# Technical Papers Technical C Plus Plus Questions Part II

Examrace Placement Series prepares you for the toughest placement exams to top companies.

1. Name some pure object oriented languages. Answer: Smalltalk, Java, Eiffel, Sather.
2. Name the operators that cannot be overloaded. Answer: Sizeof. *. →
3. What is a node class? Answer: A node class is a class that, relies on the base class for services and implementation, provides a wider interface to te 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.
4. What is an orthogonal base class? Answer: 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.
5. What is a container class? What are the types of container classes? Answer: 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.
6. What is a protocol class? Answer: An abstract class is a protocol class if: It neither contains nor inherits from classes that contain member data, non-virtual functions, or private (or protected) members of any kind. It has a non-inline virtual destructor defined with an empty implementation, all member functions other than the destructor including inherited functions, are declared pure virtual functions and left undefined.
7. What is a mixin class? Answer: A class that provides some but not all of the implementation for a virtual base class is often called mixin. Derivation done just for the purpose of redefining the virtual functions in the base classes is often called mixin inheritance. Mixin classes typically don't share common bases.

8. What is a concrete class? Answer: A concrete class is used to define a useful object that can be instantiated as an automatic variable on the program stack. The implementation of a concrete class is defined. The concrete class is not intended to be a base class and no attempt to minimize dependency on other classes in the implementation or behavior of the class.

9. What is the handle class? Answer: A handle is a class that maintains a pointer to an object that is programmatically accessible through the public interface of the handle class. Explanation: In case of abstract classes, unless one manipulates the objects of these classes through pointers and references, the benefits of the virtual functions are lost. User code may become dependent on details of implementation classes because an abstract type cannot be allocated statistically or on the stack without its size being known. Using pointers or references implies that the burden of memory management falls on the user. Another limitation of abstract class object is of fixed size. Classes however are used to represent concepts that require varying amounts of storage to implement them. A popular technique for dealing with these issues is to separate what is used as a single object in two parts: a handle providing the user interface and a representation holding all or most of the object's state. The connection between the handle and the representation is typically a pointer in the handle. Often, handles have a bit more data than the simple representation pointer, but not much more. Hence the layout of the handle is typically stable, even when the representation changes and also that handles are small enough to move around relatively freely so that the user needn't use the pointers and the references.

10. What is an action class? Answer: The simplest and most obvious way to specify an action in C + + is to write a function. However, if the action has to be delayed, has to be transmitted ‘elsewhere’ before being performed, requires its own data, has to be combined with other actions, etc then it often becomes attractive to provide the action in the form of a class that can execute the desired action and provide other services as well. Manipulators used with iostreams is an obvious example. Explanation: A common form of action class is a simple class containing just one virtual function. Class Action { public: Virtual int do_it (int) = 0; virtual ~Action (); } Given this, we can write code say a member that can store actions for later execution without using pointers to functions, without knowing anything about the objects involved, and without even knowing the name of the operation it invokes. For example: Class write_file: Public Action { File& f; public: Int do_it (int) { return fwrite (). Suceed (); } }; class error_message: Public Action { response_box db (message. Cstr (), “Continue” “Cancel” “Retry” ); switch (db. Getresponse () ) { case 0: Return 0; case 1: Abort (); case 2: Current_operation. Redo (); return 1; } }; A user of the Action class will be completely isolated from any knowledge of derived classes such as write_file and error_message.

11. When can you tell that a memory leak will occur? Answer: A memory leak occurs when a program loses the ability to free a block of dynamically allocated memory.

12. What is a parameterized type? Answer: A template is a parameterized construct or type containing generic code that can use or manipulate any type. It is called parameterized because an actual type is a parameter of the code body. Polymorphism may be achieved through parameterized types. This type of polymorphism is called parameteric polymorphism. Parameteric polymorphism is the mechanism by which the same code is used on different types passed as parameters.

13. Differentiate between a deep copy and a shallow copy? Answer: Deep copy involves using the contents of one object to create another instance of the same class. In a deep copy, the two objects may contain ht same information but the target object will have its own buffers and resources. The destruction of either object will not affect the remaining object. The overloaded assignment operator would create a deep copy of objects. Shallow copy involves copying the contents of one object into another instance of the same class thus creating a mirror image. Owing to straight copying of references and pointers, the two objects will share the same externally contained contents of the other object to be unpredictable. Explanation: Using a copy constructor we simply copy the data values member by member. This method of copying is called shallow copy. If the object is a simple class, comprised of built in types and no pointers this would be acceptable. This function would use the values and the objects and its behavior would not be altered with a shallow copy, only the addresses of pointers that are members are copied and not the value the address is pointing to. The data values of the object would then be inadvertently altered by the function. When the function goes out of scope, the copy of the object with all its data is popped off the stack. If the object has any pointers a deep copy needs to be executed. With the deep copy of an object, memory is allocated for the object in free store and the elements pointed to are copied. A deep copy is used for objects that are returned from a function.

14. What is an opaque pointer? Answer: A pointer is said to be opaque if the definition of the type to which it points to is not included in the current translation unit. A translation unit is the result of merging an implementation file with all its headers and header files.

15. What is a smart pointer? Answer: A smart pointer is an object that acts, looks and feels like a normal pointer but offers more functionality. In C + +, smart pointers are implemented as template classes that encapsulate a pointer and override standard pointer operators. They have a number of advantages over regular pointers. They are guaranteed to be initialized as either null pointers or pointers to a heap object. Indirection through a null pointer is checked. No delete is ever necessary. Objects are automatically freed when the last pointer to them has gone away. One significant problem with these smart pointers is that unlike regular pointers, they don't respect inheritance. Smart pointers are unattractive for polymorphic code. Given below is an example for the implementation of smart pointers. Example: Template class smart_pointer { public: Smart_pointer ();//makes a null pointer smart_pointer (const X& x)//makes pointer to copy of x X& operator * (); const X& operator * () const; X * operator → () const; smart_pointer (const smart_pointer &); const smart_pointer & operator = (const smart_pointer&); ~smart_pointer (); private://... };

This class implement a smart pointer to an object of type X. The object itself is located on the heap. Here is how to use it: Smart_pointer p = employee ( “Harris” 1333); Like other overloaded operators, p will behave like a regular pointer, cout<< * p; p → raise_salary (0.5)

16. What is reflexive association? Answer: The ‘is-a’ is called a reflexive association because the reflexive association permits classes to bear the is-a association not only with their super-classes but also with themselves. It differs from a ‘specializes-from’ as ‘specializes-from’ is usually used to describe the association between a super-class and a sub-class. For example: Printer is-a printer.

17. What is slicing? Answer: Slicing means that the data added by a subclass are discarded when an object of the subclass is passed or returned by value or from a function expecting a base class object. Explanation: Consider the following class declaration: Class base { ... Base& operator = (const base&); base (const base&); } void fun () { base e = m; e = m; } As base copy functions don't know anything about the derived only the base part of the derived is copied. This is commonly referred to as slicing. One reason to pass objects of classes in a hierarchy is to avoid slicing. Other reasons are to preserve polymorphic behavior and to gain efficiency.

18. What is name mangling? Answer: Name mangling is the process through which your c + + compilers give each function in your program a unique name. In C + +, all programs have at-least a few functions with the same name. Name mangling is a concession to the fact that linker always insists on all function names being unique. Example: In general, member names are made unique by concatenating the name of the member with that of the class e. g. Given the declaration: Class Bar { public: Int ival; ... }; ival becomes something like://a possible member name mangling ival__3Bar Consider this derivation: Class Foo: Public Bar { public: Int ival; ... } The internal representation of a Foo object is the concatenation of its base and derived class members.//Pseudo C + + code//Internal representation of Foo class Foo { public: Int ival__3Bar; int ival__3Foo; ... }; Unambiguous access of either ival members is achieved through name mangling. Member functions, because they can be overloaded, require an extensive mangling to provide each with a unique name. Here the compiler generates the same name for the two overloaded instances (Their argument lists make their instances unique).

19. What are proxy objects? Answer: Objects that points to other objects are called proxy objects or surrogates. Its an object that provides the same interface as its server object but does not have any functionality. During a method invocation, it routes data to the true server object and sends back the return value to the object.

20. Differentiate between declaration and definition in C + +. Answer: A declaration introduces a name into the program; a definition provides a unique description of an entity (e. g. Type, instance, and function). Declarations can be repeated in a given scope, it introduces a name in a given scope. There must be exactly one definition of every object,

function or class used in a C + + program. A declaration is a definition unless: It declares a function without specifying its body, it contains an extern specifier and no initializer or function body, it is the declaration of a static class data member without a class definition, it is a class name definition, it is a typedef declaration. A definition is a declaration unless: It defines a static class data member, it defines a non-inline member function.

21. What is cloning? Answer: An object can carry out copying in two ways i.e., it can set itself to be a copy of another object, or it can return a copy of itself. The latter process is called cloning.

22. Describe the main characteristics of static functions. Answer: The main characteristics of static functions include, It is without the a this pointer, It can't directly access the non-static members of its class It can't be declared const, volatile or virtual. It doesn't need to be invoked through an object of its class, although for convenience, it may.

23. Will the inline function be compiled as the inline function always? Justify. Answer: An inline function is a request and not a command. Hence it won't be compiled as an inline function always. Explanation: Inline-expansion could fail if the inline function contains loops, the address of an inline function is used, or an inline function is called in a complex expression. The rules for inlining are compiler dependent.

24. Define a way other than using the keyword inline to make a function inline. Answer: The function must be defined inside the class.

25. How can a ‘’ operator be used as unary operator? Answer: The scope operator can be used to refer to members of the global namespace. Because the global namespace doesn't have a name, the notation: Member-name refers to a member of the global namespace. This can be useful for referring to members of global namespace whose names have been hidden by names declared in nested local scope. Unless we specify to the compiler in which namespace to search for a declaration, the compiler simple searches the current scope, and any scopes in which the current scope is nested, to find the declaration for the name.

26. What is placement new?

Answer:

When you want to call a constructor directly, you use the placement new. Sometimes you have some raw memory that's already been allocated, and you need to construct an object in the memory you have. Operator new's special version placement new allows you to do it.

class Widget { public: Widget (int widgetsize); ... Widget * Construct_widget_int_buffer (void * buffer, int widgetsize) { return new (buffer) Widget (widgetsize); } }.

This function returns a pointer to a Widget object that's constructed within the buffer passed to the function. Such a function might be useful for applications using shared memory or memory-

mapped I/O, because objects in such applications must be placed at specific addresses or in memory allocated by special routines.

OOAD

What do you mean by analysis and design?

Analysis:

Basically, it is the process of determining what needs to be done before how it should be done. In order to accomplish this, the developer refers the existing systems and documents. So, simply it is an art of discovery.

Design:

It is the process of adopting/choosing the one among the many, which best accomplishes the users needs. So, simply, it is compromising mechanism.

What are the steps involved in designing?

Before getting into the design the designer should go through the SRS prepared by the System Analyst.

The main tasks of design are Architectural Design and Detailed Design.

In Architectural Design we find what are the main modules in the problem domain.

In Detailed Design we find what should be done within each module.

What are the main underlying concepts of object orientation?

Objects, messages, class, inheritance and polymorphism are the main concepts of object orientation.

What do u meant by “SBI” of an object?

SBI stands for State, Behavior and Identity. Since every object has the above three.

State:

It is just a value to the attribute of an object at a particular time.

Behaviour:

It describes the actions and their reactions of that object.

Identity:

An object has an identity that characterizes its own existence. The identity makes it possible to distinguish any object in an unambiguous way, and independently from its state.

Differentiate persistent & non-persistent objects?

Persistent refers to an object's ability to transcend time or space. A persistent object stores/saves its state in a permanent storage system with out losing the information represented by the object.

A non-persistent object is said to be transient or ephemeral. By default objects are considered as non-persistent.

What do you meant by active and passive objects?

Active objects are one which instigate an interaction which owns a thread and they are responsible for handling control to other objects. In simple words it can be referred as client.

Passive objects are one, which passively waits for the message to be processed. It waits for another object that requires its services. In simple words it can be referred as server.

Diagram:

client server

(Active) (Passive)

What is meant by software development method?

Software development method describes how to model and build software systems in a reliable and reproducible way. To put it simple, methods that are used to represent ones'thinking using graphical notations.

What are models and meta models?

Model:

It is a complete description of something (i.e., system).

Meta model:

It describes the model elements, syntax and semantics of the notation that allows their manipulation.

What do you meant by static and dynamic modeling?

Static modeling is used to specify structure of the objects that exist in the problem domain. These are expressed using class, object and USECASE diagrams.

But Dynamic modeling refers representing the object interactions during runtime. It is represented by sequence, activity, collaboration and statechart diagrams.

How to represent the interaction between the modeling elements?

Model element is just a notation to represent (Graphically) the entities that exist in the problem domain. e. g. For modeling element is class notation, object notation etc.

Relationships are used to represent the interaction between the modeling elements.

The following are the Relationships.

Association: Its'just a semantic connection two classes.

Eg.

Aggregation: Its'the relationship between two classes which are related in the fashion that master and slave. The master takes full rights than the slave. Since the slave works under the master. It is represented as line with diamond in the master area.

ex:

car contains wheels, etc.

car

Containment: This relationship is applied when the part contained with in the whole part, dies when the whole part dies.

It is represented as darked diamond at the whole part.

example:

class A {//some code }.

class B { A aa;//an object of class A;//some code for class B; }.

In the above example we see that an object of class A is instantiated with in the class B. So the object class A dies when the object class B dies. We can represnt it in diagram like this.

Generalization: This relationship used when we want represents a class, which captures the common states of objects of different classes. It is represented as arrow line pointed at the class, which has captured the common states.

Dependency: It is the relationship between dependent and independent classes. Any change in the independent class will affect the states of the dependent class.

DIAGRAM:

class A class B

Why generalization is very strong?

Even though Generalization satisfies Structural, Interface, Behaviour properties. It is mathematically very strong, as it is Antisymmetric and Transitive.

Antisymmetric: Employee is a person, but not all persons are employees. Mathematically all As ‘are B, but all Bs’ not A.

Transitive: A ⇒ B, B ⇒ c then A ⇒ c.

1. Salesman.

2. Employee.

3. Person.

Note: All the other relationships satisfy all the properties like Structural properties, Interface properties, Behaviour properties.

Differentiate Aggregation and containment?

Aggregation is the relationship between the whole and a part. We can add/subtract some properties in the part (slave) side. It won't affect the whole part.

Best example is Car, which contains the wheels and some extra parts. Even though the parts are not there we can call it as car.

But, in the case of containment the whole part is affected when the part within that got affected. The human body is an apt example for this relationship. When the whole body dies the parts (heart etc) are died.

Can link and Association applied interchangeably?

No, You cannot apply the link and Association interchangeably. Since link is used represent the relationship between the two objects.

But Association is used represent the relationship between the two classes.

link: Student: Abhilash course: MCA

Association: Student course

what is meant by “method-wars”

Before 1994 there were different methodologies like Rumbaugh, Booch, Jacobson, Meyer etc who followed their own notations to model the systems. The developers were in a dilemma to choose the method which best accomplishes their needs. This particular span was called as “method-wars”

Whether unified method and unified modeling language are same or different?

Unified method is convergence of the Rumbaugh and Booch.

Unified modeling lang. Is the fusion of Rumbaugh, Booch and Jacobson as well as Betrand Meyer (whose contribution is “sequence diagram” ). Its'the superset of all the methodologies.

Who were the three famous amigos and what was their contribution to the object community?

The Three amigos namely.

James Rumbaugh (OMT): A veteran in analysis who came up with an idea about the objects and their Relationships (in particular Associations).

Grady Booch: A veteran in design who came up with an idea about partitioning of systems into subsystems.

Ivar Jacobson (Objectory): The father of USECASES, who described about the user and system interaction.

Differentiate the class representation of Booch, Rumbaugh and UML?

If you look at the class representaiton of Rumbaugh and UML, It is some what similar and both are very easy to draw.

Representation: OMT UML.

Diagram:

Booch: In this method classes are represented as “Clouds” which are not very easy to draw as for as the developer's view is concern.

Diagram:

What is an USECASE? Why it is needed?

A Use Case is a description of a set of sequence of actions that a system performs that yields an observable result of value to a particular action.

In SSAD process ⇔ In OOAD USECASE. It is represented elliptically.

Representation:

Who is an Actor?

An Actor is someone or something that must interact with the system. In addition to that an Actor initiates the process (that is USECASE).

It is represented as a stickman like this.

Diagram:

What is guard condition?

Guard condition is one, which acts as a firewall. The access from a particular object can be made only when the particular condition is met.

For Example.

customer check customer number ATM.

Here the object on the customer accesses the ATM facility only when the guard condition is met.

Differentiate the following notations?

I: Obj1: Obj2

II: Obj1: Obj2

In the above representation I, obj1 sends message to obj2. But in the case of II the data is transferred from obj1 to obj2.

USECASE is an implementation independent notation. How will the designer give the implementation details of a particular USECASE to the programmer?

This can be accomplished by specifying the relationship called refinement which talks about the two different abstraction of the same thing.

Or example.

calculate pay calculate

class1 class2 class3

Suppose a class acts an Actor in the problem domain, how to represent it in the static model?

In this scenario you can use stereotype. Since stereotype is just a string that gives extra semantic to the particular entity/model element. It is given with in the << >>

class A

<< Actor>>

attributes

methods.

Why does the function arguments are called as “signatures”

The arguments distinguish functions with the same name (functional polymorphism). The name alone does not necessarily identify a unique function. However, the name and its arguments (signatures) will uniquely identify a function.

In real life we see suppose, in class there are two guys with same name, but they can be easily identified by their signatures. The same concept is applied here.

ex:

class person { public: Char getsex (); void setsex (char); void setsex (int); }.

In the above example we see that there is a function setsex () with same name but with different signature.