**C++ IMP Interview Questions 🙋**

**1) Static member functions and variables.**

Static function - we can access it an only created class and we call it only class '::'(Scope Resolution Operator) Name of Function/Method. we can initial a static variable as value we can't change it again.

Ex-

#include<iostream>

#include<conio.h>

using namespace std;

class Demo{

public:

static void int x;

cout<<x<<endl;

};

int Demo::x=9; //initialize the static variable

int main(){

Demo::x;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**2) Constructor**

Constructor has no any return type

1.Default Constructor

2.Parameterized Constructor

3.Copy Constructor

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**3) Constructor Is private in class is private Why?**

Ans- We can created a constructor as private in class, to restrict create an object of that class.

Ex: class construct{

int x;

private construct(){

}

}

int main(){

construct c1; //we can't create object because constructor is private // interface

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**4) Copy Constructor in C++**

Ans- It is created by using Reference variable. It can execute when the one object is called That time one block copy of constructor is pass as argument in other constructor.

// copy Constructor using reference variable //

#include<iostream>

#include<conio.h>

using namespace std;

class Demo{

int x, y;

public:

Demo(Demo &a){ //copy constructor created by reference

a.x++;

a.y++;

cout<<a.x<<" "<<a.y<<endl;

}

Demo(int p, int q){

x=p; y=q;

cout<<x<<" "<<y<<endl;

}

};

int main(){

Demo d1(1,2),d2(d1); // pass the d1 Object using d2 object

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**5) Inheritance In C++**

1.Single-Level Inheritance

2.Multi-level inheritance

3.Multiple Inheritance

4.Hibrid Inheritance

5.Hierarchical Inheritance

Syntax – child\_ class : parent\_ class

Inheritance Total 5 Types

//1.Single Ex:

// class A{

// }

// class B:A{

// }

//2.Multilevel

// class A{

// }

// class B:A{

// }

// class C:B{

// }

//3.Multiple //One exception Ambiguity Confused the child when two parent is created

// class A{

// }

// class B{

// }

// class C:A,B{

// }

//4.Hierarchical

// class A{

// }

// class B:A{

// }

// class C:A{

// }

//5.Hybrid

// class A{

// }

// class B{

// }

// class C:A,B{

// }

// class D:C{

// }

// class E:D{

// }

//The property of child class display with help of parent class is called as Inheritance

#include<iostream>

#include<conio.h>

using namespace std;

class Samsung{

int keypad, mic, speaker, volume;

public:

void setValue(){

keypad=67;

mic=56;

speaker=90;

volume =23;

}

};

class Samsung2 : protected Samsung{

int cam, display;

public:

void setValue2(){

cam=678;

display=34;

cout<<cam<<" "<<display<<endl;

}

};

int main(){

Samsung2 s1;

s1.setValue2();

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**6) Constructor Using Inheritance**

Constructer is Executed when parent class and child class are created that time the child class object is created. The calling sequence is child to parent but the execution sequence is parent to child.

\*Destructor- Destructor executed is first call child constructor and then Parent constructor.

And Execute sequence is Child To parent.

Ex-

#include<iostream>

#include<conio.h>

using namespace std;

class Samsung1{

public:

Samsung1(){

cout<<"Parent constructer executed"<<endl;

}

~Samsung1(){

cout<<"Parent Destructor"<<endl;

}

};

class Samsung2 : public Samsung1{

public:

Samsung2 : Samsung1 //When we are created child constructor

{

cout<<"Child constructer executed";

}

~Samsung2(){

cout<<"Child Destructor"<<endl;

}

};

int main(){

Samsung2 s1;

}

o/p- Parent constructer executed

Child constructer executed

child destructor

Parent Destructor

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**7) Command Line Arguments**

In C++ we can see that the main function is mostly return type is int. and main function is no any parameter received. We can also pass the argument in main function using two argument first is number of command line arguments and second one is list of command line argument.

Syntax:

int main (int argc, char\* argv[]){........}

OR

int main(int argc, char \*\*argv){........}

Ex-

#include<iostream>

using namespace std;

int min(int argc, char\* argv[]){

cout<<"You Have entered "<<argc<<" arguments"<<endl;

for(int i=0;i<argc;i++){

cout<<argv[i]<<endl;

}

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**8) Access Specifier's**

1. Public

2. Private

3. Protected

1. Public - Is a type of access modifiers which is access anywhere. That means when we can declare a member or variable as public that is that variable is accessible anywhere of that program.

Ex-

#include<iostream>

using namespace std;

class Demo {

public:

int x;

void fun () {

cout<<x<<endl;

}

}

int main () {

Demo d1;

d1.x=89;

d1.fun ();

}

2. Private- It is type of access specifiers which access only within That class which created an variable as private. We can access it an using that class public member functions.

Ex-

#include<iostream>

using namespace std;

class Demo{

Private:

int x;

fun(int y){

x = y;

cout<<x;

}

int main(){

Demo d1;

d1.fun();

}

3. Protected- It is type of access specifier which is same as private. It is not access any where but it can access by using the Derived(child) class of the Parent class. When the child class created using inheritance that time we can used the protected variables or member functions.

EX-

#include<iostream>

using namespace std;

class Parent{

Protected:

int x;

}

class Child : Parent{

fun(int y){

x = y;

cout<<x;

}

int main(){

Child c1;

c1.fun();

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**9) Function Overloading And Overriding**

Function Overloading Is possible when one function is overloaded that mince the one function name is Demo and second Function is Demo it's name is same but one no any argument is received and second one is one argument received that time the function is overload.

Ex-

#include<iostream>

using namespace std;

class Overload{

int a;

Public:

void Demo(){

cout<<"First Function"<<endl;

}

void Demo(int x){

a = x;

cout<<Second Function: "<<a<<endl;

}

};

int main(){

Overload a;

a.Demo();

a.Demo(2);

}

Function Overriding - Is possible only the different classes (Scope) when the function name is same and that functions work as well same that time the function is called overriding function

Ex-

#include<iostream>

using namespace std;

class Overriding {

Public:

void Demo () {

cout<<"First Function"<<endl;

}

class Overiding

cout<<Second Function: "<<a<<endl;

}

};

int main(){

Overload a;

a.Demo();

a.Demo(2);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**10) Abstract Class or Interface**

It is an concept that is when we are created any function in class is pure virtual function that time that class was declared as Abstract class. We cant instantiate(Can't create object) of that class.

Ex-

#include<iostream>

using namespace std;

class Abstract{

Public:

virtual void Demo()=0;

fun(){

cout<<"Hello";

}

};

class B : Abstract{

public:

void Demo(){

}

int main(){

B b1;

b1.fun();

}

O/P- Hello

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*VERY IMP

In the Abstract class is Constructor. We can created an constructer in abstract class to initialize the value in private members in the abstract class.

Ex-

#include<iostream>

using namespace std;

class Bank{

int x, y; //private

public:

virtual void ROI()=0;

Bank(){

x = 8;

y = 9;

cout<<"Parent Constructor"<<" x="<<x<<" y="<<y<<endl;

}

};

class SavingAccount : Bank{

public:

void ROI(){

}

SavingAccount(){

cout<<"Child Constructor: "<<endl;

}

};

int main(){

SavingAccount s1;

}

O/P-

Parent Constructor 8 9

Child Constructor:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**11) This Keyword And Super Keyword**

'this' keyword is mainly known as the when the this keyword is used in the any variable that is the variable is known as the that class. When we can use the in C++ :: scope resolution operator [java 'super' keyword] that time the variable is known as the parent class variable.

Ex-

#include<iostream>

using namespace std;

class Parent{

public:

int x,y;

void fun(){

cout<<"Parent x= "<<x<<" Parent y= "<<y<<endl;

}

};

class Child :public Parent{

int x, y;

public:

public:

void fun1(int x, int y){

this->x = x; // Child x

Parent::y = y; //Parent y

}

void Display(){

cout<<"Child x= "<<x<<" Child y= "<<y<<endl;

}

};

int main(){

Child c1;

c1.fun1(2,3);

c1.Display();

c1.fun();

// Parent p1;

// p1.fun();

}

O/P-Child x= 2 Child y= 4201035

Parent x= 1997963885 Parent y= 3

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**12) Template in C++ (STL)**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*