ESKO Technical Interview Questions

Q1. What is conversion constructor?

ANS:

Constructor with a single argument makes that constructor as conversion ctor and it can be used for type conversion.
For example:

```
class Boo
{
public:
Boo( int i );
};
Boo BooObject = 10 ; // assigning int 10 Boo object
```

Q2. What is reference?

ANS:

1. Reference is a name that acts as an alias, or alternative name, for a previously defined variable or an object.
2. Prepending variable with & symbol makes it as reference.
3. For example:

```
int a;
int &b = a;
```

Q3. What is virtual function?

ANS:

When derived class overrides the base class method by redefining the same function, then if client wants to access redefined the method from derived class through a pointer from base class object, then you must define this function in base class as virtual function.

```
class parent
{
void Show()
```

```
{
cout << im parent << endl;
}
};
class child: public parent
{
void Show()
{
cout << im child << endl;
}
};
parent * parent_object_ptr = new child;
parent_object_ptr->show() // calls parent->show() i
now we goto virtual world...
class parent
{
virtual void Show()
{
cout << im parent << endl;
}
};
class child: public parent
{
void Show()
{
cout << im child << endl;
}
};
parent * parent_object_ptr = new child;
parent_object_ptr->show() // calls child->show()
```

Q4. What is overloading?

ANS:

With the C++ language, you can overload functions and operators. Overloading is the practice of supplying more than one definition for a given function name in the same scope.

- Any two functions in a set of overloaded functions must have different argument lists.
- Overloading functions with argument lists of the same types, based on return type alone, is an error.

Q5. What is - this - pointer?

ANS:

The this pointer is a pointer accessible only within the member functions of a class, struct, or union type. It points to the object for which the member function is called. Static member functions do not have a this pointer.

When a non static member function is called for an object, the address of the object is passed as a hidden argument to the function. For example, the following function call:

```
myDate.setMonth( 3 );
can be interpreted this way:
setMonth( &myDate, 3 );
```

The objects address is available from within the member function as the this pointer. It is legal, though unnecessary, to use the this pointer when referring to members of the class.

Q6. What are the access privileges in C++? What is the default access level?

ANS:

The access privileges in C++ are private, public and protected. The default access level assigned to members of a class is private. Private members of a class are accessible only within the class and by friends of the class. Protected members are accessible by the class itself and its sub-classes. Public members of a class can be accessed by anyone.

Q7. What will happen if any string in C is converted to integer explicitly?

ANS:

If we try to convert string into integer explicitly then strings address will get stored in integer.

```
int main(){
char *a=abhas;
int b=(int)a; // now b will hold address of a
}
```

Q8. What is constructor or ctor?

ANS:

Constructor creates an object and initializes it. It also creates v table for virtual functions. It is different from other methods in a class.

Q9. What is encapsulation?

ANS:

Containing and hiding information about an object, such as internal data structures and code. Encapsulation isolates the internal complexity of an objects operation from the rest of the application. For example, a client component asking for net revenue from a business object need not know the datas origin.

Q10. What is a local class? Why can it be useful?

ANS:

Local class is a class defined within the scope of a function -- any function, whether a member function or a free function. For example:

```
// Example 2: Local class
```

```
//
int f()
{
class LocalClass
{
// ...
};
// ...
};
```

Like nested classes, local classes can be a useful tool for managing code dependencies.

Q11. What is copy constructor?

ANS:

Constructor which initializes its object member variables (by shallow copying) with another object of the same class. If you do not implement one in your class then compiler implements one for you.
For example:

```
Boo Obj1(10); // calling Boo constructor
Boo Obj2(Obj1); // calling boo copy constructor
Boo Obj2 = Obj1;// calling boo copy constructor
```

Q12. What is inheritance?

ANS:

Inheritance allows one class to reuse the state and behavior of another class. The derived class inherits the properties and method implementations of the base class and extends it by overriding methods and adding additional properties and methods.

Q13. What is the difference between new and operator new?

ANS:

Operator new works like malloc.

Q14. What are storage qualifiers in C++?

ANS:

They are..
const
volatile
mutable

Const keyword indicates that memory once initialized, should not be altered by a program.

Volatile keyword indicates that the value in the memory location can be altered even though nothing in the program code modifies the contents. For example if you have a pointer to hardware location that contains the time, where hardware changes the value of this pointer variable and not the program. The intent of this keyword to improve the optimization ability of the compiler.

Mutable keyword indicates that particular member of a structure or class can be altered even if a particular structure variable, class, or class member function is constant.

```
struct data
{
char name[80];
mutable double salary;
}
const data MyStruct = { Satish Shetty, 1000 }; //initlized by complier
strcpy ( MyStruct.name, Shilpa Shetty); // compiler error
MyStruct.salary = 2000 ; // complier is happy allowed
```

Q15. When are temporary variables created by C++ compiler?

ANS:

Provided that function parameter is a const reference, compiler generates temporary variable in following 2 ways.

a) The actual argument is the correct type, but it is not L value

```
double Cube(const double & num)
{
num = num * num * num;
return num;
}
double temp = 2.0;
double value = cube(3.0 + temp); // argument is a expression and not a L value;
```

b) The actual argument is of the wrong type, but of a type that can be converted to the correct type

```
long temp = 3L;
double value = cube root ( temp); // long to double conversion
```

Q16. What problem does the namespace feature solve?

ANS:

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 librarys external declarations with a unique namespace that eliminates the potential for those collisions.
namespace [identifier] { namespace-body }

A namespace declaration identifies and assigns a name to a declarative region.
The identifier in a namespace declaration must be unique in the declarative region in which it is used. The identifier is the name of the namespace and is used to reference its members.

Q17. What is a dangling pointer?

ANS:

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.

Q18. How are prefix and postfix versions of operator++() differentiated?

ANS:

The postfix version of operator++() has a dummy parameter of type int. The prefix version does not have dummy parameter.

Q19. What is Virtual Destructor?

ANS:

Using virtual destructors, you can destroy objects without knowing their type - the correct destructor for the object is invoked using the virtual function mechanism. Note that destructors can also be declared as pure virtual functions for abstract classes.

If someone will derive from your class, and if someone will say new Derived, where Derived is derived from your class, and if someone will say delete p, where the actual objects type is Derived but the pointer ps type is your class.

Q20. What is Polymorphism?

ANS:

1. Polymorphism allows a client to treat different objects in the same way even if they were created from different classes and exhibit different behaviors.

2. You can use implementation inheritance to achieve polymorphism in languages such as C++ and Java.
3. Base class objects pointer can invoke methods in derived class objects.
4. You can also achieve polymorphism in C++ by function overloading and operator overloading.

Q21. What is conversion operator?

ANS:

Class can have a public method for specific data type conversions.
For example:

```
class Boo
{
double value;
public:
Boo(int i )
operator double()
{
return value;
}
};
Boo BooObject;
double i = BooObject; // assigning object to variable i of type double.
now conversion operator gets called to assign the value.
```

Q22. What are C++ storage classes?

ANS:

auto
register
static
extern

auto: the default. Variables are automatically created and initialized when they are defined and are destroyed at the end of the block containing their definition. They are not visible outside that block

register: a type of auto variable. a suggestion to the compiler to use a CPU register for performance

static: a variable that is known only in the function that contains its definition but is never destroyed and retains its value between calls to that function. It exists from the time the program begins execution

extern: a static variable whose definition and placement is determined when all object and library modules are combined (linked) to form the executable code file. It can be visible outside the file where it is defined.

Q23. When do use const reference arguments in function?

ANS:

a) Using const protects you against programming errors that inadvertently alter data.
b) Using const allows function to process both const and non-const actual arguments, while a function without const in the prototype can only accept non constant arguments.
c) Using a const reference allows the function to generate and use a temporary variable appropriately.

Q24. What is Memory alignment?

ANS:

The term alignment primarily means the tendency of an address pointer value to be a multiple of some power of two. So a pointer with two byte alignment has a zero in the least significant bit. And a pointer with four byte alignment has a zero in both the two least significant bits. And so on. More alignment means a longer sequence of zero bits in the lowest bits of a pointer.

Q25. How can I handle a destructor that fails?

ANS:

1. Write a message to a log-file. But do not throw an exception.
2. The C++ rule is that you must never throw an exception from a destructor that is being called during the stack unwinding process of another exception. For example, if someone says throw Foo(), the stack will be unwound so all the stack frames between the throw Foo() and the } catch (Foo e) { will get popped. This is called stack unwinding.
3. During stack unwinding, all the local objects in all those stack frames are destructed. If one of those destructors throws an exception (say it throws a Bar object), the C++ runtime system is in a no-win situation: should it ignore the Bar and end up in the } catch (Foo e) { where it was originally headed? Should it ignore the Foo and look for a } catch (Bar e) { handler? There is no good answer -- either choice loses information.
4. So the C++ language guarantees that it will call terminate() at this point, and terminate() kills the process. Bang you are dead.

Q26. What is destructor?

ANS:

Destructor usually deletes any extra resources allocated by the object.

Q27. What is difference between malloc()/free() and new/delete?

ANS:

malloc allocates memory for object in heap but does not invoke objects constructor to initialize the object.

new allocates memory and also invokes constructor to initialize the object.

malloc() and free() do not support object semantics
Does not construct and destruct objects

```
string * ptr = (string *)(malloc (sizeof(string)))
```

Are not safe
Does not calculate the size of the objects that it construct
Returns a pointer to void

```
int *p = (int *) (malloc(sizeof(int)));
int *p = new int;
```

Are not extensible
new and delete can be overloaded in a class
delete first calls the objects termination routine (i.e. its destructor) and then releases the space the object occupied on the heap memory. If an array of objects was created using new, then delete must be told that it is dealing with an array by preceding the name with an empty []:-

```
Int_t *my_ints = new Int_t[10];
...
delete []my_ints;
```

Q28. What is passing by reference?

ANS:

Method of passing arguments to a function which takes parameter of type reference.
For example:

```
void swap( int & x, int & y )
{
int temp = x;
```

```
x = y;
y = temp;
}
int a=2, b=3;
swap( a, b );
```

Basically, inside the function there wont be any copy of the arguments x and y instead they refer to original variables a and b. so no extra memory needed to pass arguments and it is more efficient.

Q29. What is pure virtual function? or what is abstract class?

ANS:

When you define only function prototype in a base class without implementation and do the complete implementation in derived class. This base class is called abstract class and client wont able to instantiate an object using this base class.
You can make a pure virtual function or abstract class this way..

```
class Boo
{
void foo() = 0;
}
Boo MyBoo; // compilation error
```

Q30. How virtual functions are implemented C++?

ANS:

Virtual functions are implemented using a table of function pointers, called the v table. There is one entry in the table per virtual function in the class. This table is created by the constructor of the class. When a derived class is constructed, its base class is constructed first which creates the v table. If the derived class overrides any of the base classes virtual functions, those entries in the v table are overwritten by the derived class constructor. This is why you should never call virtual functions from a constructor: because the v table entries for the object may not have been set up by the derived class

constructor yet, so you might end up calling base class implementations of those virtual functions

Q31. What is Over riding?

ANS:

To override a method, a subclass of the class that originally declared the method must declare a method with the same name, return type (or a subclass of that return type), and same parameter list.
The definition of the method overriding is:

1. Must have same method name.
2. Must have same data type.
3. Must have same argument list.

Overriding a method means that replacing a method functionality in child class. To imply overriding functionality we need parent and child classes. In the child class you define the same method signature as one defined in the parent class.

Q32. What is an Iterator class?

ANS:

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. Something like a pointer.

Q33. What is difference between template and macro?

ANS:

There is no way for the compiler to verify that the macro parameters are of compatible types. The macro is expanded without any special type checking. If macro parameter has a post incremented variable (like c++), the increment is performed two times.

Because macros are expanded by the preprocessor, compiler error messages will refer to the expanded macro, rather than the macro definition itself. Also, the macro will show up in expanded form during debugging.
For example:

Macro:

```
#define min(i, j) (i < j ? i : j)
```

template:

```
template
T min (T i, T j)
{
return i < j ? i : j;
}
```

Q34. What is assignment operator?

ANS:

Default assignment operator handles assigning one object to another of the same class. Member to member copy (shallow copy)

Q35. What is a container class? What are the types of container classes?

ANS:

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.

Q36. What is name mangling in C++?

ANS:

The process of encoding the parameter types with the function/method name into a unique name is called name mangling. The inverse process is called demangling.

For example

Foo::bar(int, long) const is mangled as `bar__C3Fooil.

For a constructor, the method name is left out. That is Foo::Foo(int, long) const is mangled as `__C3Fooil.

Q37. What is a nested class? Why can it be useful?

ANS:

A nested class is a class enclosed within the scope of another class. For example:

Example 1: Nested class

```
//class OuterClass
{
class NestedClass
{
// ...
```

```
};
// ...
};
```

Nested classes are useful for organizing code and controlling access and dependencies. Nested classes obey access rules just like other parts of a class do; so, in Example 1, if NestedClass is public then any code can name it as OuterClass::NestedClass. Often nested classes contain private implementation details, and are therefore made private; in Example 1, if NestedClass is private, then only OuterClasss members and friends can use NestedClass.
When you instantiate as outer class, it wont instantiate inside class.

Q38. Can a copy constructor accept an object of the same class as parameter, instead of reference of the object?

ANS:

No. It is specified in the definition of the copy constructor itself. It should generate an error if a programmer specifies a copy constructor with a first argument that is an object and not a reference.

Q39. When are copy constructors called?

ANS:

Copy constructors are called in following cases:
a) when a function returns an object of that class by value
b) when the object of that class is passed by value as an argument to a function
c) when you construct an object based on another object of the same class
d) When compiler generates a temporary object

Q40. What is the use of using declaration?

ANS:

A using declaration makes it possible to use a name from a namespace without the scope operator.

Q41. What do you mean by Stack unwinding?

ANS:

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

Q42. What is the difference between const char *myPointer and char *const myPointer?

ANS:

Const char *myPointer is a non constant pointer to constant data; while char *const myPointer is a constant pointer to non constant data.

Q43. How can I handle a constructor that fails?

ANS:

throw an exception. Constructors do not have a return type, so it is not possible to use return codes. The best way to signal constructor failure is therefore to throw an exception.

Q44. What is default constructor?

ANS:

Constructor with no arguments or all the arguments has default values.

Q45. What are all the implicit member functions of the class?

ANS:

1. default ctor

2. copy ctor
3. assignment operator
4. default destructor
5. address operator

Q46. What is inline function?

ANS:

The __inline keyword tells the compiler to substitute the code within the function definition for every instance of a function call. However, substitution occurs only at the compilers discretion. For example, the compiler does not inline a function if its address is taken or if it is too large to inline.

Q47. What is the difference between a pointer and a reference?

ANS:

A reference must always refer to some object and, therefore, must always be initialized; pointers do not have such restrictions. A pointer can be reassigned to point to different objects while a reference always refers to an object with which it was initialized.

Q48. What is multiple inheritance (virtual inheritance)? What are its advantages and disadvantages?

ANS:

Multiple Inheritance is the process whereby a child can be derived from more than one parent class. The advantage of multiple inheritance is that it allows a class to inherit the functionality of more than one base class thus allowing for modeling of complex relationships. The disadvantage of multiple inheritance is that it can lead to a lot of confusion (ambiguity) when two base classes implement a method with the same name.

Q49. How do you access the static member of a class?

Q50. What does *extern C int func(int *, Foo)* accomplish?

ANS:

It will turn off name mangling for func so that one can link to code compiled by a C compiler.