**Question: Define C++?**

**Answer:**C++ is a computer programming language that is a superset of C wherein additional features are made in the C language.

**Question: Can we call C++ OOPS? and Why?**

**Answer:**Yes, C++ can be called OOPS. The full form of OOPS is an Object-Oriented Programming System, which means a paradigm that provides an application of various concepts, including data binding, polymorphism, inheritance, and various others.

**Question: Define Class in C++?**

**Answer:**Class is referred to as the designing of the user-defined data type. It reflects the different entities, attributes, and actions.

**Question: Define Object in C++?**

**Answer:**Object is an instance of the class. An object can have fields, methods, constructors, and related. For example, a bike in real life is an object, but it has various features such as brakes, color, size, design, and others, which are instances of its class.

**Question: Define Encapsulation in C++?**

**Answer:**Encapsulation is the process of binding together the data and functions in a class. It is applied to prevent direct access to the data for security reasons. The functions of a class are applied for this purpose. For example, the customers' net banking facility allows only the authorized person with the required login id and password to get access. That is too only for his/her part of the information in the bank data source.

**Question: What is an abstraction in C++?**

**Answer:**An abstraction in C++ is hiding the internal implementations and displaying only the required details. For example, when you send an important message through email, at that time, only writing and clicking the send option is used. This outcome is just the success message that is displayed to confirm that your email has been sent. However, the process followed in transferring the data through email is not displayed because it is of no use to you.

**Question**: **Briefly explain the concept of Inheritance in C++.**

**Answer**: C++ allows classes to inherit some of the commonly used state and behavior from other classes. This process is known as inheritance.

**Question**: **Define access specifier and its various types in C++**

**Answer**: An access specifier offers how it is possible to define how the class members, i.e., functions and variables, will be accessed outside the class's scope. There are three types of access specifier in C++:

* Private – Such class members can’t be accessed outside the class in which they are declared and are only accessible within the same class. Even child classes are disabled to access private members of its parent class.
* **Protected –** In addition to the class in which they are declared, the child classes can access its parent class's protected members.
* Public – Class members declared as public can be accessed throughout the program (code)

**Question: Define a namespace?**

**Answer:** A namespace is used for resolving the name conflict of the identifier, which is accomplished by placing them under various namespaces. This way, it helps in the logical division of the different codes.

**Question: Define a class template?**

**Answer:**A class template is a name given to the generic class. The use of the keyword template is made for defining a class template.

**Question: What is the function of the keyword “Volatile”?**

**Answer:**"Volatile" is a function that helps in declaring that the particular variable is volatile and thereby directs the compiler to change the variable externally- this way, the compiler optimization on the variable reference can be avoided.

**Question: Define storage class in C++? Name some?**

**Answer:**Storage class in C++ specifically resemble life or even the scope of symbols, including the variables, functions, etc. Some of the storage class names in C++ include mutable, auto, static, extern, register, etc.

**Question**:**Can we have a recursive inline function in C++?**

**Answer**:Even though it is possible to call an inline function from within itself in C++, the compiler may not generate the inline code. This is so because the compiler won’t determine the depth of the recursion at the compile time.

Nonetheless, a compiler with a good optimizer is able to inline recursive calls until some depth is fixed at compile-time and insert non-recursive calls at compile time for the cases when the actual depth exceeds run time.

**Question**: **Define an Inline Function in C++? Write its syntax. Is it possible for the C++ compiler to ignore inlining?**

**Answer**: In order to reduce the function call overhead, C++ offers inline functions. As the name suggests, an inline function is expanded in line when it is called.

As soon as the inline function is called, the whole code of the same gets either inserted or substituted at the particular point of the inline function call. The substitution is complete by the C++ compiler at compile time. Small inline functions might increase program efficiency.

The syntax of a typical inline function is:

Inline **return**-type **function**-**name**(parameters)

{

// Function code goes here

}

As the inlining is a request, not a command, the compiler can ignore it.

**Question**: **Explain ‘this’ pointer?**

**Answer**: The ‘this’ pointer is a constant pointer, and it holds the memory address of the current object. It passes as a hidden argument to all the nonstatic member function calls. Also, it is available as a local variable within the body of all the nonstatic functions.

As static member functions can be called even without any object, i.e., with the class name, the ‘this’ pointer is not available for them.

**Question**: **What are the most important differences between C and C++?**

**Answer**:

* C++ supports references while C doesn’t
* Features like friend functions, function overloading, inheritance, templates, and virtual functions are inherent to C++. These are not available in the C programming language.
* In C, exception handling is taken care of in the traditional if-else style. On the other hand, C++ offers support for exception handling at the language level.
* Mainly used input and output in C are scanf() and printf(), respectively. In C++, cin is the standard input stream while cout serves as the standard output stream.
* While C is a [procedural programming language](https://hackr.io/blog/procedural-programming), C++ provides support for both procedural and object-oriented programming approaches.

**Question**: **Why do we need the Friend class and function?**

**Answer**: Sometimes, there is a need for allowing a particular class to access private or protected members of a class. The solution is a friend class, which can access the protected and private members of the class in which it is declared as a friend.

Similar to the friend class, a friend function is able to access private and protected class members. A friend function can either be a global function or a method of some class.

Some important points about friend class and friend function:

* Friendship is not inherited.
* Friendship isn’t mutual, i.e., if some class called Friend is a friend of some other class called NotAFriend, then it doesn’t automatically become a friend of the Friend class.
* The total number of friend classes and friend functions should be limited in a program as the overabundance of the same might lead to a depreciation of the concept of encapsulation of separate classes, which is an inherent and desirable quality of object-oriented programming.

***Intermediate Level Interview Questions***

**Question**: **Explain the significance of vTable and vptr in C++ and how the compiler deals with them**

**Answer**: vTable is a table containing function pointers. Every class has a vTable. vptr is a pointer to vTable. Each object has a vptr. In order to maintain and use vptr and vTable, the C++ compiler adds additional code at two places:

1. In every constructor – This code sets vptr:
   1. Of the object being created
   2. To point to vTable of the class
2. Code with the polymorphic functional call – At every location where a polymorphic call is made, the compiler inserts code in order to first look for vptr using the base class pointer or reference. The vTable of a derived class can be accessed once the vptr is successfully fetched. Address of derived class function show() is accessed and called using the vTable.

**Question**: **How is function overloading different from operator overloading?**

**Answer**: Function overloading allows two or more functions with different type and number of parameters to have the same name. On the other hand, operator overloading allows for redefining the way an operator works for user-defined types.

**Question**: **Is it possible for a C++ program to be compiled without the main() function?**

**Answer**: Yes, it is possible. However, as the main() function is essential for the execution of the program, the program will stop after compiling and will not execute.

**Question: What is a destructor?**

**Answer:**A destructor is the member function of the class. It has the same name as the class name and also prefixed with a tilde symbol. It can be executed automatically whenever an object loses its scope.

**Question: Can we overload a destructor?**

**Answer:**No, a destructor cannot be overloaded, and it has the only form without the parameters.

**Question: What is the default constructor?**

**Answer:**The compiler provides a constructor to every class in case the provider does not offer the same. This is when the programmer provides the constructor with no specific parameters than it is called a default constructor. The code for default constructor can be displayed in the following example.

// Cpp program to illustrate the  
// concept **of** Constructors  
#include <iostream>  
using namespace std;  
class construct {  
public:  
    int a, b;  
    // Default Constructor  
    construct()  
    {  
        a = 10;  
        b = 20;  
    }  
};  
int main()  
{  
    // Default constructor called automatically  
    // when the object is created  
    construct c;  
    cout << "a: " << c.a << endl  
         << "b: " << c.b;  
    return 1;  
}

**Question: Can we provide one default constructor for our class?**

**Answer:**No, we cannot provide one default constructor for our class. When a variable in the class type is set to null, it means that it was never initialized and the outcomes will be zero.

**Question: What is the main difference between the keyword struct and class?**

**Answer:**The keyword struct is used for resembling public members by default, while the keyword class is used for resembling private members by default.

**Question**: **Draw a comparison between C++ and Java**

**Answer**:

* C++ has destructors, which are invoked automatically when an object is destroyed. Java has something called automatic garbage collection
* C++ supports multiple inheritance, operator overloading, pointers, structures, templates, and unions. Java doesn’t have any of them
* Java has a Thread class that is inherited in order to create a new thread. C++ has no inbuilt support for threads
* In C++, a goto statement offers a way to jump from a location to some labeled statement in the same function. There is no goto statement in Java
* C++ run and compile using the compiler, which converts the source code into machine level language. Hence, it is platform-dependent. Java compiler, on the other hand, converts the source code into JVM bytecode, which is platform-independent.

**Question**: **Explain Virtual Functions and the concept of Runtime Polymorphism in C++ with a code example.**

**Answer**: Any function when accompanying the virtual keyword exhibits the behavior of a virtual function. Unlike normal functions that are called in accordance with the type of pointer or reference used, virtual functions are called as per the type of the object pointed or referred.

In simple terms, virtual functions resolve at runtime, not anytime sooner. Use of virtual functions could also be understood as writing a C++ program leveraging the concept of [runtime polymorphism](https://www.quora.com/What-exactly-is-the-runtime-polymorphism-in-C++). Things essential to writing a virtual function in C++ are:

* A base class
* A derived class
* A function with the same name in both the classes i.e. the base class and the derived class
* A pointer or reference of base class type that points or refers, respectively to an object of the derived class

An example demonstrating the use of virtual functions (or runtime polymorphism at play) is:

#include

**using** **namespace** std;

**class** **Base** {

**public**:

**virtual** **void** **show**() { cout<<" In Base \n"; }

};

**class** **Derived**: **public** Base {

**public**:

**void** **show**() { cout<<"In Derived \n"; }

};

**int** **main**(**void**) {

Base \*bp = **new** Derived;

bp->show(); // <- Runtime Polymorphism in Action

**return** 0;

}

In the aforementioned program bp is a pointer of type Base. A call to bp->show() calls show() function of the Derived class. This is because bp points to an object of the Derived class.

**Question**: **What differences separate structure from a class in C++?**

**Answer**: There are two important distinctions between a class and a structure in C++. These are:

1. When deriving a structure from a class or some other structure, the default access specifier for the base class or structure is public. On the contrary, default access specifier is private when deriving a class.
2. While the members of a structure are public by default, the members of a class are private by default

**Question**: **What does a Static member in C++ mean?**

**Answer**: Denoted by the static keyword, a static member is allocated storage, in the static storage area, only once during the program lifetime. Some important facts pertaining to the static members are:

* Any static member function can’t be virtual
* Static member functions don’t have ‘this’ pointer
* The const, const volatile, and volatile declaration aren’t available for static member functions

**Question: Define the Reference variable?**

**Answer:**The reference variable in C++ is the name given to the existing variables. The variable name and reference variable point share the same memory location in C++, which helps in updating the original variable using the reference variable. The code can be displayed in the following example.

#include<iostream>  
using namespace std;  
int **main**()  
{  
  int x = 10;  
  // ref is a reference to x.  
  int& ref = x;  
  // Value of x is now changed to 20  
  ref = 20;  
  cout << "x = " << x << endl ;  
  // Value of x is now changed to 30  
  x = 30;  
  cout << "ref = " << ref << endl ;  
  return 0;  
}

***Advanced Level Interview Questions***

**Question**: **Define the Copy Constructor used in C++ along with its general function prototype. Also, explain the various scenarios in which it is called.**

**Answer**: A member function that initializes an object using another object of the same class is known as a copy constructor in C++. Copy Constructor can also be made private. A call to the Copy Constructor can happen in any of the following 4 scenarios when:

1. The compiler generates a temporary object
2. An object is constructed or based on some another object of the same class
3. An object of the class is returned by value
4. An object of the class is passed (i.e., to a function) by value as an argument

The general function prototype for the Copy Constructor is:

ClassName (const ClassName &old\_obj);  
Point(int x1, int y1) { x=x1; y=y1;}  
Point(const Point &p2) { x=p2.x; y=p2.y; }

**Question: What is a mutable storage class specifier? How can they be used?**

**Answer:**A mutable storage class specifier is applied only on the class's non-static and non-constant member variable. It is used for altering the constant class object's member by declaring it. This can be done by using a storage class specifier.

**Question: What are the differences between a shallow copy and a deep copy?**

**Answer:**The differences between a shallow copy and a deep copy can be stated as under.

|  |  |
| --- | --- |
| Shallow Copy | Deep Copy |
| It allows memory dumping on a bit by bit basis from one object to another. | It allows the copy field, which is done by field from one object to another. |
| It is achieved by using a copy instructor and an overloading assignment operator. | It is used for shallow copy purposes. |

**Question: Define an Abstract class in C++?**

**Answer:**An abstract class in C++ is referred to as the base class, which has at least one pure virtual function. In such a function, a person cannot instantiate an abstract class. This way, an Abstract class a pure virtual function is defined by using a pure specifier which is equal to zero during the declaration of the virtual member function in the class declaration. The code sample can be displayed as follows in example.

// An abstract class  
class Test  
{   
    // Data members of class  
public:  
    // Pure Virtual Function  
    virtual void show() = 0;  
   /\* Other members \*/  
};

**Question; Can we have a String primitive data type in C++?**

**Answer:**No, we cannot have a String Primitive data type in C++. Instead, we can have a class from the Standard Template Library (STL).

**Question: Can we use access specifiers to achieve data hiding in C++?**

**Answer:**Yes, we can use access specifiers to achieve data hiding in C++. These include Private and Protected.

**Question: Define Block scope variable?**

**Answer:**A Block scope variable is the one that is specified as a block using the C++ that can be declared anywhere within the block.

**Question: What are the functions of the scope resolution operator?**

**Answer:**The functions of the scope resolution operator include the following.

1. It helps in resolving the scope of various global variables.
2. It helps in associating the function with the class when it is defined outside the class.

The code of the scope resolution operator can be displayed as follows.

#include <iostream>    
using namespace std;    
int my\_var = 0;  
int main(void) {  
   int my\_var = 0;  
   ::my\_var = 1;  // set global my\_var to 1  
   my\_var = 2;    // set local my\_var to 2  
   cout << ::my\_var << ", " << my\_var;  
   return 0;  
}

**Question: What is the function of the keyword "Auto"?**

**Answer:**The keyword “Auto” is used by default for various local variables to make function work automatically.

**Question: Define a token in C++? Give examples?**

**Answer:**A token is a name given to the various functions in C++ programs. Examples of tokens include a keyword, symbol, string literal, identifier, constant, etc. The code of token in C++ other than C, can be displayed in the following example.

**asm**      **bool**    **catch**      class  
const\_cast   **delete** **dynamic\_cast**   explicit  
export   **false**   **friend**         inline  
mutable  **namespace**   **new** operator  
private  **protected**   **public** reinterpret\_cast  
static\_cast  **template** **this**       throw  
true     **try**     **typeid**     typename  
using    **virtual** **wchar\_t**

**Question**:**What is the ‘diamond problem’ that occurs with multiple inheritance in C++? Explain using an example.**

**Answer**:The diamond problem in C++ represents the inability of the programming language to support hybrid inheritance using multiple and hierarchical inheritance.

Suppose we have a university with some faculty members and some graduate students. A simple inheritance scheme in this scenario might have different types of people in different roles. However, all of them inherit from the same Person class.

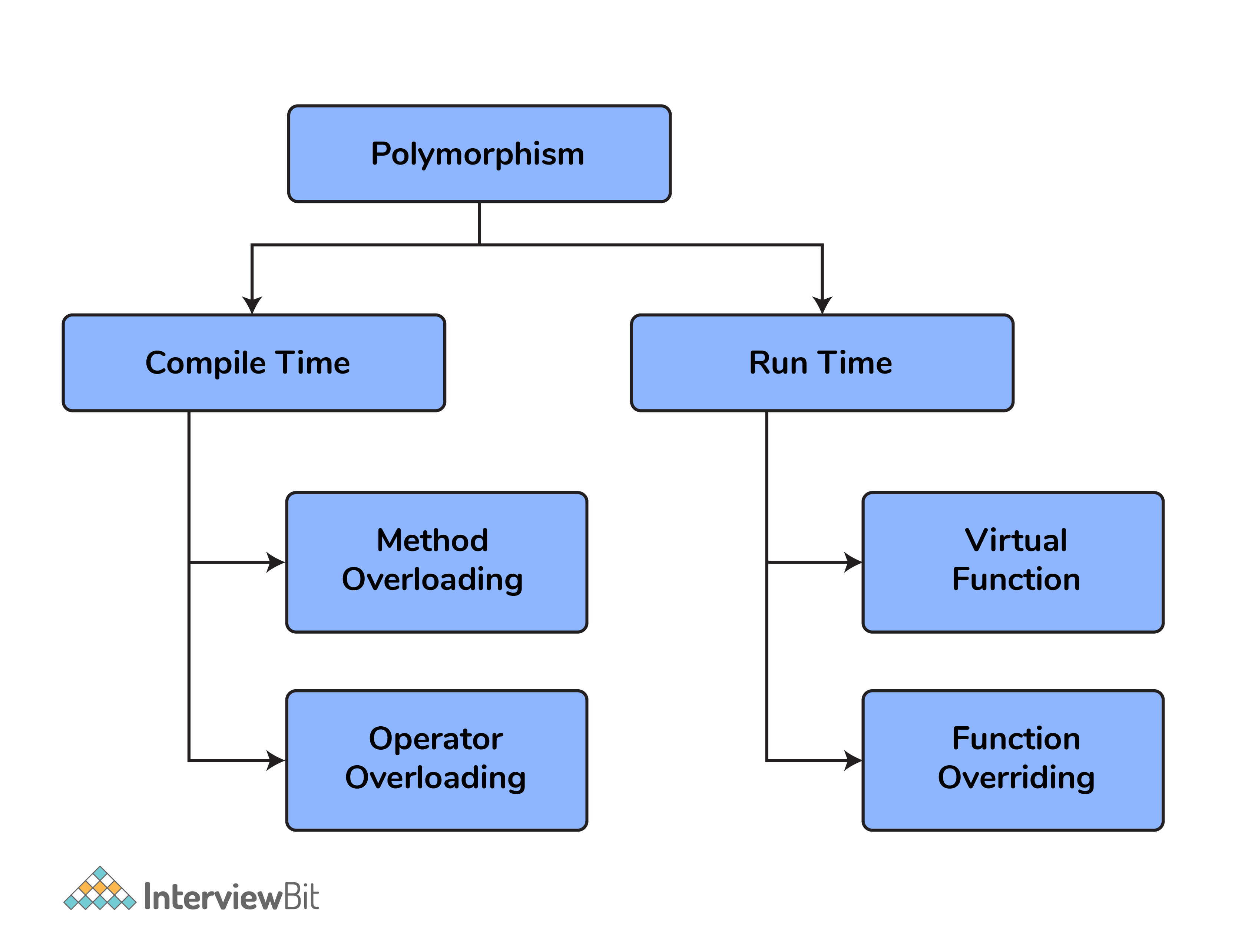
The Person class defines an abstract getRole() method that would then be overridden by its subclasses in order to return the correct role type. Things up till this point is simple, however, if we wish to model the role of a TA or Teaching Assistant then things get weird.

A Teaching Assistant is both a student and a faculty member. This will yield the diamond problem, as illustrated in the figure below:

The problem generates an inheritance diagram resembling a diamond, hence the name, diamond problem.

Which getRole() implementation should the Teaching Assistant inherit? Graduate Student or the Faculty Member? A potential answer might be to have the Teaching Assistant class override the getRole() method and return a newly-defined role, say TA.

However, such an answer would also be far from complete as it will hide the fact that a teaching assistant is both a faculty member and a graduate student.

Polymorphism in C++

**8. Tell me about virtual function**

**Virtual function** is a member function in the base class that you redefine in a derived class. A virtual function is declared using the virtual keyword. When the function is made virtual, C++ determines which function is to be invoked at the runtime based on the type of the object pointed by the base class pointer.

**9. Compare compile time polymorphism and Runtime polymorphism**

The main difference between compile-time and runtime polymorphism is provided below:

| **Compile-time polymorphism** | **Run time polymorphism** |
| --- | --- |
| In this method, we would come to know at compile time which method will be called. And the call is resolved by the compiler. | In this method, we come to know at run time which method will be called. The call is not resolved by the compiler. |
| It provides fast execution because it is known at the compile time. | It provides slow execution compared to compile-time polymorphism because it is known at the run time. |
| It is achieved by function overloading and operator overloading. | It can be achieved by virtual functions and pointers. |
| Example -  **int** **add**(**int** a, **int** b){  **return** a+b;  }  **int** **add**(**int** a, **int** b, **int** c){  **return** a+b+c;  }  **int** **main**(){  cout<<add(2,3)<<endl;  cout<<add(2,3,4)<<endl;  **return** 0;  } | Example -  **class** **base**{  **public**:  **void** **fun**(){  cout<<”base “;  }  };  **class** **derived**: **public** base{  **public**:  **void** **fun**(){  cout<<”derived ”;  }  };  **int** **main**(){  A \*a=**new** B;  a->fun();  **return** 0;  } |

**10. What do you know about friend class and friend function?**

A friend class can access private, protected, and public members of other classes in which it is declared as friends.

Like friend class, friend function can also access private, protected, and public members. But, Friend functions are not member functions.

For example -

**class** **A**{

**private**:

**int** data\_a;

**public**:

A(**int** x){

data\_a=x;

}

**friend** **int** **fun**(A, B);

}

**class** **B**{

**private**:

**int** data\_b;

**public**:

A(**int** x){

data\_b=x;

}

**friend** **int** **fun**(A, B);

}

**int** **fun**(A a, B b){

**return** a.data\_a+b.data\_b;

}

**int** **main**(){

A **a**(10);

B **b**(20);

cout<<fun(a,b)<<endl;

**return** 0;

}

Here we can access the private data of class A and class B.

**15. Is deconstructor overloading possible? If yes then explain and if no then why?**

No destructor overloading is not possible. Destructors take no arguments, so there’s only one way to destroy an object. That’s the reason destructor overloading is not possible.

**16. What do you mean by call by value and call by reference?**

In call by value method, we pass a copy of the parameter is passed to the functions. For these copied values a new memory is assigned and changes made to these values do not reflect the variable in the main function.

In call by reference method, we pass the address of the variable and the address is used to access the actual argument used in the function call. So changes made in the parameter alter the passing argument.

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

A class is called an abstract class whose objects can never be created. Such a class exists as a parent for the derived classes. We can make a class abstract by placing a pure virtual function in the class.

**18. What are destructors in C++?**

A constructor is automatically called when an object is first created. Similarly when an object is destroyed a function called destructor automatically gets called. A destructor has the same name as the constructor (which is the same as the class name) but is preceded by a tilde.

**Example:**

**class** **A**{

**private**:

**int** val;

**public**:

A(**int** x){

val=x;

}

A(){

}

~A(){ //destructor

}

}

**int** **main**(){

A **a**(3);

**return** 0;

}

**19. What are the static members and static member functions?**

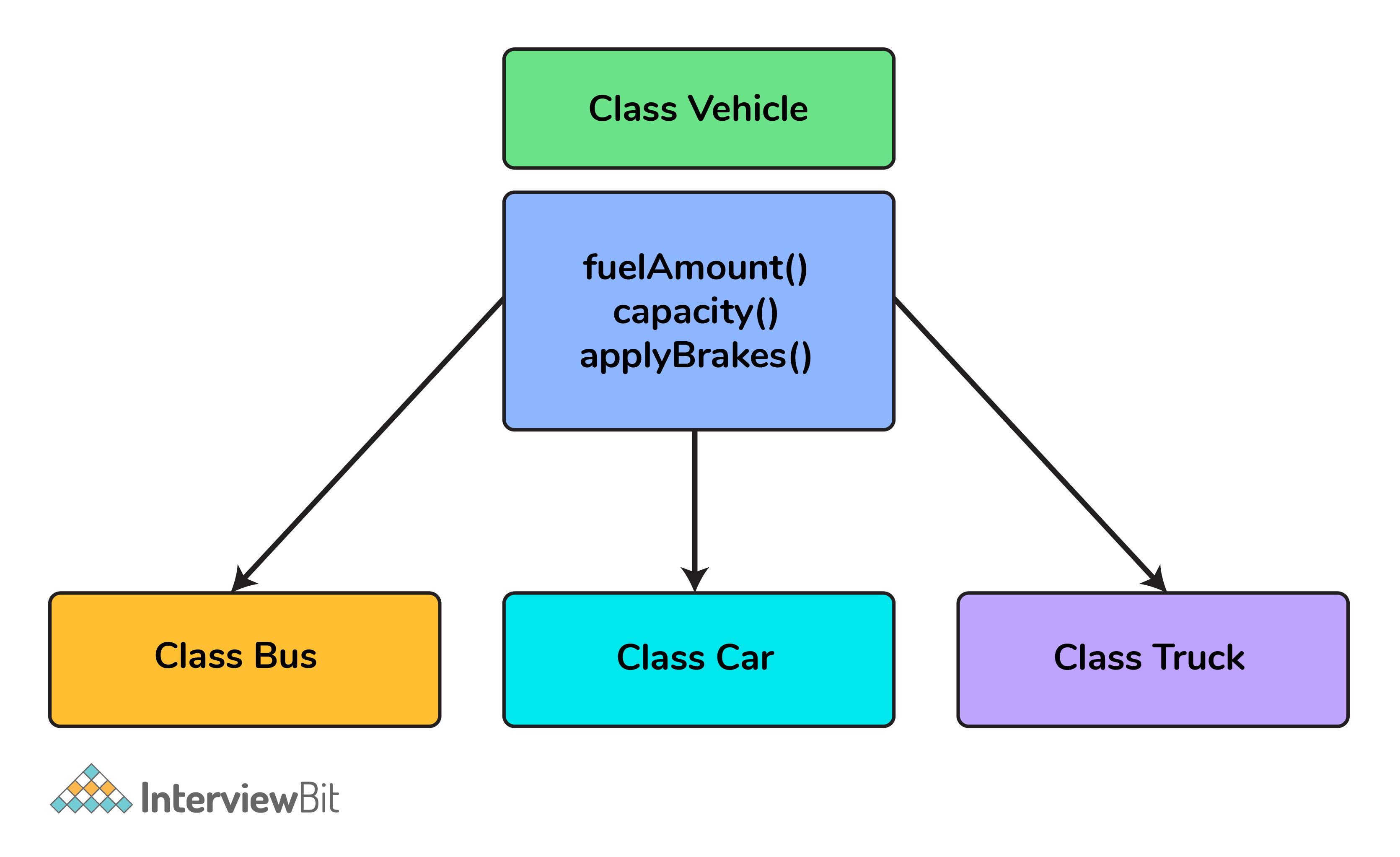
When a variable in a class is declared static, space for it is allocated for the lifetime of the program. No matter how many objects of that class have been created, there is only one copy of the static member. So same static member can be accessed by all the objects of that class.

A static member function can be called even if no objects of the class exist and the static function are accessed using only the class name and the scope resolution operator ::

**20. Explain inheritance**

Inheritance is the process of creating new classes, called derived classes, from existing classes. These existing classes are called base classes. The derived classes inherit all the capabilities of the base class but can add new features and refinements of their own.

**Example-**

Inheritance in C++

Class Bus, Class Car, and Class Truck inherit the properties of Class Vehicle.

The most important thing about inheritance is that it permits code reusability.

**C++ Interview Questions For Experienced**

**21. What is a copy constructor?**

A copy constructor is a member function that initializes an object using another object of the same class.

**Example-**

**class** **A**{

**int** x,y;

A(**int** x, **int** y){

**this**->x=x;

**this**->y=y;

}

};

**int** **main**(){

A **a1**(2,3);

A a2=a1; //default copy constructor is called

**return** 0;

}

We can define our copy constructor. If we don’t define a copy constructor then the default copy constructor is called.

**22. What is the difference between shallow copy and deep copy?**

The difference between shallow copy and a deep copy is given below:

| **Shallow Copy** | **Deep Copy** |
| --- | --- |
| Shallow copy stores the references of objects to the original memory address. | Deep copy makes a new and separate copy of an entire object with its unique memory address. |
| Shallow copy is faster. | Deep copy is comparatively slower. |
| Shallow copy reflects changes made to the new/copied object in the original object. | Deep copy doesn’t reflect changes made to the new/copied object in the original object |

**23. What is the difference between virtual functions and pure virtual functions?**

A virtual function is a member function in the base class that you redefine in a derived class. It is declared using the virtual keyword.

**Example-**

**class** **base**{

**public**:

**virtual** **void** **fun**(){

}

};

A pure virtual function is a function that has no implementation and is declared by assigning 0. It has no body.

**Example-**

**class** **base**{

**public**:

**virtual** **void** **fun**()=0;

};

Here, = sign has got nothing to do with the assignment, and value 0 is not assigned to anything. It is used to simply tell the compiler that a function will be pure and it will not have anybody.

**24. If class D is derived from a base class B. When creating an object of type D in what order would the constructors of these classes get called?**

The derived class has two parts, a base part, and a derived part.  When C++ constructs derived objects, it does so in phases. First, the most-base class(at the top of the inheritance tree) is constructed. Then each child class is constructed in order until the most-child class is constructed last.   
So the first Constructor of class B will be called and then the constructor of class D will be called.

During the destruction exactly reverse order is followed. That is destructor starts at the most-derived class and works its way down to base class.  
So the first destructor of class D will be called and then the destructor of class B will be called.

**25. Can we call a virtual function from a constructor?**

Yes, we can call a virtual function from a constructor. But the behavior is a little different in this case. When a virtual function is called, the virtual call is resolved at runtime. It is always the member function of the current class that gets called. That is the virtual machine doesn’t work within the constructor.

**For example-**

**class** **base**{

**private**:

**int** value;

**public**:

base(**int** x){

value=x;

}

**virtual** **void** **fun**(){

}

}

**class** **derived**{

**private**:

**int** a;

**public**:

derived(**int** x, **int** y):base(x){

base \*b;

b=**this**;

b->fun(); //calls derived::fun()

}

**void** **fun**(){

cout<<”fun inside derived **class**”<<endl;

}

}

**26. What are void pointers?**

A void pointer is a pointer which is having no datatype associated with it. It can hold addresses of any type.

For example-

**void** \*ptr;

**char** \*str;

p=str; // no error

str=p; // error because of type mismatch

We can assign a pointer of any type to a void pointer but the reverse is not true unless you typecast it as

str=(**char**\*) ptr;

**27. What is this pointer in C++?**

The member functions of every object have a pointer named this, which points to the object itself. The value of this is set to the address of the object for which it is called. It can be used to access the data in the object it points to.

**Example**

**class** **A**{

**private**:

**int** value;

**public**:

**void** **setvalue**(**int** x){

**this**->value=x;

}

};

**int** **main**(){

A a;

a.setvalue(5);

**return** 0;

}

**28. How do you allocate and deallocate memory in C++?**

The new operator is used for memory allocation and deletes operator is used for memory deallocation in C++.

**For example-**

**int** value=**new** **int**; //allocates memory for storing 1 integer

**delete** value; // deallocates memory taken by value

**int** \*arr=**new** **int**[10]; //allocates memory for storing 10 int

**delete** []arr; // deallocates memory occupied by arr

**Additional Resources**

<https://www.interviewbit.com/cpp-interview-questions/>

**For Answers of below Questions look above site**

**C++ MCQs**

1.

Which operator can not be overloaded in C++?

+

::

\*

++

2.

What will be the output of the following C++ program:

#**include**<iostream>

**using** **namespace** std;

**int** **main**(){

**int** a=1;

cout<<(a++)\*(++a)<<endl;

**return** 0;

}

1

6

2

3

3.

What will be the value of x in the following C++ program

#**include**<iostream>

**using** **namespace** std;

**int** **main**(){

**int** a=1;

**int** x=(a++)++;

cout<<x<<endl;

**return** 0;

}

Compile Time Error

3

1

2

4.

What is an abstract class?

Class declared with abstract keyword

Class which has exactly one virtual function

Class which hash at least one pure virtual function

None of the above

5.

Consider the following C++ program

#**include**<iostream>

**using** **namespace** std;

**class** **A**{

**public**:

**virtual** **void** **a**()=0;

A(){

cout<<"A ";

}

};

**class** **B**: **public** A

{

**public**:

B(){

cout<<"B ";

}

};

**int** **main**(){

A \*a=**new** B();

**return** 0;

}

What will be output?

A B

B A

Compile-time error

None of the above

6.

What is the size of void in C++?

0

1

2

4

7.

If a base class and derived class each include a member function with the same name. Function from which class will be called if called by an object of the derived class

Member function of the base class

Member function of the derived class

Depend on the parameter

None of the above

8.

Memory used by an array is

Contiguous

Non-contiguous

Not determined

None of the above

9.

Which of the following statement is correct?

An object is an instance of the class

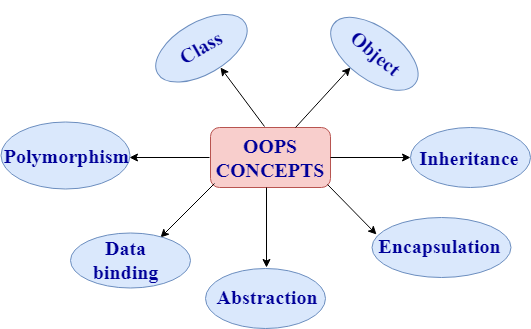
A friend function can access private members of a class

Members of the class are private by default

All of the above

6) What are the various OOPs concepts in C++?

The various OOPS concepts in C++ are:



* **Class:**

The class is a user-defined data type which defines its properties and its functions. For example, Human being is a class. The body parts of a human being are its properties, and the actions performed by the body parts are known as functions. The class does not occupy any memory space. Therefore, we can say that the class is the only logical representation of the data.

**The syntax of declaring the class:**

1. **class** student
2. {
3. //data members;
4. //Member functions
5. }

* **Object:**

An object is a run-time entity. An object is the instance of the class. An object can represent a person, place or any other item. An object can operate on both data members and member functions. The class does not occupy any memory space. When an object is created using a new keyword, then space is allocated for the variable in a heap, and the starting address is stored in the stack memory. When an object is created without a new keyword, then space is not allocated in the heap memory, and the object contains the null value in the stack.

1. **class** Student
2. {
3. //data members;
4. //Member functions
5. }

**The syntax for declaring the object:**

1. Student s = **new** Student();

* **Inheritance:**

Inheritance provides reusability. Reusability means that one can use the functionalities of the existing class. It eliminates the redundancy of code. Inheritance is a technique of deriving a new class from the old class. The old class is known as the base class, and the new class is known as derived class.

**Syntax**

1. **class** derived\_class :: visibility-mode base\_class;

Note: The visibility-mode can be public, private, protected.

* **Encapsulation:**

Encapsulation is a technique of wrapping the data members and member functions in a single unit. It binds the data within a class, and no outside method can access the data. If the data member is private, then the member function can only access the data.

* **Abstraction:**

Abstraction is a technique of showing only essential details without representing the implementation details. If the members are defined with a public keyword, then the members are accessible outside also. If the members are defined with a private keyword, then the members are not accessible by the outside methods.

* **Data binding:**

Data binding is a process of binding the application UI and business logic. Any change made in the business logic will reflect directly to the application UI.

* **Polymorphism:**

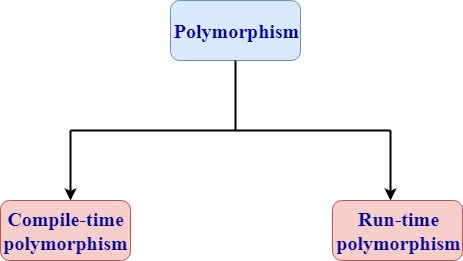
Polymorphism means multiple forms. Polymorphism means having more than one function with the same name but with different functionalities. Polymorphism is of two types:

1. Static polymorphism is also known as early binding.
2. Dynamic polymorphism is also known as late binding.

7) What are the different types of polymorphism in C++?

Polymorphism: Polymorphism means multiple forms. It means having more than one function with the same function name but with different functionalities.

**Polymorphism is of two types:**



* **Runtime polymorphism**

Runtime polymorphism is also known as dynamic polymorphism. Function overriding is an example of runtime polymorphism. Function overriding means when the child class contains the method which is already present in the parent class. Hence, the child class overrides the method of the parent class. In case of function overriding, parent and child class both contains the same function with the different definition. The call to the function is determined at runtime is known as runtime polymorphism.

**Let's understand this through an example:**

1. #include <iostream>
2. **using** **namespace** std;
3. **class** Base
4. {
5. **public**:
6. **virtual** **void** show()
7. {
8. cout<<"javaTpoint";
9. }
10. };
11. **class** Derived:**public** Base
12. {
13. **public**:
14. **void** show()
15. {
16. cout<<"javaTpoint tutorial";
17. }
18. };
20. **int** main()
21. {
22. Base\* b;
23. Derived d;
24. b=&d;
25. b->show();
26. **return** 0;
27. }

**Output:**

javaTpoint tutorial

* **Compile time polymorphism**

Compile-time polymorphism is also known as static polymorphism. The polymorphism which is implemented at the compile time is known as compile-time polymorphism. Method overloading is an example of compile-time polymorphism.

**Method overloading:** Method overloading is a technique which allows you to have more than one function with the same function name but with different functionality.

Method overloading can be possible on the following basis:

* The return type of the overloaded function.
* The type of the parameters passed to the function.
* The number of parameters passed to the function.

**Let's understand this through an example:**

1. #include <iostream>
2. **using** **namespace** std;
3. **class** Multiply
4. {
5. **public**:
6. **int** mul(**int** a,**int** b)
7. {
8. **return**(a\*b);
9. }
10. **int** mul(**int** a,**int** b,**int** c)
11. {
12. **return**(a\*b\*c);
13. }
14. };
15. **int** main()
16. {
17. Multiply multi;
18. **int** res1,res2;
19. res1=multi.mul(2,3);
20. res2=multi.mul(2,3,4);
21. cout<<"\n";
22. cout<<res1;
23. cout<<"\n";
24. cout<<res2;
25. **return** 0;
26. }

**Output:**

6

24

* In the above example, mul() is an overloaded function with the different number of parameters.

10

9) Define token in C++.

A token in C++ can be a keyword, identifier, literal, constant and symbol.

10) Who was the creator of C++?

Bjarne Stroustrup.

11) Which operations are permitted on pointers?

Following are the operations that can be performed on pointers:

* **Incrementing or decrementing a pointer**: Incrementing a pointer means that we can increment the pointer by the size of a data type to which it points.

**There are two types of increment pointers:**

**1. Pre-increment pointer**: The pre-increment operator increments the operand by 1, and the value of the expression becomes the resulting value of the incremented. Suppose ptr is a pointer then pre-increment pointer is represented as ++ptr.

**Let's understand this through an example:**

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **int** a[5]={1,2,3,4,5};
6. **int** \*ptr;
7. ptr=&a[0];
8. cout<<"Value of \*ptr is : "<<\*ptr<<"\n";
9. cout<<"Value of \*++ptr : "<<\*++ptr;
10. **return** 0;
11. }

**Output:**

Value of \*ptr is : 1

Value of \*++ptr : 2

**2. Post-increment pointer**: The post-increment operator increments the operand by 1, but the value of the expression will be the value of the operand prior to the incremented value of the operand. Suppose ptr is a pointer then post-increment pointer is represented as ptr++.

**Let's understand this through an example:**

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **int** a[5]={1,2,3,4,5};
6. **int** \*ptr;
7. ptr=&a[0];
8. cout<<"Value of \*ptr is : "<<\*ptr<<"\n";
9. cout<<"Value of \*ptr++ : "<<\*ptr++;
10. **return** 0;
11. }

**Output:**

Value of \*ptr is : 1

Value of \*ptr++ : 1

* **Subtracting a pointer from another pointer:** When two pointers pointing to the members of an array are subtracted, then the number of elements present between the two members are returned.

12) Define 'std'.

Std is the default namespace standard used in C++.

13) Which programming language's unsatisfactory performance led to the discovery of C++?

C++was discovered in order to cope with the disadvantages of C.

14) How delete [] is different from delete?

Delete is used to release a unit of memory, delete[] is used to release an array.

15) What is the full form of STL in C++?

STL stands for Standard Template Library.

16) What is an object?

The Object is the instance of a class. A class provides a blueprint for objects. So you can create an object from a class. The objects of a class are declared with the same sort of declaration that we declare variables of basic types.

17) What are the C++ access specifiers?

The access specifiers are used to define how to functions and variables can be accessed outside the class.

There are three types of access specifiers:

* **Private**: Functions and variables declared as private can be accessed only within the same class, and they cannot be accessed outside the class they are declared.
* **Public**: Functions and variables declared under public can be accessed from anywhere.
* **Protected**: Functions and variables declared as protected cannot be accessed outside the class except a child class. This specifier is generally used in inheritance.

18) What is Object Oriented Programming (OOP)?

OOP is a methodology or paradigm that provides many concepts. The basic concepts of Object Oriented Programming are given below:

**Classes and Objects**: Classes are used to specify the structure of the data. They define the data type. You can create any number of objects from a class. Objects are the instances of classes.

**Encapsulation**: Encapsulation is a mechanism which binds the data and associated operations together and thus hides the data from the outside world. Encapsulation is also known as data hiding. In C++, It is achieved using the access specifiers, i.e., public, private and protected.

**Abstraction**: Abstraction is used to hide the internal implementations and show only the necessary details to the outer world. Data abstraction is implemented using interfaces and abstract classes in C++.

Some people confused about Encapsulation and abstraction, but they both are different.

**Inheritance**: Inheritance is used to inherit the property of one class into another class. It facilitates you to define one class in term of another class.

19) What is the difference between an array and a list?

* An Array is a collection of homogeneous elements while a list is a collection of heterogeneous elements.
* Array memory allocation is static and continuous while List memory allocation is dynamic and random.
* In Array, users don't need to keep in track of next memory allocation while In the list, the user has to keep in track of next location where memory is allocated.

20) What is the difference between new() and malloc()?

* new() is a preprocessor while malloc() is a function.
* There is no need to allocate the memory while using "new" but in malloc() you have to use sizeof().
* "new" initializes the new memory to 0 while malloc() gives random value in the newly allotted memory location.
* The new() operator allocates the memory and calls the constructor for the object initialization and malloc() function allocates the memory but does not call the constructor for the object initialization.
* The new() operator is faster than the malloc() function as operator is faster than the function.

21) What are the methods of exporting a function from a DLL?

There are two ways:

* By using the DLL's type library.
* Taking a reference to the function from the DLL instance.

22) Define friend function.

Friend function acts as a friend of the class. It can access the private and protected members of the class. The friend function is not a member of the class, but it must be listed in the class definition. The non-member function cannot access the private data of the class. Sometimes, it is necessary for the non-member function to access the data. The friend function is a non-member function and has the ability to access the private data of the class.

**To make an outside function friendly to the class, we need to declare the function as a friend of the class as shown below:**

1. **class** sample
2. {
3. // data members;
4. **public**:
5. **friend** **void** abc(**void**);
6. };

**Following are the characteristics of a friend function:**

* The friend function is not in the scope of the class in which it has been declared.
* Since it is not in the scope of the class, so it cannot be called by using the object of the class. Therefore, friend function can be invoked like a normal function.
* A friend function cannot access the private members directly, it has to use an object name and dot operator with each member name.
* Friend function uses objects as arguments.

**Let's understand this through an example:**

1. #include <iostream>
2. **using** **namespace** std;
3. **class** Addition
4. {
5. **int** a=5;
6. **int** b=6;
7. **public**:
8. **friend** **int** add(Addition a1)
9. {
10. **return**(a1.a+a1.b);
11. }
12. };
13. **int** main()
14. {
15. **int** result;
16. Addition a1;
17. result=add(a1);
18. cout<<result;
19. **return** 0;
20. }

**Output:**

11

23) What is a virtual function?

* A virtual function is used to replace the implementation provided by the base class. 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.
* A virtual function is a member function which is present in the base class and redefined by the derived class.
* When we use the same function name in both base and derived class, the function in base class is declared with a keyword virtual.
* When the function is made virtual, then C++ determines at run-time which function is to be called based on the type of the object pointed by the base class pointer. Thus, by making the base class pointer to point different objects, we can execute different versions of the virtual functions.

**Rules of a virtual function:**

* The virtual functions should be a member of some class.
* The virtual function cannot be a static member.
* Virtual functions are called by using the object pointer.
* It can be a friend of another class.
* C++ does not contain virtual constructors but can have a virtual destructor.

24) When should we use multiple inheritance?

You can answer this question in three manners:

1. Never
2. Rarely
3. If you find that the problem domain cannot be accurately modeled any other way.

25) What is a destructor?

A Destructor is used to delete any extra resources allocated by the object. A destructor function is called automatically once the object goes out of the scope.

**Rules of destructor:**

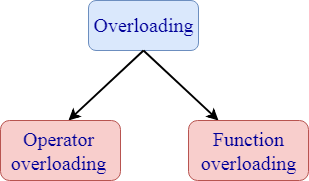
* Destructors have the same name as class name and it is preceded by tilde.
* It does not contain any argument and no return type.

26) What is an overflow error?

It is a type of arithmetical error. It happens when the result of an arithmetical operation been greater than the actual space provided by the system.

27) What is overloading?

* When a single object behaves in many ways is known as overloading. A single object has the same name, but it provides different versions of the same function.
* C++ facilitates you to specify more than one definition for a function name or an operator in the same scope. It is called function overloading and operator overloading respectively.
* **Overloading is of two types:**



**1. Operator overloading:** Operator overloading is a compile-time polymorphism in which a standard operator is overloaded to provide a user-defined definition to it. For example, '+' operator is overloaded to perform the addition operation on data types such as int, float, etc.

**Operator overloading can be implemented in the following functions:**

* Member function
* Non-member function
* Friend function

**Syntax of Operator overloading:**

1. Return\_type classname :: Operator Operator\_symbol(argument\_list)
2. {
3. // body\_statements;
4. }

**2. Function overloading:** Function overloading is also a type of compile-time polymorphism which can define a family of functions with the same name. The function would perform different operations based on the argument list in the function call. The function to be invoked depends on the number of arguments and the type of the arguments in the argument list.

28) What is function overriding?

If you inherit a class into a derived class and provide a definition for one of the base class's function again inside the derived class, then this function is called overridden function, and this mechanism is known as function overriding.

29) What is virtual inheritance?

Virtual inheritance facilitates you to create only one copy of each object even if the object appears more than one in the hierarchy.

30) What is a constructor?

A Constructor is a special method that initializes an object. Its name must be same as class name.

31) What is the purpose of the "delete" operator?

The "delete" operator is used to release the dynamic memory created by "new" operator.

32) Explain this pointer?

This pointer holds the address of the current object.

33) What does Scope Resolution operator do?

A scope resolution operator(::) is used to define the member function outside the class.

34) What is the difference between delete and delete[]?

Delete [] is used to release the array of allocated memory which was allocated using new[] whereas delete is used to release one chunk of memory which was allocated using new.



36) What is the difference between struct and class?

|  |  |
| --- | --- |
| **Structures** | **class** |
| A structure is a user-defined data type which contains variables of dissimilar data types. | The class is a user-defined data type which contains member variables and member functions. |
| The variables of a structure are stored in the stack memory. | The variables of a class are stored in the heap memory. |
| We cannot initialize the variables directly. | We can initialize the member variables directly. |
| If access specifier is not specified, then by default the access specifier of the variable is "public". | If access specifier is not specified, then by default the access specifier of a variable is "private". |
| The instance of a structure is a "structure variable". |  |
| **Declaration of a structure:**  struct structure\_name  {  // body of structure;  } ; | **Declaration of class:**  class class\_name  {  // body of class;  } |
| A structure is declared by using a struct keyword. | The class is declared by using a class keyword. |
| The structure does not support the inheritance. | The class supports the concept of inheritance. |
| The type of a structure is a value type. | The type of a class is a reference type. |

37)

38) What is the difference between function overloading and operator overloading?

**Function overloading:** Function overloading is defined as we can have more than one version of the same function. The versions of a function will have different signature means that they have a different set of parameters.

**Operator overloading:** Operator overloading is defined as the standard operator can be redefined so that it has a different meaning when applied to the instances of a class.

39)

**1) Explain what is a class in C++?**

A class in C++ can be defined as a collection of function and related data under a single name. It is a blueprint of objects. A C++ program can consist of any number of classes.

**2) How can you specify a class in C++?**

By using the keyword class followed by identifier (name of class) you can specify the class in C++. Inside curly brackets, body of the class is defined. It is terminated by semi-colon in the end.

For example,

class name{

// some data

// some functions

};

**3)**

**4) Explain what is C++ objects?**

Class gives blueprints for object, so basically an object is created from a class or in other words an object is an instance of a class. The data and functions are bundled together as a self-contained unit called an object. Here, in the example A and B is the Object.

For example,

Class Student

{

Public:

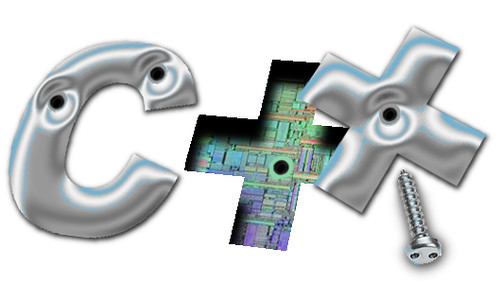
Int rollno;

String name;

} A, B;

**5) Explain what are the characteristics of Class Members in C++?**

* Data and Functions are members in C++,
* Within the class definition, data members and methods must be declared
* Within a class, a member cannot be re-declare
* Other that in the class definition, no member can be added elsewhere

[](https://www.guru99.com/images/2/cpp_interview_questions.jpg)

**6) Explain what is Member Functions in Classes?**

The member function regulates the behaviour of the class. It provides a definition for supporting various operations on data held in the form of an object.

**7) Define basic type of variable used for a different condition in C++?**

The variable used for a different condition in C++ are

* Bool: Variable to store boolean values (true or false)
* Char: Variable to store character types
* int : Variable with integral values
* float and double: Types of variables with large and floating point values

**8) What is namespace std; and what is consists of?**

Namespace std; defines your standard C++ library, it consists of classes, objects and functions of the standard C++ library. You can specify the library by using namespace std or std: : throughout the code. Namespace is used to differentiate the same functions in a library by defining the name.

**9) Explain what is Loop function? What are different types of Loops?**

In any programming language, to execute a set of statements repeatedly until a particular condition is satisfied Loop function is used. The loop statement is kept under the curly braces { } referred as Loop body.

In C++ language, three types of loops are used

* While loop
* For loop
* Do-while loop

**10) Explain how functions are classified in C++ ?**

In C++ functions are classified as

* Return type
* Function Name
* Parameters
* Function body

**11) Explain what are Access specifiers in C++ class? What are the types?**

Access specifiers determine the access rights for the statements or functions that follow it until the end of class or another specifier is included. Access specifiers decide how the members of the class can be accessed. There are three types of specifiers.

* Private
* Public
* Protected

**12) Explain what are Operators and explain with an example?**

Operators are specific operands in C++ that is used to perform specific operations to obtain a result. The different types of operators available for C++ are Assignment Operator, Compound Assignment Operator, Arithmetic Operator, Increment Operator and so on.

For example arithmetic operators, you want to add two values a+b

#include

Using namespace std;

main ()

{

int a= 21 ;

int b= 10 ;

int c;

c= a + b;

cout << "Line 1- Value of c is : " << c << endl ;

return 0;

}

It will give the output as 31 when you run the command

**13) What is the C-style character string?**

The string is actually a one-dimensional array of characters that is terminated by a null character '\0'.

For example, to type hello word

#include

Using namespace std;

int main ()

{

char greeting[6] = { 'H' , 'e' , 'l' ,'l' , 'o' , '\0'};

cout << "Greeting message:" ;

cout << greeting << endl;

return 0;

}

On executing this code it will give the result like? Greeting message: Hello

**14) Explain what is a reference variable in C++?**

A reference variable is just like a pointer with few differences. It is declared using & Operator. In other words, reference is another name for an already existing variable.

**15) Explain what is Polymorphism in C++?**

Polymorphism in C++ is the ability to call different functions by using only one type of the function call. Polymorphism is referred to codes, operations or objects that behave differently in a different context.

For example, the addition function can be used in many contests like

* 5+5 Integer addition
* Medical+Internship The same ( + ) operator can be used with different meaning with strings
* 3.14 + 2.27 The same ( + ) operator can be used for floating point addition

**16) Explain what is data abstraction in C++?**

Data abstraction is a technique to provide essential information to the outside world while hiding the background details. Here in below example you don’t have to understand how cout display the text “Hello guru99” on the user screen and at the same time implementation of cout is free to change

For example,

#include

Using namespace std;

int main ( )

{

cout << "Hello guru99" <<endl;

return 0 ;

}

**17) Explain what is C++ exceptional handling?**

The problem that arises during execution of a program is referred as exceptional handling. The exceptional handling in C++ is done by three keywords.

* Try: It identifies a block of code for which particular exceptions will be activated
* Catch: The catch keyword indicates the catching of an exception by an exception handler at the place in a program
* Throw: When a problem exists while running the code, the program throws an exception

**18) Explain what is data encapsulation in C++?**

Encapsulation is an object oriented programming concept (oops) which binds together the data and functions. It is also referred as data hiding mechanism.

**19) Mention what are the types of Member Functions?**

The types of member functions are

* Simple functions
* Static functions
* Const functions
* Inline functions
* Friend functions

**20) Mention what are the decision making statements in C++? Explain if statement with an example?**

The decision making statements in C++ are

* if statement
* switch statement
* conditional operator

For example, we want to implement if condition in C++

#include

int main ( )

{

int, x, y;

X= 10;

Y= 5;

if (x > y)

{

Cout << "x is greater than y";

}

}

**21) Explain what is multi-threading in C++?**

To run two or more programs simultaneously multi-threading is useful. There are two types of

* Process-based: It handles the concurrent execution of the program
* Thread-based: It deals with the concurrent execution of pieces of the same program

**22) Explain what is upcasting in C++?**

Upcasting is the act of converting a sub class references or pointer into its super class reference or pointer is called upcasting.

**23) Explain what is pre-processor in C++?**

Pre-processors are the directives, which give instruction to the compiler to pre-process the information before actual compilation starts.

**24) Explain what is COPY CONSTRUCTOR and what is it used for?**

COPY CONSTRUCTOR is a technique that accepts an object of the same class and copies its data member to an object on the left part of the assignment.

### ****1. What is C++?****

As an extension of the C language, C++ was developed by Bjarne Stroustrup as a general purpose cross-platform language which gives programmers a high level of control over system resources and memory.

### ****2. What is namespace in C++?****

If there are two or more functions with the same name defined in different libraries then how will the compiler know which one to refer to? Thus namespace came to picture. A namespace defines a scope and differentiates functions, classes, variables etc. with the same name available in different libraries. The namespace starts with the keyword “namespace”. The syntax for the same is as follows:

|  |  |
| --- | --- |
| 1  2  3 | namespace namespace\_name {     // code declarations  } |

### ****3. How to input string in C++?****

There are three ways to input a string, using cin, get, and getline. All three methods are mentioned in the sample program below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #include <iostream>  using namespace std;    int main()  {      char s[10];        cout << "Enter a string: ";      cin >> str;        cout << "\nEnter another string: ";      cin.get(s, 10);        getline(cin, str);        return 0;  } |

### ****4. What is operator overloading in C++?****

An overloaded declaration is a declaration in the same scope of function or operator declared with the same name more than once.

### ****5. How to learn C++?****

C++ is a programming language which is an extension of C. Thus, one should prefer to learn C first (it’s not necessary). After learning C, then understand the basic difference between C and C++. Implement all the basic programs you learnt in C in C++ also. Then dive into the OOPs concept of C++. Do as many hands-on as possible to understand basic OOPs, and then dive into advanced level OOPs. When all the basics are clear, build a small game to understand the structure and remain concepts if any. By following all these steps one can learn C++.

### ****6. What is the difference between C and C++?****

The difference between c and c++ is that C++ is a object oriented language, which means that it has all the features of C as well as its own thing that is the concept of OOP. C++ has many functionalities of OOP that are missing from C such as encapsulation, abstraction, classes, objects, etc.

### ****7. How to reverse a string in C++?****

To reverse a string, a sample code is mentioned below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | #include<iostream>  #include<string.h>  using namespace std;  int main ()  {      char n[50], t;      int i, j;      cout << "Enter a string : ";      gets(n);      i = strlen(n) - 1;      for (j = 0; j < i; j++,i--)      {          t = s[j];          s[j] = s[i];          s[i] = t;      }      cout << "\nReverse string : " << s;      return 0;  } |

### ****8. What is template in C++?****

A template in C++ is used to pass data types as parameters . These make it easier and more simpler to use classes and functions.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | template <typename T>                    int fun (T a,T b)                  {                          return (a+b);                  }                    int main(){                          cout<<fun<int>(11,22);                  } |

### ****9. What is using namespace std in C++?****

Using namespace std in C++ tells the compiler that you will be making use of the name space called ‘std’. The ‘std’ namespace contains all the features of the standard library. You need to put this statement at the start of all your C++ codes if you don’t want to keep on writing std:: infront of every variable/string or whatever standard library feature you are making use of, as it becomes tedious to do so.

### ****20. What is type casting in C++?****

Type casting in C is used to change the data type. They are of two types: Implicit Type Conversion: It is automatic. Explicit Type Conversion: It is user-defined.

### ****33. What is function overriding in C++ ?****

When a function with same name is present in both parent and child class then it is called function overriding.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | #include <iostream>  using namespace std;  class parent {  public:     void display(){        cout<<"Parent Class";     }  };  class child: public parent{  public:     void display() {        cout<<"Child Class";     }  };  int main() {     child o = parent();     o.display();     return 0;  } |

### ****44. How to generate random numbers in C++ with a range?****

Using the rand() function we can generate random numbers in C++ within a range.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | #include <iostream>  #include <random>  int main()  {     int max=100, min=54,i;     int range = max - min + 1;     for (i=min; i<max;i++)      {          int num = rand() % range + min;          cout<<num;      }      return 0;  } |

### ****49. What is iterator in C++?****

Any object which has an ability to iterate through elements of the range it has been pointing to is called iterator.

### ****50. What is :: in C++?****

:: is called a scope resolution operator which is used to access global variables with the same name as of local variables, for defining functions outside the class, for accessing static variables, and for referring to a class inside of another class.

### ****51. What is enum in C++?****

enum is abbreviation of Enumeration which assigns names to integer constant to make a program easy to read. Syntax for the same:  
  
enum enum\_name{const1, const2, ……. };

### ****52. What is endl in C++?****

Endl is a predefined object of ostream class to insert a new line characters.

### ****53. How to save a file in C++?****

When you have written code in the file (notepad),save the file as “hello.cpp.” If you want to write in a file using C++ code, you can do it using iostream and fstream libraries in C++.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | #include <iostream>  #include <fstream>  using namespace std;    int main () {    ofstream file\_name;    file\_name.open ("sample.txt");    file\_name<< "Write in the file";    file\_name.close();    return 0;  } |

### ****54. Which operators can be overloaded in C++?****

List of operators that can be overloaded are:  
  
+ , – , \* , / , % , ^, & , | , ~ , !, =, ++ , –, ==, != , && , ||  
+= , -= , /= , %= , ^= , &=, |= , \*= , = , [] , (), ->, ->\* , new , new [] , delete , delete []

### ****55. How to include all libraries in C++?****

The library <bits/stdc++.h> in c++ is used to include all the libraries.

### ****57. What is an expression in C++?****

An expression is a combination of operators, constants and variables. There seven types of expressions for examples:  
  
– Constant expressions: 89 +10/4.0  
– Integral expressions: x \* y  
– Floating expressions: 17.89  
– Relational expressions: a<=b  
– Logical expressions: a > b && a == 7  
– Pointer expressions: \*ptr  
– Bitwise expressions: p << 5

### ****58. How to write a class in C++?****

A class in C++ is the building block that leads to Object-Oriented programming and is a user-defined data type which holds data and functions. The syntax to write a class in C++ is as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Class (keyword) Class\_Name (this is user defined)  {      Access specifier: // private, public, protected      Data members //int, char, float, double etc. variables to be used      Member function() { }  // Methods to access data members  };     //Class end |

For example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | class Sample  {      // Access specifier      private:        // Data Members      string s;        // Member Functions()      void printname()      {         cout << s;      }  }; |

### ****60. How to use strcmp function in C++?****

strcmp() function is an in-built function of <string.h> header file which takes two strings as arguments and compares these two strings lexicographically.  
  
The syntax of the function is as follows:  
int strcmp(const char \*l, const char \*r );

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | #include<stdio.h>  #include<string.h>  int main()  {      // z has greater ASCII value than g      char a[] = "zfz";      char b[] = "gfg";        int r = strcmp(a, b);        if (r==0)          printf("Strings are equal");      else          printf("Strings are unequal");        printf("%d" , r);        return 0;  } |

### ****61. How to write to a file in C++?****

A file is read in c++ using a fstream header file.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | #include <iostream>  #include <fstream>  using namespace std;  int main()  {      ofstream fout;      string r;        fout.open("test.txt");        while (fout) {          getline(cin, r);          if (r == "-1")              break;          fout << line << endl;      }      fout.close();        ifstream fin;      fin.open("test.txt");      while (fin) {           getline(fin, line);            cout << line << endl;      }      fin.close();      return 0;  } |

### ****62.****

## ****Advanced C++ Interview Questions****

### ****68. What is inline function in C++?****

Inline functions are functions used to increase the execution time of a program. Basically, if a function is inline, the compiler puts the function code wherever the function is used during compile time. The syntax for the same is as follows:  
  
inline return\_type function\_name(argument list) {  
//block of code  
}

### ****69. What is friend function in C++?****

A friend function has the access rights to all private and protected members of the class.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | class Circle{     double radius;       public:        friend void printradius( Circle c );    };  void printradius(Circle c ) {     /\* Because printradius() is a friend of Circle, it can     directly access any member of this class \*/     cout << "Radius of circle: " << c.width;  }    int main() {     Circle c;       // Use friend function to print the radius.     printradius( c);       return 0;  } |

### ****70. What is exception handling in C++?****

Exceptions are errors that happen during execution of code. To handle them we use throw, try & catch keywords.

### ****74. What are character constants in C++?****

Character constant are members of the character set in which a program is written which is surrounded by single quotation marks (‘).

### ****75. What are templates in C++?****

A feature that allows functions and classes to operate with generic types that means a function or class can work on different data types without being rewritten is called template.

### ****76. How to sort vector in C++?****

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | "#include <bits/stdc++.h>  using namespace std;  int main()  {      vector<int> vec{ 1,9,4,3,2,8,5,7};        sort(vec.begin(), vec.end());        for (auto x : v)          cout << x << "" "";        return 0;  }  " |

### ****77. What is pure virtual function in C++?****

A pure virtual function is a type of virtual function which does not have implementation, but is only declared. It is declared by assigning 0 in declaration. Syntax for the same is as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | class Test  {      // Data members of class  public:        virtual void show() = 0;       /\* Other members \*/  }; |

### ****78. How to use map in C++?****

Associative containers storing a combination of a key value or mapped value is called Maps. Syntax: map<key\_type , value\_type> map\_name;

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | #include <iostream>  #include <iterator>  #include <map>    using namespace std;    int main()  {      map<int, int> test;        // inserting elements      test.insert(pair<int, int>(1, 2));      test.insert(pair<int, int>(2, 3));        map<int, int>::iterator itr;      for (itr = test.begin(); itr != test.end(); ++itr) {          cout << itr->first          cout << itr->second << '\n';      }   return 0;  ) |

### ****79. How to empty a vector in C++?****

Std::vector::empty tests whether a vector is empty or not. A sample code for illustrating the same is as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | #include <iostream>  #include <vector>    int main ()  {    std::vector<int> vec;    int add (0);      for (int i=1;i<=5;i++) vec.push\_back(i);      while (!vec.empty())    {       add+= vec.back();       vec.pop\_back();    }      std::cout << add;      return 0;  } |

### ****80. What is visual C++?****

C++ is a standardized language and Visual C++ is a product that implements the standard of C++. One can write portable C++ programs using Visual C++, but one can also use Microsoft-only extensions which destroys portability but enhances your productivity.

### ****81. How to remove segmentation fault in C++?****

Segmentation fault indicates an error memory corruption. In layman terms, when a piece of code tries to do read and write operation in a read only location in memory. Below are the reasons and solutions for segmentation error:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | Reason: Accessing an address that is freed  int\* p = malloc(8);  \*p = 100;     free(p);   \*p = 110;    Solution: Before freeing the pointer check the assignment or any operation required to perform.    Reason: Accessing out of array index bounds  int arr[2];  arr[3] = 10;  Solution: Correcting the array bound    Reason: Improper use of scanf()  int n = 2;  scanf("%d",n);  Solution: To avoid this is the only solution    Reason: Dereferencing uninitialized pointer  int \*p;  printf("%d",\*p);  Solution: A pointer must point to valid memory before accessing it.    Reason: Stack Overflow  Solution: It can be resolved by having a base condition to return from the recursive function. |

### ****82. What is stl in C++ with example?****

STL in C++ is a library and abbreviation of Standard Template Library. STL is a generalized library that provides common programming data structures/ container classes, functions, algorithms, and iterators. STL has four components  
  
– Algorithms: Searching and sorting algorithms such as binary search, merge sort etc.  
– Containers: Vector, list, queue, arrays, map etc.  
– Functions: They are objects that act like functions.  
– Iterators: It is an object that allows transversing through elements of a container, e.g., vector<int>::iterator.

### ****83. What is flush in C++?****

std::flush synchronizes the stream buffer with its controlled output sequence.

### ****84. How to initialize a 2d vector in C++?****

The syntax to initialize a 2d vector is as follows:  
std::vector<std::vector<int> > name\_of\_vector;  
  
For example: std::vector<std::vector<int> > v { { 1, 2, 1 },  
{ 2, 6, 7 } };

## ****C++ Programming Interview Questions****

### ****85. How to input a string in C++?****

There are two ways to input a string in C++.  
Way 1:

|  |  |
| --- | --- |
| 1  2  3 | string str\_1;  cout << "Enter the string";  cin >> str\_1; |

Way 2:

|  |  |
| --- | --- |
| 1  2  3 | string str\_1;  cout << "Enter the string";  getline (cin, str\_1) |

### ****86. 87. How to find length of array in C++?****

The length of an array in C++ can be calculated using sizeof() function. The code depicting the same is mentioned below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | #include <iostream>  using namespace std;    void main()  {    int a[] = {0,1,2,3,4,5};    int a\_size = sizeof(a)/sizeof(a[0]);    cout << ""Size of the array is: "" << a\_size;  }  " |

### ****88. How to convert int to string in C++?****

There are 2 approaches to convert integer variables to string. Both the approaches with a sample code are mentioned below.

Approach 1:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | #include<iostream>  #include<string>  using namespace std;  void main()  {      int n= 1;      string s= to\_string(n);      cout << s;  } |

Approach 2:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | #include<iostream>  #include <sstream>  #include <string>  using namespace std;  int main()  {      int n = 17;        // declaring output string stream      ostringstream s1;        // Sending a number as a stream into output str      s<< n;      // the str() converts number into string      string fin = s.str();      // Displaying the string      cout << fin;      return 0;  } |

### ****89. How to sort a string in C++?****

To sort a string, the sort function in c++ can be used. The sample code for the same is as follows.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | #include<iostream>  #include <stdio.h>  using namespace std;  void str\_sort(string &amp;s)  {     sort(s.begin(), s.end());     cout << s;  }    int main()  {      string s = ""anmbdfc"";      str\_sort(s);      return 0;  }    Output: abcdfmn  " |

### ****90. How to convert string to int in C++?****

This can be done using the stoi() or atoi() function:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | int main()         {                 string str\_value = "1122";                 int int\_value = stoi(str\_value);         }         or         int main()         {                 const char \*str\_ptr = "1122";                 int int\_value = atoi(str\_value);         } |

### ****91. How to compare two strings in C++?****

Two string can be compared using strcmp() function. It return boolean value; if it returns ‘0’ then the strings are same and if it returns ‘1’ then the two strings are not same.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | int main()  {          if (strcmp(str\_value\_a, str\_value\_b) == 0)                  cout<<"These strings are the same";          else                  cout<<"These strings are not the same";  } |

### ****92. How to set precision in C++?****

Precision in C++ can be set by using the following functions- floor(), ceil(), round(), trunc() and setprecision(). Using setprecision() in a program.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | #include<bits/stdc++.h>  using namespace std;    void main()  {      double p = 8.04149;      cout << fixed << setprecision(2) << pi<<endl;  }    The output will be 8.04. |

### ****93. What is this pointer in C++?****

Using this pointer, every object is provided access to its own address. It is an implicit parameter to all member functions. Only member functions have a this pointer and not a friend function.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | #include<iostream>  using namespace std;    class sample  {  private:     int a;  public:     void set\_var (int a)     {             this->a = a;     }     void print()       {         cout << x ;       }  };    int main()  {     sample obj;     int a = 2;     obj.set\_var(a);     obj.print();     return 0;  } |

### ****94. How to concatenate two strings in C++?****

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | "#include <iostream>  using namespace std;  int main  {      string str\_a = ""Hi, My name is"";      string str\_b = ""Raj"";      string str\_c = str\_a + str\_b;      cout<<""The string is:""<<str\_c;      return 0;  }" |

### ****95. How to find string length in C++?****

You can do this by using size() function.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | int main()  {          string str[]=""Hello World!"";          int g = str.size();          cout<<""The size of the string is""<<g;  }  or you can use the strlen function." |

### ****96. How to initialize vector in C++?****

There are multiple ways to do it:  
You can do it like arrays:  
vector<int> value{ 11, 22, 33 };  
or by pushing values one by one:  
vector<int> value;  
value.push\_back(11);  
value.push\_back(22);  
value.push\_back(33);  
value.push\_back(44);  
value.push\_back(55);

### ****97. How to use getline in C++?****

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | "int main()  {          string yourname;          getline (cin,yourname);    }" |

## ****OOPS Interview Questions**** ****C++****

### ****114. How to access private members of a class in C++?****

Private members of the class are not accessible by object or function outside the class. Only functions inside the class can access them or friend functions. However, pointers can be used to access private data members outside the class. Sample code is as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | #include <iostream>  using namespace std;    class sample\_test{  private:      int n;    public:      sample\_test() { n = 45; }      int display() {  return n;       }  }; |

### ****116.****

### ****117.****

### ****118.****

### ****119. What is runtime polymorphism in C++?****

Polymorphism means having many forms either it is a function or operator in programming.  
Runtime polymorphism is achieved by function overriding.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | #include <bits/stdc++.h>  using namespace std;    class parent  {  public:        void print()      { cout<< ""base class""; }  };    class child:public parent  {  public:        void print()      { cout<< ""derived class""; }  };    int main()  {      parent \*p;      child c;      p = &c;        //virtual function, binded at runtime (Runtime polymorphism)      p->print();      return 0;  }  " |

### ****120. What is copy constructor in C++?****

A copy constructor is a constructor which creates an object by initializing it with an object of the same class, which has been created previously. The syntax for copy constructor is as follows:  
  
classname (const classname &obj) {  
// body of constructor  
}

### ****121. How is modularity introduced in C++?****

Modularity is a way of mapping encapsulated abstractions into real and physical modules which is closely related to Encapsulation. It is a concept in which separate programs are divided into separate modules.  
  
For example, when building a house it is built in modular way. First foundation is laid, then structure is made and so on.

### ****122. What is the size of empty class in C++?****

Size of an empty class is 1 byte generally just to ensure that the two different objects will have different addresses.

**Q #1) What is the basic structure of a C++ program?**

**Answer:** The basic structure of a C++ program is shown below:

|  |
| --- |
| #include<iostream.h>  int main()  {                  cout<<”Hello,World!”;                  return 0;  } |

The first line that begins with “***#***” is a **preprocessor directive**. In this case, we are using ***include*** as a directive which tells the compiler to include a header while “***iostream.h***” which will be used for basic input/output later in the program.

The next line is the “main” function that returns an integer. The main function is the starting point of execution for any C++ program. Irrespective of its position in the source code file, the contents of the main function are always executed first by the C++ compiler.

In the next line, we can see open curly braces that indicate the start of a block of code. After this, we see the programming instruction or the line of code that uses the count which is the standard output stream (its definition is present in iostream.h).

This output stream takes a string of characters and prints it to a standard output device. In this case, it is, “Hello, World!”. Please note that each C++ instruction ends with a semicolon (;), which is very much necessary and omitting it will result in compilation errors.

Before closing the braces}, we see another line “return 0;”. This is the returning point to the main function.

Every C++ program will have a basic structure as shown above with a preprocessor directive, main function declaration followed by a block of code and then a returning point to the main function which indicates successful execution of the program.

**Q #2) What are the Comments in C++?**

**Answer:** Comments in C++ are simply a piece of source code ignored by the compiler. They are only helpful for a programmer to add a description or additional information about their source code.

**In C++ there are two ways to add comments:**

* //single-line comment
* /\* block comment \*/

The first type will discard everything after the compiler encounters “//”. In the second type, the compiler discards everything between “/\*” and “\*/”.

Variables, Data Types, And Constants

**Q #3) Difference between Declaration and Definition of a variable.**

**Answer:** The declaration of a variable is merely specifying the data type of a variable and the variable name. As a result of the declaration, we tell the compiler to reserve the space for a variable in the memory according to the data type specified.

**Example:**

|  |
| --- |
| int Result;  char c;  int a,b,c; |

All the above are valid declarations. Also, note that as a result of the declaration, the value of the variable is undetermined.

Whereas, a definition is an implementation/instantiation of the declared variable where we tie up appropriate value to the declared variable so that the linker will be able to link references to the appropriate entities.

**From above Example**,

Result = 10;

C = ‘A’;

These are valid definitions.

**Q #4) Comment on Local and Global scope of a variable.**

**Answer:** The scope of a variable is defined as the extent of the program code within which the variable remains active i.e. it can be declared, defined or worked with.

**There are two types of scope in C++:**

1. **Local Scope:**A variable is said to have a local scope or is local when it is declared inside a code block. The variable remains active only inside the block and is not accessible outside the code block.
2. **Global Scope:**A variable has a global scope when it is accessible throughout the program. A global variable is declared on top of the program before all the function definitions.

**Example:**

|  |
| --- |
| #include <iostream.h>  Int globalResult=0; //global variable  int main()  {  Int localVar = 10; //local variable.  …..    } |

**Q #5) What is the precedence when there are a Global variable and a Local variable in the program with the same name?**

**Answer:**Whenever there is a local variable with the same name as that of a global variable, the compiler gives precedence to the local variable.

**Example:**

|  |
| --- |
| #include <iostream.h>   int globalVar = 2;  int main()  {   int globalVar = 5;   cout<<globalVar<<endl;  } |

The output of the above code is 5. This is because, although both the variables have the same name, the compiler has given preference to the local scope.

**Q #6) When there are a Global variable and Local variable with the same name, how will you access the global variable?**

**Answer:** When there are two variables with the same name but different scope, i.e. one is a local variable and the other is a global variable, the compiler will give preference to a local variable.

In order to access the global variable, we make use of a “***scope resolution operator (::)***”. Using this operator, we can access the value of the global variable.

**Example:**

|  |
| --- |
| #include<iostream.h>  int x= 10;  int main()  {   int x= 2;   cout<<”Global Variable x = “<<::x;   cout<<”\nlocal Variable x= “<<x;  } |

**Output:**

Global Variable x = 10  
local Variable x= 2

**Q #7) How many ways are there to initialize an int with a Constant?**

**Answer:** **There are two ways:**

* The first format uses traditional C notation.  
  int result = 10;
* The second format uses the constructor notation.  
  int result (10);

Constants

**Q #8) What is a Constant? Explain with an example.**

**Answer:**A constant is an expression that has a fixed value. They can be divided into integer, decimal, floating-point, character or string constants depending on their data type.

Apart from the decimal, C++ also supports two more constants i.e. octal (to the base 8) and hexadecimal (to the base 16) constants.

**Examples of Constants:**

* 75 //integer (decimal)
* 0113 //octal
* 0x4b //hexadecimal
* 3.142 //floating point
* ‘c’ //character constant
* “Hello, World” //string constant

**Note:** When we have to represent a single character, we use single quotes and when we want to define a constant with more than one character, we use double-quotes.

**Q #9) How do you define/declare constants in C++?**

**Answer:** In C++, we can define our own constants using the**#define** preprocessor directive.

***#define Identifier value***

**Example:**

|  |
| --- |
| #include<iostream.h>  #define PI 3.142  int main ()  {                   float radius =5, area;                   area = PI \* r \* r;                   cout<<”Area of a Circle = “<<area;  } |

**Output:** Area of a Circle = 78.55

As shown in the above example, once we define a constant using #define directive, we can use it throughout the program and substitute its value.

We can declare constants in C++ using the “***const***” keyword. This way is similar to that of declaring a variable, but with a const prefix.

**Examples of declaring a constant**

const int pi = 3.142;  
const char c = “sth”;  
const zipcode = 411014;

In the above examples, whenever the type of a constant is not specified, the C++ compiler defaults it to an integer type.

Operators

**Q #10) Comment on Assignment Operator in C++.**

**Answer:** Assignment operator in C++ is used to assign a value to another variable.

a = 5;

This line of code assigns the integer value **5**to variable **a**.

The part at the left of the =operator is known as an ***lvalue***(left value) and the right as ***rvalue*** (right value). **L*value*** must always be a variable whereas the right side can be a constant, a variable, the result of an operation or any combination of them.

The assignment operation always takes place from the right to left and never at the inverse.

One property which C++ has over the other programming languages is that the assignment operator can be used as the *rvalue*(or part of an *rvalue*) for another assignment.

**Example:**

a = 2 + (b = 5);

is equivalent to:

b = 5;  
a = 2 + b;

Which means, first assign **5**to variable **b**and then assign to **a,**the value **2**plus the result of the previous expression of **b**(that is 5), leaves **a**with a final value of **7**.

**Thus, the following expression is also valid in C++:**

a = b = c = 5;

assign 5 to variables **a**, **b**and **c**.

**Q #11) What is the difference between equal to (==) and Assignment Operator (=)?**

**Answer:** In C++, equal to (==) and assignment operator (=) are two completely different operators.

Equal to (==) is an equality relational operator that evaluates two expressions to see if they are equal and returns true if they are equal and false if they are not.

The assignment operator (=) is used to assign a value to a variable. Hence, we can have a complex assignment operation inside the equality relational operator for evaluation.

**Q #12) What are the various Arithmetic Operators in C++?**

**Answer: C++ supports the following arithmetic operators:**

* + addition
* – subtraction
* \* multiplication
* / division
* % module

**Let’s demonstrate the various arithmetic operators with the following piece of code.**

**Example:**

|  |
| --- |
| #include <iostream.h>  int main ()  {                int a=5, b=3;  cout<<”a + b = “<<a+b;  cout<”\na – b =”<<a-b;  cout<<”\na \* b =”<<a\*b;  cout<<”\na / b =”<<a/b;  cout<<”\na % b =“<<a%b;    return 0;   } |

**Output**:

a + b = 8  
a – b =2  
a \* b =15  
a / b =2  
a % b=1

As shown above, all the other operations are straightforward and the same as actual arithmetic operations, except the modulo operator which is quite different. Modulo operator divides a and b and the result of the operation is the remainder of the division.

**Q #13) What are the various Compound Assignment Operators in C++?**

**Answer:** **Following are the Compound assignation operators in C++:**

**+=, -=, \*=, /=, %=, >>=, <<=, &=, ^=,|=**

Compound assignation operator is one of the most  important features of C++ language which allow us to change the value of a variable with one of the basic operators:

**Example:**

|  |
| --- |
| value += increase; is equivalent to value = value + increase;  if base\_salary is a variable of type int.                 int base\_salary = 1000;                 base\_salary += 1000; #base\_salary = base\_salary + 1000                 base\_salary \*= 5; #base\_salary = base\_salary \* 5; |

**Q #14) State the difference between Pre and Post Increment/Decrement Operations.**

**Answer:** C++ allows two operators i.e ++ (increment) and –(decrement), that allow you to add 1 to the existing value of a variable and subtract 1 from the variable respectively. These operators are in turn, called increment (++) and decrement (–).

**Example:**

a=5;  
a++;

The second statement, a++, will cause 1 to be added to the value of a. Thus a++ is equivalent to

a = a+1; or  
a += 1;

A unique feature of these operators is that we can prefix or suffix these operators with the variable. Hence, if a is a variable and we prefix the increment operator it will be

++a;

This is called Pre-increment. Similarly, we have pre-decrement as well.

If we prefix the variable a with an increment operator, we will have,

a++;

This is the post-increment. Likewise, we have post-decrement too.

The difference between the meaning of pre and post depends upon how the expression is evaluated and the result is stored.

In the case of the pre-increment/decrement operator, the increment/decrement operation is carried out first and then the result passed to an lvalue. Whereas for post-increment/decrement operations, the lvalue is evaluated first and then increment/decrement is performed accordingly.

**Example:**

a = 5; b=6;  
++a;       #a=6  
b–;         #b=6  
–a;         #a=5  
b++;      #6

I/O through Console

**Q #15) What are the Extraction and Insertion operators in C++? Explain with examples.**

**Answer:** In the iostream.h library of C++, **cin**, and **cout** are the two data streams that are used for input and output respectively. Cout is normally directed to the screen and cin is assigned to the keyboard.

**“cin” (extraction operator):** By using overloaded operator >> with cin stream, C++ handles the standard input.

|  |
| --- |
| int age;  cin>>age; |

As shown in the above example, an integer variable ‘age’ is declared and then it waits for cin (keyboard) to enter the data. “cin” processes the input only when the RETURN key is pressed.

**“cout” (insertion operator):** This is used in conjunction with the overloaded << operator. It directs the data that followed it into the cout stream.

**Example:**

|  |
| --- |
| cout<<”Hello, World!”;   cout<<123; |

Control Structures And Functions

Control Structures And Loops

**Q #16) What is the difference between while and do while loop? Explain with examples.**

**Answer:** The format of while loop in C++ is:

**While (expression)**  
**{statements;}**

The statement block under while is executed as long as the condition in the given expression is true.

**Example:**

|  |
| --- |
| #include <iostream.h>   int main()  {                 int n;                 cout<<”Enter the number : “;                 cin>>n;                while(n>0)                {                                cout<<” “<<n;                                --n;                }                cout<<”While loop complete”;   } |

In the above code, the loop will directly exit if n is 0. Thus in the while loop, the terminating condition is at the beginning of the loop and if it’s fulfilled, no iterations of the loop are executed.

Next, we consider the do-while loop.

**The general format of do-while is:**

**do {statement;} while(condition);**

**Example:**

|  |
| --- |
| #include<iostream.h>  int main()  {                 int n;                 cout<<”Enter the number : “;                 cin>>n;                 do {                            cout<<n<<”,”;                            --n;                       }while(n>0);                      cout<<”do-while complete”;  } |

In the above code, we can see that the statement inside the loop is executed at least once as the loop condition is at the end. These are the main differences between the while and do-while.

In case of the while loop, we can directly exit the loop at the beginning, if the condition is not met whereas in the do-while loop we execute the loop statements at least once.

Functions

**Q #17) What do you mean by ‘void’ return type?**

**Answer:**All functions should return a value as per the general syntax.

However, in case, if we don’t want a function to return any value, we use “**void**” to indicate that. This means that we use “**void**” to indicate that the function has no return value or it returns “**void**”.

**Example:**

|  |
| --- |
| void myfunc()  {                           Cout<<”Hello,This is my function!!”;  }  int main()  {  myfunc();  return 0;  } |

**Q #18) Explain Pass by Value and Pass by Reference.**

**Answer:** While passing parameters to the function using “Pass by Value”, we pass a copy of the parameters to the function.

Hence, whatever modifications are made to the parameters in the called function are not passed back to the calling function. Thus the variables in the calling function remain unchanged.

**Example:**

|  |
| --- |
| void printFunc(int a,int b,int c)  {                    a \*=2;                    b \*=2;                    c \*=2;  }    int main()    {                     int x = 1,y=3,z=4;                   printFunc(x,y,z);                   cout<<”x = “<<x<<”\ny = “<<y<<”\nz = “<<z;  } |

**Output:**

x=1  
y=3  
z=4

As seen above, although the parameters were changed in the called function, their values were not reflected in the calling function as they were passed by value.

However, if we want to get the changed values from the function back to the calling function, then we use the “Pass by Reference” technique.

**To demonstrate this we modify the above program as follows:**

|  |
| --- |
| void printFunc(int& a,int& b,int& c)  {                                a \*=2;                                b \*=2;                                c \*=2;  }    int main()  {                    int x = 1,y=3,z=4;                    printFunc(x,y,z);                     cout<<”x = “<<x<<”\ny = “<<y<<”\nz = “<<z;  } |

**Output:**  
x=2  
y=6  
z=8

As shown above, the modifications done to the parameters in the called functions are passed to the calling function when we use the “Pass by reference” technique. This is because using this technique we do not pass a copy of the parameters but we actually pass the variable’s reference itself.

**Q #19) What are Default Parameters? How are they evaluated in the C++ function?**

**Answer:**Default Parameter is a value that is assigned to each parameter while declaring a function.

This value is used if that parameter is left blank while calling to the function. To specify a default value for a particular parameter, we simply assign a value to the parameter in the function declaration.

If the value is not passed for this parameter during the function call, then the compiler uses the default value provided. If a value is specified, then this default value is stepped on and the passed value is used.

**Example:**

|  |
| --- |
| int multiply(int a, int b=2)  {                int r;                r = a \* b;                return r;  }    int main()   {                 Cout<<multiply(6);               Cout<<”\n”;               Cout<<multiply(2,3);   } |

**Output:**

12  
6

As shown in the above code, there are two calls to multiply function. In the first call, only one parameter is passed with a value. In this case, the second parameter is the default value provided. But in the second call, as both the parameter values are passed, the default value is overridden and the passed value is used.

**Q #20) What is an Inline function in C++?**

**Answer:**Inline function is a function that is compiled by the compiler as the point of calling the function and the code is substituted at that point. This makes compiling faster. This function is defined by prefixing the function prototype with the keyword “inline”.

Such functions are advantageous only when the code of the inline function is small and simple. Although a function is defined as Inline, it is completely compiler dependent to evaluate it as inline or not.

Advanced-Data Structure

Arrays

**Q #21) Why are arrays usually processed with for loop?**

**Answer:**Array uses the index to traverse each of its elements.

If A is an array then each of its elements is accessed as A[i]. Programmatically, all that is required for this to work is an iterative block with a loop variable i that serves as an index (counter) incrementing from 0 to A.length-1.

This is exactly what a loop does and this is the reason why we process arrays using for loops.

**Q #22) State the difference between delete and delete[].**

**Answer:** “delete[]” is used to release the memory allocated to an array which was allocated using new[]. “delete” is used to release one chunk of memory which was allocated using new.

**Q #23) What is wrong with this code?**

**T \*p = new T[10];**  
**delete p;**

**Answer:**The above code is syntactically correct and will compile fine.

The only problem is that it will just delete the first element of the array. Though the entire array is deleted, only the destructor of the first element will be called and the memory for the first element is released.

**Q #24) What’s the order in which the objects in an array are destructed?**

**Answer:**Objects in an array are destructed in the reverse order of construction: First constructed, last destructed.

**In the following Example,** the order for destructors will be a[9], a[8], …, a[1], a[0]:

|  |
| --- |
| voiduserCode()  {                   Car a[10];                    ...  } |

Pointers

**Q #25) What is wrong with this code?**

**T \*p = 0;**  
**delete p;**

**Answer:** In the above code, the pointer is a null pointer. Per the C++ 03 standard, it’s perfectly valid to call delete on a NULL pointer. The delete operator would take care of the NULL check internally.

**Q #26) What is a Reference Variable in C++?**

**Answer:**A reference variable is an alias name for the existing variable. This means that both the variable name and the reference variable point to the same memory location. Hence, whenever the variable is updated, the reference is updated too.

**Example:**

|  |
| --- |
| int a=10;   int& b = a; |

Here, b is the reference of a.

Storage Classes

**Q #27) What is a Storage Class? Mention the Storage Classes in C++.**

**Answer:** Storage class determines the life or scope of symbols such as variable or functions.

**C++ supports the following storage classes:**

* Auto
* Static
* Extern
* Register
* Mutable

**Q #28) Explain Mutable Storage class specifier.**

**Answer:** The variable of a constant class object’s member cannot be changed. However, by declaring the variables as “mutable”, we can change the values of these variables.

**Q #29) What is the keyword auto for?**

**Answer:** By default, every local variable of the function is automatic i.e. *auto*. In the below function both the variables ‘i’ and ‘j’ are automatic variables.

|  |
| --- |
| void f()   {   int i;   auto int j;   } |

**NOTE**: A global variable is not an automatic variable.

**Q #30) What is a Static Variable?**

**Answer:** A static variable is a local variable that retains its value across the function calls. Static variables are declared using the keyword “static”. Numeric variables which are static have the default value as zero.

**The following function will print 1 2 3 if called thrice.**

|  |
| --- |
| void f()  {  static int i;  ++i;  printf(“%d “,i);  } |

If a global variable is static, then its visibility is limited to the same source code.

**Q #31) What is the purpose of the Extern Storage Specifier?**

**Answer:** “Extern” specifier is used to resolve the scope of a global symbol.

|  |
| --- |
| #include <iostream >   using nam espace std;   main()   {  extern int i;   cout<<i<<endl;   }   int i=20; |

In the above code, “i” can be visible outside the file where it is defined.

**Q #32) Explain Register Storage Specifier.**

**Answer:** “Register” variable should be used whenever the variable is used. When a variable is declared with a “register” specifier, then the compiler gives CPU register for its storage to speed up the lookup of the variable.

**Q #33) When to use “const” reference arguments in a function?**

**Answer:** **Using “const” reference arguments in a function is beneficial in several ways:**

* “const” protects from programming errors that could alter data.
* As a result of using “const”, the function is able to process both const and non-const actual arguments, which is not possible when “const” is not used.
* Using a const reference will allow the function to generate and use a temporary variable in an appropriate manner.

Structure & User-Defined Data Types

**Q #34) What is a Class?**

**Answer:**Class is a user-defined data type in C++. It can be created to solve a particular kind of problem. After creation, the user is not required to know the details of the working of a class.

In general, class acts as a blueprint of a project and can include in various parameters and functions or actions operating on these parameters. These are called the members of the class.

**Q #35) Difference between Class and Structure.**

**Answer:**

**Structure:** In C language, the structure is used to bundle different types of data types together. The variables inside a structure are called the members of the structure. These members are by default public and can be accessed by using the structure name followed by a dot operator and then the member name.

**Class:** Class is a successor of the Structure. C++ extends the structure definition to include the functions that operate on its members. By default all the members inside the class are private.

Object-Oriented Programming With C++

Classes, Constructors, Destructors

**Q #36) What is Namespace?**

**Answer:**Namespace allows us to group a set of global classes, objects and/or functions under a specific name.

**The general form to use namespaces is:**

**namespace identifier { namespace-body }**

Where identifier is any valid identifier and the namespace-body is the set of classes, objects, and functions that are included within the namespace. Namespaces are especially useful in the case where there is a possibility for more than one object to have the same name, resulting in name clashes.

**Q #37) What is the use of ‘using’ declaration?**

**Answer:**Using Declaration is used to refer a name from the namespace without the scope resolution operator.

**Q #38) What is Name Mangling?**

**Answer:** C++ compiler encodes the parameter types with function/method into a unique name. This process is called name mangling. The inverse process is called as demangling.

**Example:**

**A::b(int, long)** const is mangled as **‘b\_\_C3Ail’**.

For a constructor, the method name is left out.

That is **A:: A(int, long)** const is mangled as **‘C3Ail’.**

**Q #39) What is the difference between an Object and a Class?**

**Answer:**Class is a blueprint of a project or problem to be solved and consists of variables and methods. These are called the members of the class. We cannot access methods or variables of the class on its own unless they are declared static.

In order to access the class members and put them to use, we should create an instance of a class which is called an Object. The class has an unlimited lifetime whereas an object has a limited lifespan only.

**Q #40) What are the various Access Specifiers in C++?**

**Answer:** **C++ supports the following access specifiers:**

* **Public:** Data members and functions are accessible outside the class.
* **Private:** Data members and functions are not accessible outside the class. The exception is the usage of a friend class.
* **Protected:** Data members and functions are accessible only to the derived classes.

**Example:**

Describe PRIVATE, PROTECTED and PUBLIC along with their differences and give examples.

|  |
| --- |
| class A{               int x; int y;               public int a;               protected bool flag;               public A() : x(0) , y(0) {} //default (no argument) constructor   };     main(){    A MyObj;    MyObj.x = 5; // Compiler will issue a ERROR as x is private    int x = MyObj.x; // Compiler will issue a compile ERROR MyObj.x is private     MyObj.a = 10; // no problem; a is public member   int col = MyObj.a; // no problem     MyObj.flag = true; // Compiler will issue a ERROR; protected values are read only   bool isFlag = MyObj.flag; // no problem |

**Q #41) What is a Constructor and how is it called?**

**Answer:**Constructor is a member function of the class having the same name as the class. It is mainly used for initializing the members of the class. By default constructors are public.

**There are two ways in which the constructors are called:**

1. **Implicitly:** Constructors are implicitly called by the compiler when an object of the class is created. This creates an object on a Stack.
2. **Explicit Calling:** When the object of a class is created using new, constructors are called explicitly. This usually creates an object on a Heap.

**Example:**

|  |
| --- |
| class A{   int x; int y;   public A() : x(0) , y(0) {} //default (no argument) constructor   };   main()   {                      A Myobj; // Implicit Constructor call. In order to allocate memory on stack,                                      //the default constructor is implicitly called.                    A \* pPoint = new A(); // Explicit Constructor call. In order to allocate                                                    //memory on HEAP we call the default constructor.   } |

**Q #42) What is a COPY CONSTRUCTOR and when is it called?**

**Answer:**A copy constructor is a constructor that accepts an object of the same class as its parameter and copies its data members to the object on the left part of the assignment. It is useful when we need to construct a new object of the same class.

**Example:**

|  |
| --- |
| class A{               int x; int y;               public int color;               public A() : x(0) , y(0) {} //default (no argument) constructor               public A( const A& ) ;  };  A::A( const A & p )  {                this->x = p.x;                this->y = p.y;                this->color = p.color;  }  main()  {              A Myobj;              Myobj.color = 345;              A Anotherobj = A( Myobj ); // now Anotherobj has color = 345  } |

**Q #43) What is a Default Constructor?**

**Answer:**Default constructor is a constructor that either has no arguments or if there are any, then all of them are default arguments.

**Example:**

|  |
| --- |
| class B {   public: B (int m = 0) : n (m) {} int n;   };   int main(int argc, char \*argv[])  {   B b; return 0;   } |

**Q #44) What is a Conversion Constructor?**

**Answer:** It is a constructor that accepts one argument of a different type. Conversion constructors are mainly used for converting from one type to another.

**Q #45) What is an Explicit Constructor?**

**Answer:** A conversion constructor is declared with the explicit keyword. The compiler does not use an explicit constructor to implement an implied conversion of types. Its purpose is reserved explicitly for construction.

**Q #46) What is the role of the Static keyword for a class member variable?**

**Answer:** The static member variable shares a common memory across all the objects created for the respective class. We need not refer to the static member variable using an object. However, it can be accessed using the class name itself.

**Q #47) Explain the Static Member Function.**

**Answer:** A static member function can access only the static member variable of the class. Same as the static member variables, a static member function can also be accessed using the class name.

**Q #48) What’s the order in which the local objects are destructed?**

**Answer: Consider following a piece of code:**

|  |
| --- |
| Class A{   ….   };   int main()   {   A a;   A b;   ...   } |

In the main function, we have two objects created one after the other. They are created in order, first a then b. But when these objects are deleted or if they go out of the scope, the destructor for each will be called in the reverse order in which they were constructed.

Hence, the destructor of b will be called first followed by a. Even if we have an array of objects, they will be destructed in the same way in the reverse order of their creation.

Overloading

**Q #49) Explain Function Overloading and Operator Overloading.**

**Answer:**C++ supports OOPs concept Polymorphism which means “many forms”.

In C++ we have two types of polymorphism, i.e. Compile-time polymorphism, and Run-time polymorphism. Compile-time polymorphism is achieved by using an Overloading technique. Overloading simply means giving additional meaning to an entity by keeping its base meaning intact.

**C++ supports two types of overloading:**

**Function Overloading:**

Function overloading is a technique that allows the programmer to have more than one function with the same name but different parameter list. In other words, we overload the function with different arguments i.e. be it the type of arguments, number of arguments or the order of arguments.

Function overloading is never achieved on its return type.

**Operator Overloading:**

This is yet another type of compile-time polymorphism that is supported by C++. In operator overloading, an operator is overloaded, so that it can operate on the user-defined types as well with the operands of the standard data type. But while doing this, the standard definition of that operator is kept intact.

**For Example,** an Addition operator (+) that operates on numerical data types can be overloaded to operate on two objects just like an object of complex number class.

**Q #50) What is the difference between Method Overloading and Method Overriding in C++?**

**Answer:**Method overloading is having functions with the same name but different argument lists. This is a form of compile-time polymorphism.

Method overriding comes into picture when we rewrite the method that is derived from a base class. Method overriding is used while dealing with run-time polymorphism or virtual functions.

**Q #51) What is the difference between a Copy Constructor and an Overloaded Assignment Operator?**

**Answer:**A copy constructor and an overloaded assignment operator basically serve the same purpose i.e. assigning the content of one object to another. But still, there is a difference between the two.

**Example:**

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

In the above example, the second statement c1 = c2 is an overloaded assignment statement.

Here, both c1 and c2 are already existing objects and the contents of c2 are assigned to the object c1. Hence, for overloaded assignment statement both the objects need to be created already.

Next statement, complex c3 = c2 is an example of the copy constructor. Here, the contents of c2 are assigned to a new object c3, which means the copy constructor creates a new object every time when it executes.

**Q #52) Name the Operators that cannot be Overloaded.**

**Answer:**

* sizeof – sizeof operator
* . – Dot operator
* .\* – dereferencing operator
* -> – member dereferencing operator
* :: – scope resolution operator
* ?: – conditional operator

**Q #53) Function can be overloaded based on the parameter which is a value or a reference. Explain if the statement is true.**

**Answer:** False. Both, Passing by value and Passing by reference look identical to the caller.

**Q #54) What are the benefits of Operator Overloading?**

**Answer:**By overloading standard operators on a class, we can extend the meaning of these operators, so that they can also operate on the other user-defined objects.

Function overloading allows us to reduce the complexity of the code and make it more clear and readable as we can have the same function names with different argument lists.

Inheritance

**Q #55) What is Inheritance?**

**Answer:** Inheritance is a process by which we can acquire the characteristics of an existing entity and form a new entity by adding more features to it.

In terms of C++, inheritance is creating a new class by deriving it from an existing class so that this new class has the properties of its parent class as well as its own.

**Q #56) What are the advantages of Inheritance?**

**Answer:**Inheritance allows code re-usability, thereby saving time on code development.

By inheriting, we make use of a bug-free high-quality software that reduces future problems.

**Q #57) Does C++ support Multilevel and Multiple Inheritances?**

**Answer:** Yes.

**Q #58) What are Multiple Inheritances (virtual inheritance)? What are its advantages and disadvantages?**

**Answer:**In multiple inheritances, we have more than one base classes from which a derived class can inherit. Hence, a derived class takes the features and properties of more than one base class.

**For Example**, a class **driver** will have two base classes namely, **employee**and a person because a driver is an employee as well as a person. This is advantageous because the driver class can inherit the properties of the employee as well as the person class.

But in the case of an employee and a person, the class will have some properties in common. However, an ambiguous situation will arise as the driver class will not know the classes from which the common properties should be inherited. This is the major disadvantage of multiple inheritances.

**Q #59) Explain the ISA and HASA class relationships. How would you implement each?**

**Answer:**“ISA” relationship usually exhibits inheritance as it implies that a class “ISA” specialized version of another class. **For Example**, An employee ISA person. That means an Employee class is inherited from the Person class.

Contrary to “ISA”, “HASA” relationship depicts that an entity may have another entity as its member or a class has another object embedded inside it.

So taking the same example of an Employee class, the way in which we associate the Salary class with the employee is not by inheriting it but by including or containing the Salary object inside the Employee class. “HASA” relationship is best exhibited by containment or aggregation.

**Q #60) Does a derived class inherit or doesn’t inherit?**

**Answer:** When a derived class is constructed from a particular base class, it basically inherits all the features and ordinary members of the base class. But there are some exceptions to this rule. For instance, a derived class does not inherit the base class’s constructors and destructors.

Each class has its own constructors and destructors. The derived class also does not inherit the assignment operator of the base class and friends of the class. The reason is that these entities are specific to a particular class and if another class is derived or if it is the friend of that class, then they cannot be passed onto them.

Polymorphism

**Q #61) What is Polymorphism?**

**Answer:** The basic idea behind polymorphism is in many forms. In C++, we have two types of Polymorphism:

**(i) Compile-time Polymorphism**

In compile-time polymorphism, we achieve many forms by overloading. Hence, we have an Operator overloading and function overloading. (We have already covered this above)

**(ii) Run-time Polymorphism**

This is the polymorphism for classes and objects. General idea is 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.

This means, that an object reacts differently to the same function call. This type of polymorphism can use a virtual function mechanism.

**Q #62) What are Virtual Functions?**

**Answer:** A virtual function allows the derived classes to replace the implementation provided by the base class.

Whenever we have functions with the same name in the base as well as derived class, there arises an ambiguity when we try to access the child class object using a base class pointer. As we are using a base class pointer, the function that is called is the base class function with the same name.

To correct this ambiguity we use the keyword “virtual” before the function prototype in the base class. In other words, we make this polymorphic function Virtual. By using a Virtual function, we can remove the ambiguity and we can access all the child class functions correctly using a base class pointer.

**Q #63) Give an example of Run-time Polymorphism/Virtual Functions.**

**Answer:**

|  |
| --- |
| class SHAPE{                    public virtual Draw() = 0; //abstract class with a pure virtual method   };   class CIRCLE: public SHAPE{                    public int r;                    public Draw() { this->drawCircle(0,0,r); }   };  class SQUARE: public SHAPE{                 public int a;                 public Draw() { this->drawSquare(0,0,a,a); }   };     int main()   {                  SHAPE shape1\*;                  SHAPE shape2\*;                    CIRCLE c1;                  SQUARE s1;                   shape1 = &c1;                shape2 = &s1;               cout<<shape1->Draw(0,0,2);               cout<<shape2->Draw(0,0,10,10);   } |

In the above code, the SHAPE class has a pure virtual function and is an abstract class (cannot be instantiated). Each class is derived from SHAPE implementing Draw () function in its own way.

Further, each Draw function is virtual so that when we use a base class (SHAPE) pointer each time with the object of the derived classes (Circle and SQUARE), then appropriate Draw functions are called.

**Q #64) What do you mean by Pure Virtual Functions?**

**Answer:** A Pure Virtual Member Function is a member function in which the base class forces the derived classes to override. Normally this member function has no implementation. Pure virtual functions are equated to zero.

**Example:**

|  |
| --- |
| class Shape { public: virtual void draw() = 0; }; |

Base class that has a pure virtual function as its member can be termed as an “Abstract class”. This class cannot be instantiated and it usually acts as a blueprint that has several sub-classes with further implementation.

**Q #65) What are Virtual Constructors/Destructors?**

**Answer:**

**Virtual Destructors:** When we use a base class pointer pointing to a derived class object and use it to destroy it, then instead of calling the derived class destructor, the base class destructor is called.

**Example:**

|  |
| --- |
| Class A{                ….                ~A();  };  Class B:publicA{                 …                 ~B();  };  B b;  A a = &b;  delete a; |

As shown in the above example, when we say delete a, the destructor is called but it’s actually the base class destructor. This gives rise to the ambiguity that all the memory held by b will not be cleared properly.

This problem can be solved by using the “Virtual Destructor” concept.

What we do is, we make the base class constructor “Virtual” so that all the child class destructors also become virtual and when we delete the object of the base class pointing to the object of the derived class, the appropriate destructor is called and all the objects are properly deleted.

**This is shown as follows:**

|  |
| --- |
| Class A{                ….                virtual ~A();  };  Class B:publicA{                 …                 ~B();  };  B b;  A a = &b;  delete a; |

**Virtual constructor**: Constructors cannot be virtual. Declaring a constructor as a virtual function is a syntax error.

Friend

**Q #66) What is a friend function?**

**Answer:** C++ class does not allow its private and protected members to be accessed outside the class. But this rule can be violated by making use of the “**Friend**” function.

As the name itself suggests, friend function is an external function that is a friend of the class. For friend function to access the private and protected methods of the class, we should have a prototype of the friend function with the keyword “friend” included inside the class.

**Q #67) What is a friend class?**

**Answer:**Friend classes are used when we need to override the rule for private and protected access specifiers so that two classes can work closely with each other.

Hence, we can have a friend class to be a friend of another class. This way, friend classes can keep private, inaccessible things in the way they are.

When we have a requirement to access the internal implementation of a class (private member) without exposing the details by making the public, we go for friend functions.

Advanced C++

Templates

**Q #68) What is a template?**

**Answer:** Templates allow creating functions that are independent of data type (generic) and can take any data type as parameters and return value without having to overload the function with all the possible data types. Templates nearly fulfill the functionality of a macro.

**Its prototype is any of the following ones:**

***template <class****identifier****>****function\_declaration;*

***template <typename****identifier****>****function\_declaration;*

The only difference between both the prototypes is the use of keyword class or typename. Their basic functionality of being generic remains the same.

Exception Handling

**Q #69) What is Exception Handling? Does C++ support Exception Handling?**

**Answer:** Yes C++ supports exception handling.

We cannot ensure that code will execute normally at all times. There can be certain situations that might force the code written by us to malfunction, even though it’s error-free. This malfunctioning of code is called **Exception**.

When an exception has occurred, the compiler has to throw it so that we know an exception has occurred. When an exception has been thrown, the compiler has to ensure that it is handled properly, so that the program flow continues or terminates properly. This is called the **handling of an exception.**

Thus in C++, we have three keywords i.e. **try**, **throw** and **catch**which are in exception handling.

**The general syntax for exception block is:**

|  |
| --- |
| try{   ….   # Code that is potentially about to throw exception goes here   ….   throw exception;   }   catch(exception type) {   …   #code to handle exception goes here   } |

As shown above, the code that might potentially malfunction is put under the try block. When code malfunctions, an exception is thrown. This exception is then caught under the catch block and is handled i.e. appropriate action is taken.

**Q #70) Comment on C++ standard exceptions?**

**Answer:** C++ supports some standard exceptions that can be caught if we put the code inside the try block. These exceptions are a part of the base class “**std:: exception”.** This class is defined in the C++ header file **<exception>**.

**Few Examples of Exceptions supported by this class include:**

*bad\_alloc – thrown by ‘new’*

*runtime\_error – thrown for runtime errors*

*bad\_typeid – thrown by type id*

Introduction to Standard Template Library

**Q #71) What is a Standard Template Library (STL)? What are the various types of STL Containers?**

**Answer:** A Standard Template Library (STL) is a library of container templates approved by the ANSI committee for inclusion in the standard C++ specification. We have various types of STL containers depending on how they store the elements.

1. **Queue, Stack** – These are the same as traditional queue and stack and are called adaptive containers.
2. **Set, Map** – These are basically containers that have key/value pairs and are associative in nature.
3. **Vector, deque** – These are sequential in nature and have similarities to arrays.

**Q #72) What is an Iterator class?**

**Answer:**In C++ a container class is a collection of different objects.

If we need to traverse through this collection of objects, we cannot do it using simple index variables. Hence, we have a special class in STL called an **Iterator** class which can be used to step through the contents of the container class.

**The various categories of iterators include** input iterators, output iterators, forward iterators, bidirectional iterators, random access, etc.

**Q #73) What is the difference between an External Iterator and an Internal Iterator? Describe an advantage of the External Iterator.**

**Answer:**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 bound to the object that has items to step through. The basic advantage of an External iterator is that it’s easy to implement as it is implemented as a separate class.

Secondly, as it’s a different class, many iterator objects can be active simultaneously.