ItsRunTym

C++

50+ interview questions/answers

1. **What is C++? Describe the advantages of C++?**

   C++ is a high-level programming language developed by Bjarne Stroustrup. It is an extension of the C programming language and supports object-oriented programming (OOP) features. By object-oriented we mean that it works with the concept of *polymorphism, inheritance, abstraction, encapsulation, object, and class*.

   C++ is widely used for developing system/software applications, games, drivers, and more.

   Advantages of C++:
   - C++ is an OOPs language that means the data is considered as objects.
   - C++ is a multi-paradigm language; In simple terms, it means that we can program the logic, structure, and procedure of the program.
   - Memory management is a key feature in C++ as it enables dynamic memory allocation
   - It is a Mid-Level programming language which means it can develop games, desktop applications, drivers, and kernels

   To read more, refer to the article – <u>What are the advantages of C++?</u>

2. **What is the difference between C and C++?**

C	C++
It is a procedural programming language. In	It is a mixture of both procedural and object-oriented

C	C++
simple words, it does not support classes and objects	programming languages. In simple words, it supports classes and objects.
It does not support any OOPs concepts like polymorphism, data abstraction, encapsulation, classes, and objects.	It supports all concepts of data
It does not support Function and Operator Overloading	It supports Function and Operator Overloading respectively
It is a function-driven language	It is an object-driven language

For more information, refer to [Difference between C and C++](#)

3. **Define 'std'?**

   '**std'** is also known as Standard or it can be interpreted as a namespace. The command "*using namespace std"* informs the compiler to add everything under the *std namespace* and inculcate them in the *global namespace*. This all inculcation of global namespace benefits us to use "**cout**" and "**cin"** without using "**std::_operator_".**

   For more information, refer to [namespace and std](#).

4. What is a class in C++?

   A class in C++ is a user-defined data type that serves as a blueprint for creating objects. It encapsulates data members (attributes) and member functions (methods) to operate on that data. Classes provide abstraction, encapsulation, inheritance, and polymorphism.

   // Example of a class in C++

```cpp
class Rectangle {
private:
int length;
int width;
public:
    // Constructor
    Rectangle(int l, int w) : length(l), width(w) {}


    // Member function to calculate area
    int area() {
    return length * width;
    }
};
```

5. What is an object in C++?
   An object in C++ is an instance of a class. It represents a real-world entity
   with attributes (data members) and behaviors (member functions). Objects
   are created using the class blueprint and can interact with each other to
   perform tasks.

   ```cpp
   // Creating objects of the Rectangle class
   Rectangle rect1(5, 4); // Object 1

   Rectangle rect2(3, 7); // Object 2


       // Calling member function on objects
       int area1 = rect1.area(); // Area of rect1
       int area2 = rect2.area(); // Area of rect2
   ```

6. Differentiate between a class and an object in C++.
   A class is a blueprint or template for creating objects, whereas an object is
   an instance of a class. A class defines the properties and behaviors common
   to all objects of that type, whereas an object represents a specific instance
   with its unique data.

7. **What are references in C++?**

When a variable is described as a reference it becomes an alias of the already existing variable. In simple terms, a referenced variable is another named variable of an existing variable keeping in mind that changes made in the reference variable will be reflected in the already existing variable. A reference variable is preceded with a **'&'** symbol.

Syntax:
int RUN = 10;

// reference variable
int& ref = RUN;
For more information, refer to references in C++

8. **What do you mean by Call by Value and Call by Reference?**
In this programming language to call a function we have 2 methods: **Call by Value** and **Call by Reference**

Call by Value	Call by Reference
A copy of a variable is passed.	A variable itself is passed fundamentally.
Calling a function by sending the values by copying variables.	Calling a function by sending the address of the passed variable.
The changes made in the function are never reflected outside the function on the variable. In short, the original value is never altered in Call by Value.	The changes made in the functions can be seen outside the function on the passed function. In short, the original value is altered in Call by reference.
Passed actual and formal parameters are stored in different memory locations. Therefore,	Passed actual and formal parameters are stored in the same memory location. Therefore, making Call by

Call by Value	Call by Reference
making Call by Value a little memory insufficient	Reference a little more memory efficient.

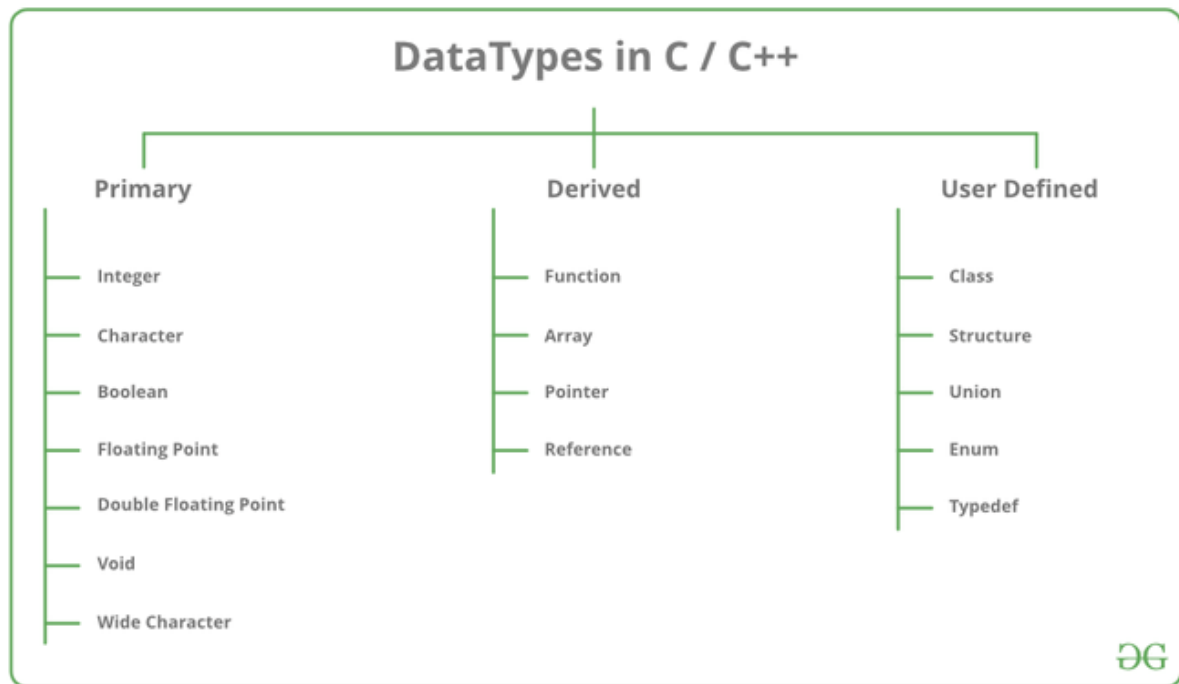For information, refer to [the difference between call by value and call by reference](#)

9. **Define token in C++**

   A token is the smallest individual element of a program that is understood by a compiler. A token comprises the following:

   - **Keywords** – That contain a special meaning to the compiler
   - **Identifiers** – That hold a unique value/identity
   - **Constants** – That never change their value throughout the program
   - **Strings** – That contains the homogenous sequence of data
   - **Special Symbols** – They have some special meaning and cannot be used for another purpose; eg: **[] () {}, ; * = #**
   - **Operators** – Who perform operations on the operand

   For more information, refer to [Tokens in C++](#)

10. **What are the different data types present in C++?**

DataTypes in C / C++

Primary
- Integer
- Character
- Boolean
- Floating Point
- Double Floating Point
- Void
- Wide Character

Derived
- Function
- Array
- Pointer
- Reference

User Defined
- Class
- Structure
- Union
- Enum
- Typedef

For more information, refer to C++ data types

11. What is the difference between struct and class?

Struct	Class
Members of the struct are always by default public mode	Members of the class can be in private, protected, and public modes.
Structures are of the value type. They only hold value in memory.	Classes are of reference type. It holds a reference of an object in memory.
The memory in structures is stored as stacks	The memory in classes is stored as heaps.

For more information, refer to the Difference between struct and class.

12. What is the difference between reference and pointer?

Reference	Pointer
The value of a reference cannot be reassigned	The value of a pointer can be reassigned
It can never hold a *null* value as it needs an existing value to become an alias of	It can hold or point at a *null* value and be termed as a *nullptr* or *null pointer*
It cannot work with arrays	It can work with arrays
To access the members of class/struct it uses a ' **.** '	To access the members of class/struct it uses a ' **->** '
The memory location of reference can be accessed easily or it can be used directly	The memory location of a pointer cannot be accessed easily as we have to use a dereference ' ***** '

For more information, refer to the [Difference between reference and pointer](#)

13. **What is the difference between function overloading and operator overloading?**

Function Overloading	Operator Overloading
It is basically defining a function in numerous ways such that there are many ways to call it or in simple terms you have multiple versions of the same function	It is basically giving practice of giving a special meaning to the existing meaning of an operator or in simple terms redefining the pre-redefined meaning

Function Overloading	Operator Overloading
Parameterized Functions are a good example of Function Overloading as just by changing the argument or parameter of a function you make it useful for different purposes	Polymorphism is a good example of an operator overloading as an object of allocations class can be used and called by different classes for different purposes

Example of Function Overloading:
 	
int RUN(int X, int Y);
 	
int RUN(char X, char Y);
	Example of Operator Overloading:
	
	int RUN() = X() + Y();
	
	int RUN() = X() − Y();

For more information, refer to Operator Overloading and Function Overloading

14. What is the difference between an array and a list?

Arrays	Lists
Array are contiguous memory locations of homogenous data types stored in a fixed location or size.	Lists are classic individual elements that are linked or connected to each other with the help of pointers and do not have a fixed size.
Arrays are static in nature. Uses less memory than linked lists.	Lists are dynamic in nature Uses more memory as it has to store the value and the pointer memory location

For more information, refer to Arrays Vs List

15.What is the difference between a while loop and a do-while loop?

While Loop	do-while Loop
While loop is also termed an entry-controlled loop	The do-while loop is termed an exit control loop
If the condition is not satisfied the statements inside the loop will not execute	Even if the condition is not satisfied the statements inside the loop will execute for at least one time
Example of a while loop: while(condition) {statements to be executed;};	**Example** of a do-while loop: do { statements to be executed; } while(condition or expression);

For more information, refer to the <u>Difference between while and do-while loop</u>

16.Discuss the difference between prefix and postfix?

prefix	postfix
It simply means putting the operator before the operand	It simply means putting the operator after the operand
It executes itself before **'; '**	It executes itself after **'; '**
Associativity of prefix ++ is right to left	Associativity of postfix ++ is left to right

For more information, refer to the <u>Difference between prefix and postfix</u>

17. What is the difference between new and malloc()?

new	malloc()
new is an operator which performs an operation	malloc is a function that returns and accepts values
new calls the constructors	malloc cannot call a constructor
new is faster than malloc as it is an operator	malloc is slower than new as it is a function
new returns the exact data type	malloc returns void*

For more information, refer to [Difference between new and malloc()](#)

18. What is the difference between virtual functions and pure virtual functions?

Virtual Function	Pure Virtual Function
A Virtual Function is a member function of a base class that can be redefined in another derived class.	A Pure Virtual Function is a member function of a base class that is only declared in a base class and defined in a derived class to prevent it from becoming an abstract class.
A virtual Function has its definition in its respective base class.	There is no definition in Pure Virtual Function and is initialized with a pure specifier **(= 0).**
The base class has a virtual function that can be	A base class having pure virtual function becomes abstract that cannot be

Virtual Function	Pure Virtual Function
represented or instanced; In simple words, its object can be made.	represented or instanced; In simple words, it means its object cannot be made.

For more information, refer to the Difference between virtual functions and pure virtual functions

19. What are classes and objects in C++?

A class is a user-defined data type where all the member functions and data members are tailor-made according to demands and requirements in addition to which these all can be accessed with the help of an **object**. To declare a user-defined data type we use a keyword **class.**

An object is an instance of a class and an entity with value and state; In simple terms, it is used as a catalyst or to represent a class member. It may contain different parameters or none.

Note: *A class is a blueprint that defines functions which are used by an object.*

For more information, refer to this What are classes and objects

20. What is Function Overriding?

When a function of the same name, same arguments or parameters, and same return type already present/declared in the base class is used in a derived class is known as Function Overriding. It is an example of Runtime Polymorphism or Late Binding which means the overridden function will be executed at the run time of the execution.

For more information, refer to Function Overriding in C++

21. What are the various OOPs concepts in C++?

- *Classes*: It is a user-defined datatype
- *Objects*: It is an instance of a class

- **Abstraction:** It is a technique of showing only necessary details
- **Encapsulation:** Wrapping of data in a single unit
- **Inheritance:** The capability of a class to derive properties and characteristics from another class
- **Polymorphism:** Polymorphism is known as many forms of the same thing

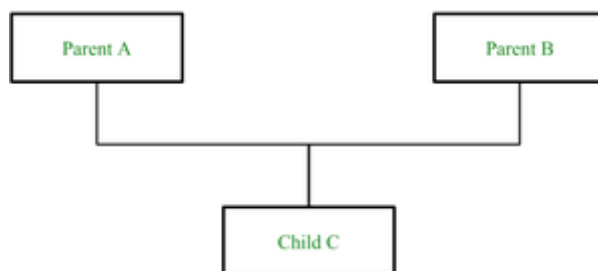For more information, refer to Various OOPs concepts in C++

22. Explain inheritance

The capability or ability of a class to derive properties and characteristics from another class is known as inheritance. In simple terms, it is a system or technique of reusing and extending existing classes without modifying them.

For more information, refer to Inheritance

23. When should we use multiple inheritance?

Multiple inheritances mean that a derived class can inherit two or more base/parent classes. It is useful when a derived class needs to combine numerous attributes/contracts and inherit some, or all, of the implementation from these attributes/contracts.

To take a real-life example consider your Parents where Parent A is your DAD Parent B is your MOM and Chid C is you.

Multiple Inheritances

For more information, refer to Multiple Inheritance.

24. What is virtual inheritance?

Virtual inheritance is a technique that ensures only one copy of a base class's member variables is inherited by grandchild-derived classes. Or in simple terms, virtual inheritance is used when we are dealing with a

situation of multiple inheritances but want to prevent multiple instances of the same class from appearing in the inheritance hierarchy.

25. What is polymorphism in C++?

Polymorphism is known as many forms of the same thing. In simple terms, we can say that Polymorphism is the ability to display a member function in multiple forms depending on the type of object that calls them.

In other words, we can also say that a man can be an employee to someone, a son of someone, a father of someone, and a husband of someone; this is how polymorphism can be projected in real life.

There is 2 type of polymorphism:

Compile Time Polymorphism

        a. Function Overloading

        b. Operator Overloading

Run Time Polymorphism

        c. Function Overriding

        d. Virtual Function

To know more about it, refer to [Polymorphism](#)

26. What are the different types of polymorphism in C++?

There is 2 type of polymorphism

1. Compile Time Polymorphism or Static Binding

This type of polymorphism is achieved during the compile time of the program which results in it making a bit faster than Run time. Also, Inheritance is not involved in it.

It is comprised of ***2 further techniques***:

- **Function Overloading:** When there are multiple functions with the same name but different parameters then this is known as function overloading.

C++

// same name different arguments
int RUN() {}
int RUN(int a) {}
float RUN(double a) {}
int RUN(int a, double b) {}

- **Operator Overloading:** It is basically giving practice of giving a special meaning to the existing meaning of an operator or in simple terms redefining the pre-redefined meaning
C++
class RUN {
// private and other modes
    statements **public** returnType
    operator symbol(arguments){ statements } statements
};

2. Run-Time Polymorphism or Late Binding
Run-time polymorphism takes place when functions are invoked during run time.

- **Function Overriding:** Function overriding occurs when a base class member function is redefined in a derived class with the same arguments and return type.
C++
// C++ program to demonstrate
// Function overriding
#include
<iostream> using namespace std;
class RUN {
    public:
        virtual void display()
        {
            cout << "Function of base class" << endl;
        }
};
class derived_RUN : **public** RUN {
    public:

```cpp
            void display()
            {
                    cout << "Function of derived class" << endl;
            }
    };

    int main()
    {
        derived_RUN rn;
        rn.display();
        return 0;
    }
```

Output:

Function of derived class

For more information, refer to [Different types of Polymorphism](#)

27. Compare compile-time polymorphism and Runtime polymorphism

Compile-Time Polymorphism	Runtime Polymorphism
28. It is also termed static binding and early binding.	It is also termed Dynamic binding and Late binding.
It is fast because execution is known early at compile time.	It is slow as compared to compile-time because execution is known at runtime.
It is achieved by function overloading and operator overloading.	It is achieved by virtual functions and function overriding.

For more information, refer to [Compile-time polymorphism and Runtime polymorphism](#)

29. Explain the constructor in C++.

A constructor is a special type of member function of a class, whose name is the same as that of the class by whom it is invoked and initializes value to the object of a class.

There are 3 types of constructors:

A. **Default constructor:** It is the most basic type of constructor which accepts no arguments or parameters. Even if it is not called the compiler calls it automatically when an object is created.

Example:

C++

class Class_name {

public:

   Class_name() { cout << "I am a default constructor"; }

};

B. **Parameterized constructor:** It is a type of constructor which accepts arguments or parameters. It has to be called explicitly by passing values in the arguments as these arguments help initialize an object when it is created. It also has the same name as that of the class.

Also, It is used to overload constructors.

Example:

C++

// CPP program to demonstrate

// parameterized constructors

#include

<iostream> using namespace std;

class RUN {

private:

   int x, y;

public:

   // Parameterized Constructor

   RUN(int x1, int y1)

   {

       x = x1;

     y = y1;

   }

int getX() { **return** x; }

```cpp
int getY() { return y; }
};

int main()
{
// Constructor called
RUN R(10, 15);
// Access values assigned by constructor
cout << "R.x = " << R.getX() << ", R.y = " << R.getY();
return 0;
}
```

Output
R.x = 10, R.y = 15

B. **Copy Constructor:** A copy constructor is a member function that initializes an object using another object of the same class. Also, the Copy constructor takes a reference to an object of the same class as an argument.

Example:
C++
```cpp
Sample(Sample& t) { id = t.id; }
```

For more information, refer to Constructors

30. What are destructors in C++?

Destructors are members of functions in a class that delete an object when an object of the class goes out of scope. Destructors have the same name as the class preceded by a tilde (~) sign. Also, destructors follow a **down-to-top** approach, unlike constructors which follow a top-to-down.

Syntax:
~constructor_name(); // tilde sign signifies that it is a destructor
For more information, refer to Destructor.

31. What is a virtual destructor?

When destroying instances or objects of a derived class using a base class pointer object, a virtual destructor is invoked to free up memory space allocated by the derived class object or instance.

Virtual destructor guarantees that first the derived class' destructor is called. Then the base class's destructor is called to release the space occupied by both destructors in the inheritance class which saves us from the memory leak. It is advised to make your destructor virtual whenever your class is polymorphic.
For more information, refer to [Virtual Destructor](#)

32. **Is destructor overloading possible? If yes then explain and if no then why?**
The simple answer is **NO** we cannot overload a destructor. It is mandatory to only destructor per class in C++. Also to mention, Destructor neither take arguments nor they have a parameter that might help to overload.

33. **Which operations are permitted on pointers?**
Pointers are the variables that are used to store the address location of another variable.
Operations that are permitted to a pointer are:
- Increment/Decrement of a Pointer
- Addition and Subtraction of integer to a pointer
- Comparison of pointers of the same type

34. **What is the purpose of the "*delete*" operator?**
The delete operator is used to delete/remove all the characteristics/properties from an object by deallocating its memory; furthermore, it returns true or false in the end. In simple terms, it destroys or deallocates array and non-array(pointer) objects which are created by new expressions.
C++
int RUN = **new** int[100];
// uses RUN for deletion
delete[] RUN;

For more information, refer to [Delete operator](#)

35. How delete [] is different from delete?

delete[]	delete
It is used for deleting a whole array	It is used to delete only one single pointer
It is used for deleting the objects of **new[];** By this, we can say that **delete[]** is used to delete an array of objects	It is used for deleting the objects of **new;** By this, we can say that **delete** is used to delete a single object
It can call as many destructors it wants	It can only call the destructor of a class once

36. What do you know about friend class and friend function?

A friend class is a class that can access both the protected and private variables of the classes where it is declared as a friend.

Example of friend class:
C++
```
class Class_1st {
    // ClassB is a friend class of ClassA
    friend class Class_2nd;
    statements;
}
class Class_2nd {
    statements;
}
```

A friend function is a function used to access the private, protected, and public data members or member functions of other classes.
It is declared with a friend keyword.
The advantage of a friend function is that it is not bound to the scope of the class and once it is declared in a class, furthermore to that, it cannot be

called by an object of the class; therefore it can be called by other functions. Considering all the mentioned points we can say that a friend function is a global function**.**

Example of friend function:
C++
class RUN {
   statements;
   friend dataype function_Name(arguments);
   statements;
} OR **class RUN** {
   Statements;
   friend int divide(10, 5);

   statements;

}

For more information, refer to the [friend function and friend class](#)

37. What is an Overflow Error?

Overflow Error occurs when the number is too large for the data type to handle. In simple terms, it is a type of error that is valid for the defined but exceeds used the defined range where it should coincide/lie.

For example, the range of int data type is **–**
2,147,483,648 to **2,147,483,647** and if we declare a variable of size **2,247,483,648** it will generate a overflow error.

38. What does the Scope Resolution operator do?

A scope resolution operator is denoted by a '**::**' symbol. Just like its name this operator resolves the barrier of scope in a program. A scope resolution operator is used to reference a member function or a global variable out of their scope furthermore to which it can also access the concealed variable or function in a program.
Scope Resolution is used for numerous amounts of tasks:
- To access a global variable when there is a local variable with the same name
- To define the function outside the class

- In case of multiple inheritances
- For namespace

For more information, refer to Scope resolution operator

39. What are the C++ access modifiers?

The access restriction specified to the class members(whether it is member function or data member) is known as access modifiers/specifiers.

Access Modifiers are of 3 types:

Private – It can neither be accessed nor be viewed from outside the class

Protected – It can be accessed if and only if the accessor is the derived class

Public – It can be accessed or be viewed from outside the class

For more information, refer to Access Modifiers

40. Can you compile a program without the main function?

Yes, it is absolutely possible to compile a program without a main().
For example Use Macros that defines the main

```cpp
C++
// C++ program to demonstrate the
// a program without main()
#include
<stdio.h>
#define fun main
int fun(void)
{
    printf("ItsRunTym");
    return 0;
}
```

For more information, refer to Can you compile a program without the main function

41. What is STL?

STL is known as Standard Template Library, it is a library that provides 4 components like container, algorithms, and iterators.
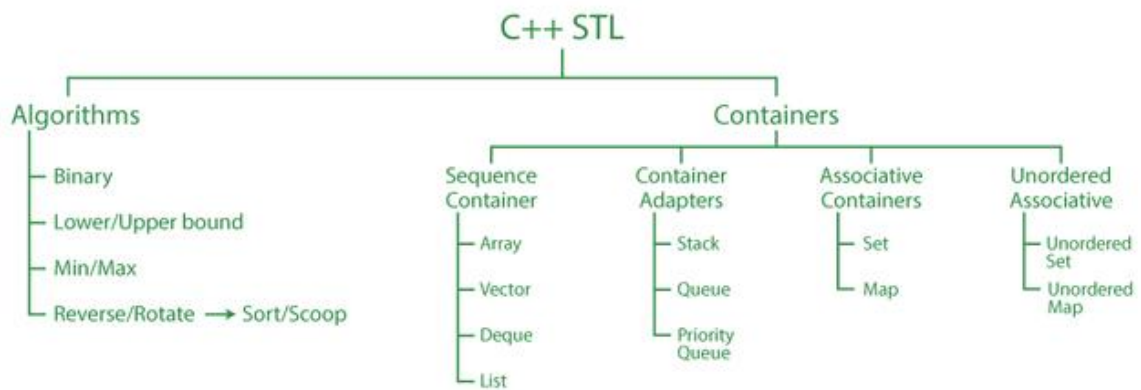
C++ STL

For more information, refer to STL in C++

42. Define inline function. Can we have a recursive inline function in C++?

An inline function is a form of request not an order to a compiler which results in the inlining of our function to the main function body.

An inline function can become overhead if the execution time of the function is less than the switching time from the caller function to called function.

To make a function inline use the keyword **inline** before and define the function before any calls are made to the function.
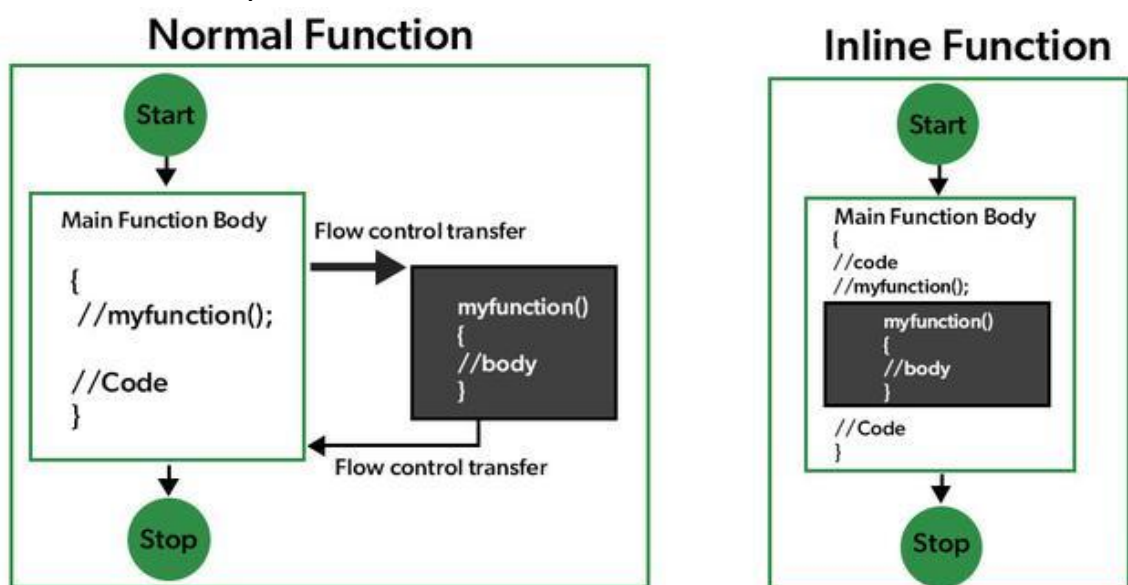
Inline Function Explanation

Syntax:

```
inline data_type function_name()
{
Body;
}
```

The answer is **No;** It cannot be recursive.

An inline function cannot be recursive because in the case of an inline function the code is merely placed into the position from where it is called and does not maintain a piece of information on the stack which is necessary for recursion.

Plus, if you write an inline keyword in front of a recursive function, the compiler will automatically ignore it because the inline is only taken as a suggestion by the compiler.

For more information, refer to Inline Function

43. What is an abstract class and when do you use it?

An abstract class is a class that is specifically designed to be used as a base class. An abstract class contains at least one pure virtual function. You declare a pure virtual function by using a ***pure specifier(= 0)*** in the declaration of a virtual member function in the class declaration

You cannot use an abstract class as a parameter type, a function return type, or the type of an explicit conversion, nor can you declare an object of an abstract class. However, it can be used to declare pointers and references to an abstract class.

An abstract class is used if you want to provide a common, implemented functionality among all the implementations of the component. Abstract classes will allow you to partially implement your class, whereas interfaces would have no implementation for any members whatsoever. In simple words, Abstract Classes are a good fit if you want to provide implementation details to your children but don't want to allow an instance of your class to be directly instantiated.

44. What are the static data members and static member functions?

The static data member of a class is a normal data member but preceded with a static keyword. It executes before main() in a program and is initialized to 0 when the first object of the class is created. It is only visible to a defined class but its scope is of a lifetime.

Syntax:
static Data_Type Data_Member;

The static member function is the member function that is used to access other static data members or other static member functions. It is also defined with a static keyword. We can access the static member function using the class name or class objects.

Syntax:
classname::function name(parameter);

45. What is the main use of the keyword "Volatile"?

Just like its name, things can change suddenly and unexpectantly; So it is used to inform the compiler that the value may change anytime. Also, the volatile keyword prevents the compiler from performing optimization on the code. It was intended to be used when interfacing with memory-mapped hardware, signal handlers, and machine code instruction.
For more information, refer to this [Volatile](#)

46. Define storage class in C++ and name some

Storage class is used to define the features(lifetime and visibility) of a variable or function. These features usually help in tracing the existence of a variable during the runtime of a program.

Syntax:
storage_class var_data_type var_name;
Some types of storage classes:

C++ Storage Class

Storage Class	Keyword	Lifetime	Visibility	Initial Value
Automatic	auto	Function Block	Local	Garbage
External	extem	Whole Program	Global	Zero
Static	static	Whole Program	Local	Zero
Register	register	Function Block	Local	Garbage
Mutable	mutable	Class	Local	Garbage

Examples of storage class

For more information, refer to Storage Class

47. What is a mutable storage class specifier? How can they be used?

Just like its name, the mutable storage class specifier is used only on a class data member to make it modifiable even though the member is part of an object declared as const. Static or const, or reference members cannot use the mutable specifier. When we declare a function as const, this pointer passed to the function becomes const.

48. Define the Block scope variable.

So the scope of a variable is a region where a variable is accessible. There are two scope regions, A global and block or local.

A block scope variable is also known as a local scope variable. A variable that is defined inside a function (like main) or inside a block (like loops and if blocks) is a local variable. It can be used ONLY inside that particular function/block in which it is declared. a block-scoped variable will not be available outside the block even if the block is inside a function.

For more information, refer to Scope of a variable

49. What is the function of the keyword "Auto"?

The auto keyword may be used to declare a variable with a complex type in a straightforward fashion. You can use auto to declare a variable if the initialization phrase contains templates, pointers to functions, references to members, etc. With type inference capabilities, we can spend less time having to write out things the compiler already knows. As all the types are deduced in the compiler phase only, the time for compilation increases slightly but it does not affect the runtime of the program.

For more information, refer to Auto in C++

50. Define namespace in C++.

Namespaces enable us to organize named items that would otherwise have global scope into smaller scopes, allowing us to give them namespace scope. This permits program parts to be organized into distinct logical scopes with names. The namespace provides a place to define or declare identifiers such as variables, methods, and classes.

Or we could say that A namespace is a declarative zone that gives the identifiers (names of types, functions, variables, and so on) within it a scope. Namespaces are used to arrange code into logical categories and to avoid name clashes, which might happen when you have many libraries in your code base.

For more information, refer to Namespace in C++

51. When is void() return type used?

The void keyword, when used as a function return type, indicates that the function does not return a value. When used as a parameter list for a function, void indicates that the function takes no parameters. Non-Value Returning functions are also known as void functions. They're called "void" since they're not designed to return anything. True, but only partially. We can't return values from void functions, but we can certainly return something. Although void functions have no return type, they can return values.

For more information, refer to Void return type.

52. What is the difference between shallow copy and deep copy?

Shallow Copy	Deep Copy
In Shallow copy, a copy of the original object is stored and only the reference address is finally copied. In simple terms, Shallow copy duplicates as little as possible	In Deep copy, the copy of the original object and the repetitive copies both are stored. In simple terms, Deep copy duplicates everything
A shallow copy of a collection is a copy of the collection structure, not the elements. With a shallow copy, two collections now share individual elements.	A deep copy of a collection is two collections with all of the elements in the original collection duplicated.
A shallow copy is faster	Deep copy is comparatively slower.

For more information, refer to Shallow copy VS Deep Copy

53. Can we call a virtual function from a constructor?

Yes, we can call a virtual function from a constructor. But it can throw an exception of overriding.

54. What are void pointers?

Just like its name a void pointer is a pointer that is not associated with anything or with any data type. Nevertheless, a void pointer can hold the address value of any type and can be converted from one data type to another.

For more, information refers to Void Pointer in C++

55. What is '*this*' pointer in C++?

this pointer enables every object to have access to its own address through an essential pointer. All member functions take *this* pointer as an implicit argument.

- **_this_** pointer may be used to refer to the calling object within a member function.
- _this_ pointer is used to pass an object as a parameter to another method.
- Each object gets its own copy of the data member.
- _this_ pointer is used to declare indexers.