. Inheritance is the one of the very important concepts in C++/OOP. It helps to modularise the code, improve reusability and reduces tight coupling between components of the system.

Q6. What is the use of volatile keyword in c++? Give an example.

Ans: Most of the times compilers will do optimization to the code to speed up the program. For example in the below code,

```
1. int a = 10;
2. while( a == 10){
3. // Do something
4. }
```

compiler may think that value of 'a' is not getting changed from the program and replace it with 'while(true)', which will result in an infinite loop. In actual scenario the value of 'a' may be getting updated from outside of the program.
Volatile keyword is used to tell compiler that the variable declared using volatile may be used from outside the current scope so that compiler wont apply any optimization. This matters only in case of multi-threaded applications.
In the above example if variable 'a' was declared using volatile, compiler will not optimize it. In shot, value of the volatile variables will be read from the memory location directly.

Q7. In how many ways we can initialize an int variable in C++?

Ans: In c++, variables can be initialized in two ways, the traditional C++ initialization using "=" operator and second using the constructor notation.

Traditional C++ initilization

- int i = 10;

variable i will get initialized to 10.

Using C++ constructor notation

- int i(10);

  Implicit conversions are performed when a type (say T) is used in a context where a compatible type (Say F) is expected so that the type T will be promoted to type F.

  short a = 2000 + 20;

  In the above example, variable a will get automatically promoted from short to int. This is called implicit conversion/coercion in c++.

Q8. What is implicit conversion/coercion in c++?

Ans: Implicit conversions are performed when a type (say T) is used in a context where a compatible type (Say F) is expected so that the type T will be promoted to type F.

short a = 2000 + 20;

In the above example, variable a will get automatically promoted from short to int. This is called implicit conversion/coercion in c++.

Q9. What are C++ inline functions?

Ans: C++ inline functions are special functions, for which the compiler replaces the function call with body/definition of function. Inline functions makes the program execute faster than the normal functions, since the overhead involved in saving current state to stack on the function call is avoided. By giving developer the control of making a function as inline, he can further optimize the code based on application logic. But actually, it's the compiler that decides whether to make a function inline or not regardless of it's declaration. Compiler may choose to make a non inline function inline and vice versa. Declaring a function as inline is in effect a request to the compiler to make it inline, which compiler may ignore. So, please note this point for the interview that, it is upto the compiler to make a function inline or not.

```
inline int min(int a, int b)

{

return (a < b)? a : b;

}
```

```
int main( )

{

cout << "min (20,10): " << min(20,10) << endl;

cout << "min (0,200): " << min(0,200) << endl;

cout << "min (100,1010): " << min(100,1010) << endl;

return 0;

}
```

If the complier decides to make the function min as inline, then the above code will internally look as if it was written like

```
int main( )

{

cout << "min (20,10): " << ((20 < 10)? 20 : 10) << endl;

cout << "min (0,200): " << ((0 < 200)? 0 : 200) << endl;

cout << "min (100,1010): " << ((100 < 1010)? 100 : 1010) << endl;

return 0;

}
```

Q10. What do you mean by translation unit in c++?

Ans: We organize our C++ programs into different source files (.cpp, .cxx etc). When you consider a source file, at the preprocessing stage, some extra content may get added to the source code (for example, the contents of header files included) and some content may get removed (for example, the part of the code in the #ifdef of #ifndef block which

resolve to false/0 based on the symbols defined). This effective content is called a translation unit. In other words, a translation unit consists of

- Contents of source file
- Plus contents of files included directly or indirectly
- Minus source code lines ignored by any conditional pre processing directives (the lines ignored by #ifdef,#ifndef etc)

Q11. What do you mean by internal linking and external linking in c++?

Ans: A symbol is said to be linked internally when it can be accessed only from with-in the scope of a single translation unit. By external linking a symbol can be accessed from other translation units as well. This linkage can be controlled by using static and extern keywords.

Q12. What do you mean by storage classes?

Ans: Storage class are used to specify the visibility/scope and life time of symbols(functions and variables). That means, storage classes specify where all a variable or function can be accessed and till what time those variables will be available during the execution of program.

Q13. How many storage classes are available in C++?

Ans: Storage class are used to specify the visibility/scope and life time of symbols(functions and variables). That means, storage classes specify where all a variable or function can be accessed and till what time those variables will be available during the execution of program.
Following storage classes are available in C++

- **auto**

It's the default storage class for local variables. They can be accessed only from with in the declaration scope. auto variables are allocated at the beginning of enclosing block and deallocated at the end of enclosing block.

```
void changeValue(void)

{

auto int i = 1 ;
```

```
i++;

printf ( "%d ", i ) ;

}

int main()

{

changeValue();

changeValue();

changeValue();

changeValue();

return 0;

}
```

Output:-

1. 2 2 2 2

In the above example, every time the method changeValue is invoked, memory is allocated for i and de allocated at the end of the method. So it's output will be same.

- **register**

It's similar to auto variables. Difference is that register variables might be stored on the processor register instead of RAM, that means the maximum size of register variable should be the size of CPU register (like 16bit, 32bit or 64bit). This is normally used for frequently accessed variables like counters, to improve performance. But note that, declaring a variable as register does not mean that they will be stored in the register. It depends on the hardware and implementation.

```
int main()
```

```
{

register int i;

int array[10] = {0,1,2,3,4,5,6,7,8,9};

for (i=0;i<10;i++)

{

printf("%d ", array[i]);

}

return 0;

}
```

Output:-

0 1 2 3 4 5 6 7 8 9

The variable i might be stored on the CPU register and due to which the access of i in the loop will be faster.

- **static**

  A static variable will be kept in existence till the end of the program unlike creating and destroying each time they move into and out of the scope. This helps to maintain their value even if control goes out of the scope. When static is used with global variables, they will have internal linkage, that means it cannot be accessed by other source files. When static is used in case of a class member, it will be shared by all the objects of a class instead of creating separate copies for each object.

```
void changeValue(void)

{
```

```
static int i = 1 ;

i++;

printf ( "%d ", i ) ;

}

int main(){

changeValue();

changeValue();

changeValue();

changeValue();

return 0;

}
```

Output:-

1. 2 3 4 5

Since static variable will be kept in existence till the end of program, variable i will retain it's value across the method invocations.

- **extern**

extern is used to tell compiler that the symbol is defined in another translation unit (or in a way, source files) and not in the current one. Which means the symbol is linked externally. extern symbols have static storage duration, that is accessible through out the life of program. Since no storage is allocated for extern variable as part of declaration, they cannot be initialized while declaring.

```
int x = 10;
```

```
int main( )

{

extern int y ;

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

printf("y: %d", y);

return 0;

}

int y = 70 ;
```

Output:-

x: 10 y: 70

extern variable is like global variable, it's scope is through out the program. It can be defined anywhere in the c++ program.

- **mutable**

mutable storage class can be used only on non static non const data a member of a class. Mutable data member of a class can be modified even is it's part of an object which is declared as const.

```
class Test

{

public:

Test(): x(1), y(1) {};
```

```
mutable int x;

int y;

};

int main()

{

const Test object;

x = 123;

//object.y = 123;

/*

* The above line if uncommented, will create compilation error.

*/

cout<< "X:"<< object.x << ", Y:" << object.y;

return 0;

}
```

Output:-

```
X:123, Y:1
```

In the above example, we are able to change the value of member variable x though it's part of an object which is declared as const. This is because the variable x is declared as mutable. But if you try to modify the value of member variable y, compiler will throw error. You can find the summary of c++ storage class specifiers below

C++ Storage Specifier	Storage Location	Scope Of Variable	Life Time
auto	Memory (RAM)	Local	With in function
static	Memory (RAM)	Local	Life time is from when the flow reaches the first declaration to the termination of program.
register	CPU register	Local	With in function
extern	Memory (RAM)	Global	Till the end of main program

Q14. What is 'Copy Constructor' and when it is called?

Ans: This is a frequent c++ interview question. Copy constructor is a special constructor of a class which is used to create copy of an object. Compiler will give a default copy constructor if you don't define one. This implicit constructor will copy all the members of source object to target object.
Implicit copy constructors are not recommended, because if the source object contains pointers they will be copied to target object, and it may cause heap corruption when both the objects with pointers referring to the same location does an update to the memory location. In this case its better to define a custom copy constructor and do a deep copy of the object.

```
class SampleClass{

public:

int* ptr;

SampleClass();

// Copy constructor declaration

SampleClass(SampleClass &obj);

};
```

```
SampleClass::SampleClass(){

ptr = new int();

*ptr = 5;

}

// Copy constructor definition

SampleClass::SampleClass(SampleClass &obj){

//create a new object for the pointer

ptr = new int();

// Now manually assign the value

*ptr = *(obj.ptr);

cout<<"Copy constructor...\n";

}
```

Q15. What is realloc() and free()? What is difference between them?

Ans:

- void* realloc (void* ptr, size_t size)

  This function is used to change the size of memory object pointed by address **ptr** to the size given by **size**. If ptr is a null pointer, then realloc will behave like malloc(). If the ptr is an invalid pointer, then defined behaviour may occur depending the implementation. Undefined behaviour may occur if the ptr has previously been deallocated by free(), or dealloc() or ptr do not match a pointer returned by an malloc(), calloc() or realloc().

- void free (void* ptr)

This function is used to deallocate a block of memory that was allocated using malloc(), calloc() or realloc(). If ptr is null, this function does not doe anything.

Q16. What is difference between shallow copy and deep copy? Which is default?

Ans:

[This question can be expected in any interviews, not just c++ interviews. This is a usual question in most of the java interviews.]

When you do a shallow copy, all the fields of the source object is copied to target object as it is. That means, if there is a dynamically created field in the source object, shallow copy will copy the same pointer to target object. So you will have two objects with fields that are pointing to same memory location which is not what you usually want. In case of deep copy, instead of copying the pointer, the object itself is copied to target. In this case if you modify the target object, it will not affect the source. By default copy constructors and assignment operators do shallow copy. To make it as deep copy, you need to create a custom copy constructor and override assignment operator.

Q17. What do you mean by persistent and non persistent objects?

Ans: Persistent objects are the ones which we can be serialized and written to disk, or any other stream. So before stopping your application, you can serialize the object and on restart you can deserialize it. [Drawing applications usually use serializations.] Objects that can not be serialized are called non persistent objects. [Usually database objects are not serialized because connection and session will not be existing when you restart the application.]

Q18. Is it possible to get the source code back from binary file?

Ans: Technically it is possible to generate the source code from binary. It is called reverse engineering. There are lot of reverse engineering tools available. But, in actual case most of them will not re generate the exact source code back because many information will be lost due to compiler optimization and other interpretations.

Q19. What are virtual functions and what is its use?

Ans: Virtual functions are member functions of class which is declared using keyword 'virtual'. When a base class type reference is initialized using object of sub class type and an overridden method which is declared as virtual is invoked using the base reference, the method in child class object will get invoked.

```
class Base

{

int a;

public:

Base()

{

a = 1;

}

virtual void method()

{

cout << a;

}

};

class Child: public Base

{

int b;

public:

Child()
```

```
{

b = 2;

}

virtual void method()

{

cout << b;}

};

int main()

{

Base *pBase;

Child oChild;

pBase = &oChild;

pBase->method();

return 0;

}
```

In the above example even though the method in invoked on Base class reference, method of the child will get invoked since its declared as virtual.

Q20. What do you mean by pure virtual functions in C++? Give an example?

Ans: Pure virtual function is a function which doesn't have an implementation and the same needs to be implemented by the the next immediate non-abstract class. (A class

will become an abstract class if there is at-least a single pure virtual function and thus pure virtual functions are used to create interfaces in c++).

Q21. How to create a pure virtual function?

Ans: A function is made as pure virtual function by the using a specific signature, ***" = 0"*** appended to the function declaration as given below,

```
class SymmetricShape

{

public:

// draw() is a pure virtual function.

virtual void draw() = 0;

};
```

Q22. Why pure virtual functions are used if they don't have implementation / When does a pure virtual function become useful?

Ans: Pure virtual functions are used when it doesn't make sense to provide definition of a virtual function in the base class or a proper definition does not exists in the context of base class. Consider the above example, class ***SymmetricShape*** is used as base class for shapes with symmetric structure(Circle, square, equilateral triangle etc). In this case, there exists no proper definition for function ***draw()*** in the base class ***SymmetricShape*** instead the child classes of ***SymmetricShape (Cirlce, Square*** etc) can implement this method and draw proper shape.

Q23. What is virtual destructors? Why they are used?

Ans: *[This c++ interview question is in a way related to polymorphism.]*

Virtual destructors are used for the same purpose as virtual functions. When you remove an object of subclass, which is referenced by a parent class pointer, only destructor of base class will get executed. But if the destructor is defined using virtual keyword, both the destructors [of parent and sub class] will get invoked.

Q24. What you mean by early binding and late binding? How it is related to dynamic binding?

Ans: *[This c++ interview question is related to question about virtual functions]*

Binding is the process of linking actual address of functions or identifiers to their reference. This happens mainly two times.

- During compilation : This is called early binding

  For all the direct function references compiler will replace the reference with actual address of the method.

- At runtime : This is called late binding.

  In case of virtual function calls using a Base reference, as in shown in the example of question no: 2, compiler does not know which method will get called at run time. In this case compiler will replace the reference with code to get the address of function at runtime.
  Dynamic binding is another name for late binding.

Q25. What is meant by reference variable in C++?

Ans: In C++, reference variable allows you create an alias (second name) for an already existing variable. A reference variable can be used to access (read/write) the original data. That means, both the variable and reference variable are attached to same memory location. In effect, if you change the value of a variable using reference variable, both will get changed (because both are attached to same memory location).

Q26. How to create a reference variable in C++

Ans: Appending an ampersand (&) to the end of datatype makes a variable eligible to use as reference variable.

```
1. int a = 20;
2. int& b = a;
```

The first statement initializes a an integer variable a. Second statement creates an **integer reference initialized to variable *a***
Take a look at the below example to see how reference variables work.

```
1.  int main ()
2.  {
3.  int   a;
4.  int&  b = a;
5.
6.  a = 10;
7.  cout << "Value of a : " << a << endl;
8.  cout << "Value of a reference (b) : " << b  << endl;
9.
10. b = 20;
11. cout << "Value of a : " << a << endl;
12. cout << "Value of a reference (b) : " << b  << endl;
13.
14. return 0;
15. }
```

Above code creates following output.

```
Value of a : 10
Value of a reference (b) : 10
Value of a : 20
Value of a reference (b) : 20
```

Q27. What are the difference between reference variables and pointers in C++?

Ans: [This question is usually asked in a twisted way during c++ interviews. Sometimes the interviewer might use examples and ask you to find the error.]

Pointers	**Reference Variables**
Pointers can be assigned to NULL	References cannot be assigned NULL. It should always be associated with actual memory, not NULL.
Pointers can be (re)pointed to any object, at any time, any number of times during the execution.	Reference variables should be initialized with an object when they are created and they cannot be reinitialized to refer to another object
Pointer has own memory address and location on stack	Reference variables has location on stack, but shares the same memory location with the object it refer to.

Q28. What will the line of code below print out and why?

Ans: cout << 25u - 50;

The answer is *not* -25. Rather, the answer (which will surprise many) is 4294967271, assuming 32 bit integers. Why?

In C++, if the types of two operands differ from one another, then the operand with the "lower type" will be promoted to the type of the "higher type" operand, using the following type hierarchy (listed here from highest type to lowest type): long double, double, float, unsigned long int, long int, unsigned int, int (lowest).

So when the two operands are, as in our example, 25u (unsigned int) and 50 (int), the 50 is promoted to also being an unsigned integer (i.e., 50u).

Moreover, the result of the operation will be of the type of the operands. Therefore, the result of 25u - 50u will itself be an unsigned integer as well. So the result of -25 converts to 4294967271 when promoted to being an unsigned integer.

C++ supports multiple inheritance. What is the "diamond problem" that can occur with multiple inheritance? Give an example.

Let's consider a simple example. A university has people who are affiliated with it. Some are students, some are faculty members, some are administrators, and so on. So a simple inheritance scheme might have different types of people in different roles, all of whom inherit from one common "Person" class. The Person class could define an abstract getRole() method which would then be overridden by its subclasses to return the correct role type.

But now what happens if we want to model a the role of a Teaching Assistant (TA)? Typically, a TA is *both* a grad student *and* a faculty member. This yields the classic diamond problem of multiple inheritance and the resulting ambiguity regarding the TA's getRole() method:

Which getRole() implementation should the TA inherit? That of the Faculty Member or that of the Grad Student? The simple answer might be to have the TA class override the getRole() method and return newly-defined role called "TA". But that answer is also imperfect as it would hide the fact that a TA is, in fact, both a faculty member and a grad student.

Q29. What is the error in the code below and how should it be corrected?

Ans: my_struct_t *bar;

/* ... do stuff, including setting bar to point to a defined my_struct_t object ... */

memset(bar, 0, sizeof(bar));

The last argument to memset should be sizeof(*bar), not sizeof(bar). sizeof(bar) calculates the size of bar (i.e., the pointer *itself*) rather than the size of the structure pointed to by bar.

The code can therefore be corrected by using sizeof(*bar) as the last argument in the call to memset.

A sharp candidate might point out that using *bar will cause a dereferencing error if bar has not been assigned. Therefore an even safer solution would be to use sizeof(my_struct_t). However, an even sharper candidate must know that in this case using *bar is absolutely safe within the call to sizeof, even if bar has not been initialized

yet, since sizeof is a compile time construct.

Q30. What will i and j equal after the code below is executed? Explain your answer.

Ans: int i = 5;

int j = i++;

After the above code executes, i will equal 6, but j will equal 5.

Understanding the reason for this is fundamental to understanding how the unary increment (++) and decrement (--) operators work in C++.

When these operators *precede* a variable, the value of the variable is modified first and *then* the modified value is used. For example, if we modified the above code snippet to instead say int j = ++i;, i would be incremented to 6 and *then* j would be set to that modified value, so both would end up being equal to 6.

However, when these operators *follow* a variable, the unmodified value of the variable is used and *then* it is incremented or decremented. That's why, in the statement int j = i++; in

the above code snippet, j is first set to the unmodified value of i (i.e., 5) and *then* i is incremented to 6.

Q31. What is the problem in the code below? What would be an alternate way of implementing this that would avoid the problem?

Ans: size_t sz = buf->size();

while (--sz >= 0)

{

/* do something */

}

The problem in the above code is that --sz >= 0 will *always* be true so you'll never exit the while loop (so you'll probably end up corrupting memory or causing some sort of memory violation or having some other program failure, depending on what you're doing inside the loop).

The reasons that --sz >= 0 will *always* be true is that the type of sz is size_t. size_t is really just an alias to one of the fundamental unsigned integer types. Therefore, since sz is unsigned, it can *never* be less than zero (so the condition can never be true).

One example of an alternative implementation that would avoid this problem would be to instead use a for loop as follows:

for (size_t i = 0; i < sz; i++)

{

/* do something */

}

Q32. Consider the two code snippets below for printing a vector. Is there any advantage of one vs. the other? Explain.

Option 1:

```
vector vec;

/* ... .. ... */

for (auto itr = vec.begin(); itr != vec.end(); itr++) {

itr->print();

}
```

Option 2:

```
vector vec;

/* ... .. ... */

for (auto itr = vec.begin(); itr != vec.end(); ++itr) {

itr->print();

}
```

Ans: Although both options will accomplish precisely the same thing, the second option is better from a performance standpoint. This is because the post-increment operator (i.e., itr++) is more expensive than pre-increment operator (i.e., ++itr). The underlying implementation of the post-increment operator makes a copy of the element before incrementing it and then returns the copy.

Q33. Implement a template function IsDerivedFrom() that takes class C and class P as template parameters. It should return true when class C is derived from class P and false otherwise.

Ans: This question tests understanding of C++ templates. An experienced developer will know that this is already a part of the C++11 std library (std::is_base_of) or part of the boost library for C++ (boost::is_base_of). Even an interviewee with only passing knowledge should write something similar to this, mostly likely involving a helper class:

```
template<typename D, typename B>

class IsDerivedFromHelper

{

class No { };

class Yes { No no[3]; };



static Yes Test( B* );

static No Test( ... );

public:

enum { Is = sizeof(Test(static_cast<D*>(0))) == sizeof(Yes) };

};

template <class C, class P>

bool IsDerivedFrom() {

return IsDerivedFromHelper<C, P>::Is;

}
```

Q34. Implement a template boolean IsSameClass() that takes class A and B as template parameters. It should compare class A and B and return false when they are different classes and true if they are the same class.

Ans:

```
template <typename T, typename U>

struct is_same
```

```
{

static const bool value = false;

};

template <typename T>

struct is_same<T, T>

{

static const bool value = true;

};

template <class A, class B>

bool IsSameClass() {

return is_same<A, B>::value;

}
```

Q35. Is it possible to have a recursive inline function?

Ans: Although you can call an inline function from within itself, the compiler will not generate inline code since the compiler cannot determine the depth of recursion at compile time.

Q36. What is the output of the following code:

Ans:

```
#include <iostream>

class A {

public:
```

```
A() {}

~A() {

throw 42;

}

};

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

try {

A a;

throw 32;

} catch(int a) {

std::cout << a;

}

}
```

This program will terminate abnormally. throw 32 will start unwinding the stack and destroy class A. The class A destructor will throw another exception during the exception handling, which will cause program to crash. This question is testing if developer has experience working with exceptions.

Q37. You are given library class Something as follows:

```
class Something {

public:
```

Something() {

topSecretValue = 42;

}

public:

bool somePublicBool;

int somePublicInt;

std::string somePublicString;

private:

int topSecretValue;

};

Implement a method to get topSecretValue for any given Something* object. The method should be cross-platform compatible and not depend on sizeof (int, bool, string).

Ans: Create another class which has all the members of Something in the same order, but has additional public method which returns the value. Your replica Something class should look like:

class SomethingReplica {

public:

int getTopSecretValue() { return topSecretValue; }

public:

bool somePublicBool;

```
int somePublicInt;

std::string somePublicString;

private:

int topSecretValue;

};
```

Then, to get the value:

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

Something a;

SomethingReplica* b = reinterpret_cast<SomethingReplica*>(&a);

std::cout << b->getTopSecretValue();

}
```

It's important to avoid code like this in a final product, but it's nevertheless a good technique when dealing with legacy code, as it can be used to extract intermediate calculation values from a library class. (Note: If it turns out that the alignment of the external library is mismatched to your code, you can resolve this using #pragma pack.)

Q38. Implement a void function F that takes pointers to two arrays of integers (A and B) and a size N as parameters. It then populates B where B[i] is the product of all A[j] where j != i.

For example: if A = {2, 1, 5, 9}, then B would be {45, 90, 18, 10}

Ans: This problem seems easy at first glance so a careless developer might write something like this:

```
void F(int* A, int* B, int N) {
```

```
int m = 1;

for (int i = 0; i < N; ++i) {

m *= A[i];

}

for (int i = 0; i < N; ++i) {

B[i] = m / A[i];

}

}
```

This will work for the given example, but when you add a 0 into input array A, the program will crash because of division by zero. The correct answer should take that edge case into account and look like this:

```
void F(int* A, int* B, int N) {

int m = 1;

int numZero = 0;

int zeroIndex = -1;

for (int i = 0; i < N; ++i) {

B[i] = 0;

if (A[i] == 0) {

++numZero;

zeroIndex = i;
```

```
} else {

m *= A[i];

}

}

if (numZero == 0) {

for (int i = 0; i < N; ++i) {

B[i] = m / A[i];

}

return;

}

if (numZero >= 2) {

return;

}

B[zeroIndex] = m;

}
```

The presented solution above has a Big O complexity of O(n). While there are simpler solutions available (ones that would ignore the need to take 0 into account), that simplicity has a price of complexity, generally running significantly slower.

Q39. When you should use virtual inheritance?

Ans: While it's ideal to avoid virtual inheritance altogether (you should know how your class is going to be used) having a solid understanding of how virtual inheritance works is still important:

So when you have a class (class A) which inherits from 2 parents (B and C), both of which share a parent (class D), as demonstrated below:

```
#include <iostream>

class D {

public:

void foo() {

std::cout << "Foooooo" << std::endl;

}

};

class C:  public D {

};

class B:  public D {

};

class A: public B, public C {

};

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

A a;
```

```
a.foo();

}
```

If you don't use virtual inheritance in this case, you will get two copies of D in class A: one from B and one from C. To fix this you need to change the declarations of classes C and B to be virtual, as follows:

```
class C:  virtual public D {

};

class B:  virtual public D {

};
```

Q40. Is there a difference between class and struct?

Ans: The only difference between a class and struct are the access modifiers. Struct members are public by default; class members are private. It is good practice to use classes when you need an object that has methods and structs when you have a simple data object.

Q41. What is the output of the following code:

Ans:

```
#include <iostream>

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

int a[] = {1, 2, 3, 4, 5, 6};

std::cout << (1 + 3)[a] - a[0] + (a + 1)[2];

}
```

The above will output 8, since:

(1+3)[a] is the same as a[1+3] == 5

a[0] == 1

(a + 1)[2] is the same as a[3] == 4

This question is testing pointer arithmetic knowledge, and the magic behind square brackets with pointers.

While some might argue that this isn't a valuable question as it appears to only test the capability of reading C constructs, it's still important for a candidate to be able to work through it mentally; it's not an answer they're expected to know off the top of their head, but one where they talk about what conclusion they reach and how.

Q42. What is the output of the following code:

Ans: #include

class Base {

virtual void method() {std::cout << "from Base" << std::endl;}

public:

virtual ~Base() {method();}

void baseMethod() {method();}

};

class A : public Base

{

void method() {std::cout << "from A" << std::endl;}

public:

~A() {method();}

```
};

int main(void) {

Base* base = new A;

base->baseMethod();

delete base;

return 0;

}
```

The above will output:

```
from A

from A

from Base
```

The important thing to note here is the order of destruction of classes and how Base's method reverts back to its own implementation once A has been destroyed.

Q43. Explain the volatile and mutable keywords

Ans: The volatile keyword informs the compiler that a variable will be used by multiple threads. Variables that are declared as volatile will not be cached by the compiler to ensure the most up-to-date value is held.

The mutable keyword can be used for class member variables. Mutable variables are allowed to change from within const member functions of the class.

Q44. How many times will this loop execute? Explain your answer.

Ans: unsigned char half_limit = 150;

```
for (unsigned char i = 0; i < 2 * half_limit; ++i)

{

// do something;

}
```