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

The 4 data types in C++ are given below:

- Primitive Datatype(basic datatype). Example- char, short, int, float, long, double, bool, etc.
- Derived datatype. Example- array, pointer, etc.
- Enumeration. Example- enum
- User-defined data types. Example- structure, class, etc.

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

The main difference between C and C++ are provided in the table below:

C	C++
C is a procedure-oriented programming language.	C++ is an object-oriented programming language.
C does not support data hiding.	Data is hidden by encapsulation to ensure that data structures and operators are used as intended.
C is a subset of C++	C++ is a superset of C.
Function and operator overloading are not supported in C	Function and operator overloading is supported in C++
Namespace features are not present in C	Namespace is used by C++, which avoids name collisions.
Functions can not be defined inside structures.	Functions can be defined inside structures.
calloc() and malloc() functions are used for memory allocation and free() function is used for memory deallocation.	new operator is used for memory allocation and deletes operator is used for memory deallocation.

3. What are class and object in C++?

A class is a user-defined data type that has data members and member functions. Data members are the data variables and member functions are the functions that are used to perform operations on these variables.

An object is an instance of a class. Since a class is a user-defined data type so an object can also be called a variable of that data type.

A class is defined as-

```
class A{
private:
 int data;
public:
 void fun(){

 }
};
```

4. What is the difference between struct and class?

Class	Structure
1. Members of a class are private by default.	1. Members of a structure are public by default.
2. An instance of a class is called an 'object'.	2. An instance of structure is called the 'structure variable'.
3. Member classes/structures of a class are private by default but not all programming languages have this default behavior eg Java etc.	3. Member classes/structures of a structure are public by default.
4. It is declared using the **class** keyword.	4. It is declared using the **struct** keyword.
5. It is normally used for data abstraction and further inheritance.	5. It is normally used for the grouping of data
6. NULL values are possible in Class.	6. NULL values are not possible.

7. **Syntax:**
class class_name{
data_member;
member_function;
};
	7. **Syntax:**
	struct structure_name{
	type structure_member1;
	type structure_member2;
	};

5. What is operator overloading?

Operator Overloading is a very essential element to perform the operations on user-defined data types. By operator overloading we can modify the default meaning to the operators like +, -, *, /, <=, etc.

```
class complex{
private:
 float r, i;
public:
 complex(float r, float i){
  this->r=r;
  this->i=i;
 }
 complex(){}
 void displaydata(){
  cout<<"real part = "<<r<<endl;
  cout<<"imaginary part = "<<i<<endl;
 }
 complex operator+(complex c){
  return complex(r+c.r, i+c.i);
 }
};
int main(){
complex a(2,3);
complex b(3,4);
complex c=a+b;
c.displaydata();
return 0;
}
```

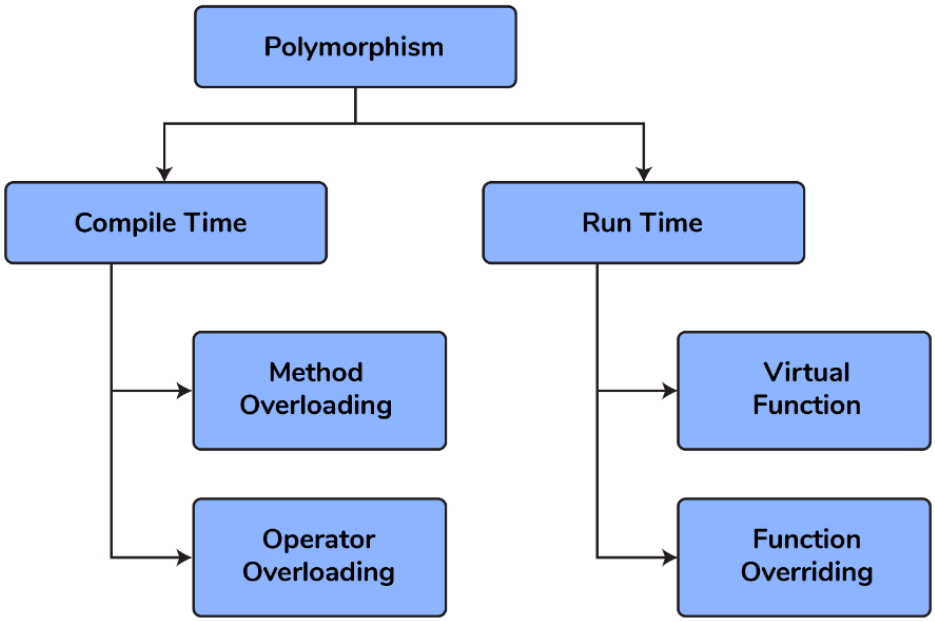
6. What is polymorphism in C++?

Polymorphism in simple means having many forms. Its behavior is different in different situations. And this occurs when we have multiple classes that are related to each other by inheritance.

For example, think of a base class called a car that has a method called car brand(). Derived classes of cars could be Mercedes, BMW, Audi - And they also have their own implementation of a cars

The two types of polymorphism in c++ are:

- Compile Time Polymorphism
- Runtime Polymorphism



7. Explain constructor in C++

The constructor is a member function that is executed automatically whenever an object is created. Constructors have the same name as the class of which they are members so that the compiler knows that the member function is a constructor. And no return type is used for constructors.

Example:

```
class A{
 private:
  int val;
 public:
  A(int x){          //one argument constructor
   val=x;
  }
  A(){               //zero argument constructor
  }
}
int main(){
 A a(3);

 return 0;
}
```

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

```cpp
#include <iostream>
using namespace std;

class base {
public:
    virtual void print() { cout << "print base class\n"; }

    void show() { cout << "show base class\n"; }
};

class derived : public base {
public:
    void print() { cout << "print derived class\n"; }

    void show() { cout << "show derived class\n"; }
};

int main()
{
    base* bptr;
    derived d;
    bptr = &d;

    // Virtual function, binded at runtime
    bptr->print();

    // Non-virtual function, binded at compile time
    bptr->show();

    return 0;
}
```

Output

```
print derived class
show base class
```

A virtual function (also known as virtual methods) is a member function that is declared within a base class and is re-defined (overridden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the method.

Explanation: Runtime polymorphism is achieved only through a pointer (or reference) of the base class type. Also, a base class pointer can point to the objects of the base class as well as to the objects of the derived class. In the above code, the base class pointer 'bptr' contains the address of object 'd' of the derived class.

Late binding (Runtime) is done in accordance with the content of the pointer (i.e. location pointed to by pointer) and **Early binding (Compile-time)** is done according to the type of pointer since the print() function is declared with the virtual keyword so it will be bound at runtime (output is *print derived class* as the pointer is pointing to object of derived class) and show() is **non-virtua**l so it will be bound **during compile time** (output is *show base class* as the pointer is of base type).

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 A{
   public:
      virtual void fun(){
         cout<<"base ";
      }
};
class B: public A{
   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.

```
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;
}
```

11. What are the C++ access specifiers?

Public: All data members and member functions are accessible outside the class.

Protected: All data members and member functions are accessible inside the class and to the derived class.

Private: All data members and member functions are not accessible outside the class.

12. Define inline function

If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time. One of the important advantages of using an inline function is that it eliminates the function calling overhead of a traditional function.

Syntax:

```
inline return-type function-name(parameters)
{
    // function code
}
```

13. What is a reference in C++?

A reference is like a pointer. It is another name of an already existing variable. Once a reference name is initialized with a variable, that variable can be accessed by the variable name or reference name both.

```
int x=10;
int &ref=x;          //reference variable
```

If we change the value of ref it will be reflected in x. Once a reference variable is initialized it cannot refer to any other variable. We can declare an array of pointers but an array of references is not possible.

14. What do you mean by abstraction in C++?

Abstraction is the process of showing the essential details to the user and hiding the details which we don't want to show to the user or hiding the details which are irrelevant to a particular user.

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 the call by value method, we pass a copy of the parameter 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 the 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.

```
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?

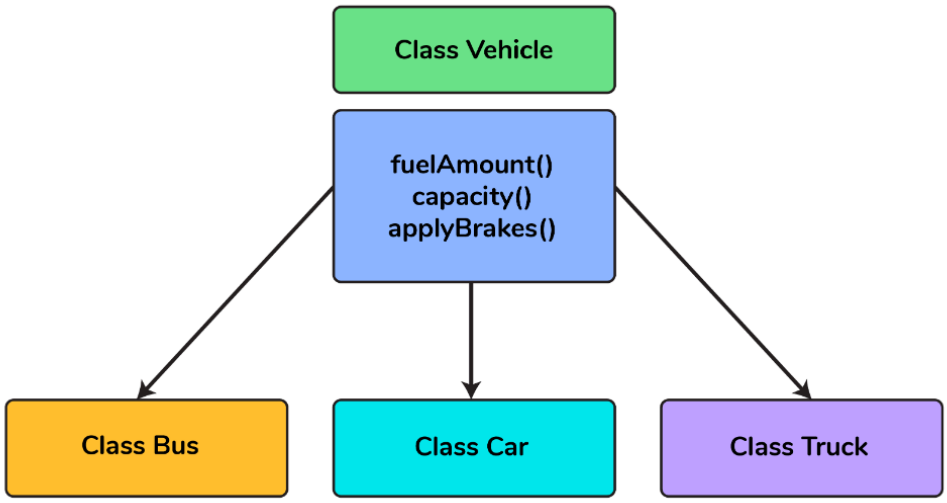
Static data members are class members that are declared using static keywords. A static member has certain special characteristics which are as follows: Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created. It is initialized before any object of this class is created, even before the main starts. It is visible only within the class, but its lifetime is the entire program.

Static Member Function in C++ A static member function is independent of any object of the class. A static member function can be called even if no objects of the class exist. A static member function can also be accessed using the class name through the scope resolution operator

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.



1. What is a copy constructor?

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

https://trainings.internshala.com/blog/copy-constructor-in-cpp/

What is a Copy Constructor in C++?

Creating exact copies of objects from the same class can be a time-consuming process. However, with a copy constructor, you can do this easily. By duplicating all data variables and respective values, you can quickly generate an identical version of whatever object needs replicating.

Additionally, copy constructors are also useful when clone objects need to be stored in different locations. Instead of creating an entirely new object from scratch with the same data variables and respective values, you can utilize a copy constructor to quickly replicate all necessary parameters without much effort. This technique is great for conserving time and computing power that would otherwise have been wasted on redundant tasks.

Example of Copy Constructor in C++

```
#include <iostream>
using namespace std;

// Declare a class
class Rectangle {
  private:
    double width;
    double length;

  public:
    // Initialize variables with a parameterized constructor
    Rectangle(double w, double l) {
      width = w;
      length = l;
    }

    // Copy constructor with a Rectangle object as a parameter
    // Copies data from the passed object
    Rectangle(const Rectangle &obj) {
      width = obj.width;
      length = obj.length;
    }

    double calculatePerimeter() {
      return 2 * (width + length);
    }
};
```

Characteristics of Copy Constructor

The following are the key characteristics of a Copy Constructor:

- It is used to initialize a new object's members by duplicating the members of an existing object. It accepts a reference to an object of the same class as a parameter.
- Copy initialization involves using the copy constructor, which carries out member-wise initialization; copying each individual member respectively between objects of identical classes.
- If not specified explicitly, this process will be handled automatically by compiler-generated code for creating such constructors within programs. These programs can be written in C++ language or similar languages that support Object Oriented Programming (OOP).
- The copy constructor is useful for creating copies of existing objects without having to redefine the object's entire data structure and attributes again.
- It also allows a programmer to provide custom modifications while copying an object, such as altering members according to certain criteria or making other changes. These changes can be more difficult in regular 'assignment' type operations between two identical classes of objects

```
int main() {
  // Create an object of the Rectangle class
  Rectangle rect1(4.0, 7.5);

  // Copy contents of rect1 to rect2 using the copy constructor
  Rectangle rect2(rect1);

  // Print perimeters of rect1 and rect2
  cout << "Perimeter of Rectangle 1: " << rect1.calculatePerimeter() << endl;
  cout << "Perimeter of Rectangle 2: " << rect2.calculatePerimeter();

  return 0;
}
```

The copy constructor of the Rectangle class is used to create a duplicate object of an existing one by copying its contents. The code for this operation is as follows:

```
Rectangle(const Rectangle &obj) {
  width = obj.width;
  length = obj.length;
}
```

The constructor of this object takes a parameter that is a reference to another Rectangle class instance. This means the values stored in obj's variables are then assigned to the new object, invoking the copy constructor and effectively copying all data from one object into another. In main(), two rectangle objects, rect1 and rect2 have been created and an example has been given where the contents of rect1 were copied over onto rect2 using this method.

```
// Copy contents of rect1 to rect2
Rectangle rect2(rect1);
```

In this instance, the rect2 object invokes its copy constructor by passing a reference to the rect1 object as an argument – namely &obj = &rect1. This is done for constructors to fulfill their purpose of initializing objects and executing default code when they are created.

Characteristics of Copy Constructor

The following are the key characteristics of a Copy Constructor:

- It is used to initialize a new object's members by duplicating the members of an existing object. It accepts a reference to an object of the same class as a parameter.
- Copy initialization involves using the copy constructor, which carries out member-wise initialization; copying each individual member respectively between objects of identical classes.
- If not specified explicitly, this process will be handled automatically by compiler-generated code for creating such constructors within programs. These programs can be written in C++ language or similar languages that support Object Oriented Programming (OOP).
- The copy constructor is useful for creating copies of existing objects without having to redefine the object's entire data structure and attributes again.
- It also allows a programmer to provide custom modifications while copying an object, such as altering members according to certain criteria or making other changes. These changes can be more difficult in regular 'assignment' type operations between two identical classes of objects

Types of Copy Constructors in C++

A copy constructor program in C++ program is categorized into the following two types.

1. Default Copy Constructor

A default copy constructor is an implicitly defined one that copies the bases and members of an object like how they would be initialized by a constructor. The order in which these elements are copied during this process follows the same logic of initialization used when creating new objects with constructors.

```
#include <iostream>
using namespace std;

class MyClass {
    int number;

public:
    void setNumber(int x) { number = x; }
    void display() { cout << endl << "Number=" << number; }
};

int main()
{
    MyClass instance1;
    instance1.setNumber(20);
    instance1.display();

    // Implicit Copy Constructor Calling
    MyClass instance2(instance1); // or instance2=instance1;
    instance2.display();
    return 0;
}

// Output:
// Number=20
// Number=20
```

2. User-Defined Copy Constructor

A user-defined copy constructor is necessary when an object holds pointers or references that cannot be shared, such as a file. In these cases, it's recommended to also create a destructor and assignment operator in addition to the copy constructor.

```
#include <iostream>
using namespace std;

class MyClass {
    int number;
public:
    void setNumber(int x) { number = x; }
    MyClass() {}  // default constructor with empty body

    MyClass(MyClass &other) { // copy constructor
        number = other.number;
    }
    void display() {
        cout << endl << "Number=" << number;
    }
};

int main() {
    MyClass instance1;
    instance1.setNumber(20);
    instance1.display();

    MyClass instance2(instance1); // or instance2=instance1; copy constructor called
    instance2.display();
    return 0;
}

// Output:
// Number=20
// Number=20
```

Difference Between Copy Constructor and Assignment Operator

Copy Constructor	Assignment Operator
It is an overloaded constructor.	It is an operator.
Initializes a new object with an already existing one.	Assigns the value of one object to another already existing one.
Has different memory locations for each object.	The same memory location is used, but different variables may point to it.
A default copy constructor is provided if not defined.	Bitwise copy is provided when the operator is not overloaded.
It creates a new object using an existing element.	Assigns an existing object to a new one.

3. 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.

4. 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 the reverse order is followed. That is, the 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.

5. 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.

```
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;
  }
}
```

6. 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;
```

7. 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;
}
```

8. 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
```

1. Which operator can not be overloaded in C++?

- [ ] +
- [x] :: ✓ Your Selection
- [ ] *
- [ ] ++

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
- [x] 2 ✗ Your Selection
- [x] 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;
}
```

- [x] Compile Time Error ✓
- [ ] 3
- [ ] 1
- [x] 2 ✗ Your Selection

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 ✓ Your Selection
- None of the above

```
#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 ✗ Your Selection
- B A
- Compile-time error ✓

6. What is the size of void in C++?

- 0 ✓ Your Selection
- 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 ✓ Your Selection
- Depend on the parameter
- None of the above

8. Memory used by an array is

- Contiguous ✓ Your Selection
- 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 ✓ Your Selection