INDEX :

Inheritance :

```cpp
#include<iostream>

using namespace std;

// Base class

class Employee{

    public:

    int id;

    float salary;

    Employee(int inpId){

        id = inpId;

        salary = 4000;

    }


    Employee(){

        id = 101;

    }

};


// Derived class syntax

/*

class {{Derived-class-name}} : {{visibility mode}} {{base-class-name}}{

    class members/methods/etc...

}

Note:-

1. Default visibility mode is private.

2. Public Visibility mode: Public members of the base class becomes public members

   of the derive class.

3. Private Visibility mode: Public members of the base class becomes private members

   of the derive class.

4. Private members are never inherited.

*/


class Programmer : Employee{  // Here class is being derived privatelly as default.

    public:

    Programmer(int inpid){
```

```
        id = inpid;
    }
    int languageCode = 9;
    void getData(){
        cout<<id<<endl;
    }
};

int main(){
    Employee harry, rohan(102);
    cout<<harry.id<<endl;
    cout<<rohan.id<<endl;
    cout<<rohan.salary<<endl;

    Programmer skillF(1);
    // cout<<skillF.id<<endl; // --> This will give error as 'id' act as private member.
    skillF.getData();
    cout<<skillF.languageCode<<endl;

return 0;
}
```

Output :-

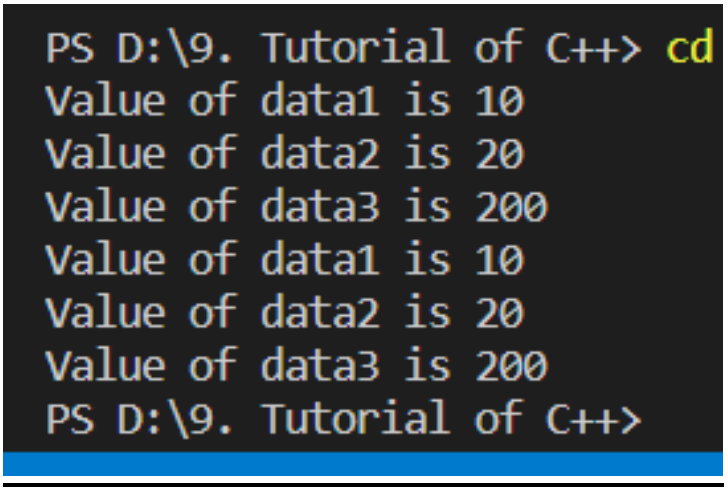
```
PS D:\9. Tutorial of C++> cd "d:
101
102
4000
1
9
PS D:\9. Tutorial of C++>
```

Single Inheritance :

```cpp
#include<iostream>

using namespace std;


class Base{

    int data1; // Private by default and is not inheritable.


    public:

    int data2;

    void setData();

    int getData1();


};


void Base :: setData(){

    data1 = 10;

    data2 = 20;

}


int Base :: getData1(){

    return data1;

}


class DerivedPublic : public Base{  // Here class is being derived publically.

    int data3;

    public:

    void process();

    void display();

};


void DerivedPublic :: process(){

    data3 = data2 * getData1();

}


void DerivedPublic :: display(){
```

```
    cout<<"Value of data1 is "<<getData1()<<endl;
    cout<<"Value of data2 is "<<data2<<endl;
    cout<<"Value of data3 is "<<data3<<endl;
}

class DerivedPrivate : private Base{  // Here class is being derived privatelly.
    int data3;
    public:
    void process();
    void display();
};

void DerivedPrivate :: process(){
    setData();
    data3 = data2 * getData1();
}

void DerivedPrivate :: display(){
    cout<<"Value of data1 is "<<getData1()<<endl;
    cout<<"Value of data2 is "<<data2<<endl;
    cout<<"Value of data3 is "<<data3<<endl;
}

int main(){
    DerivedPublic Me;
    Me.setData();
    Me.process();
    Me.display();

    DerivedPrivate Metoo;
    // Metoo.setData();  --> This will give error as this is privatelly inherited.
    Metoo.process();
    Metoo.display();
return 0;
}
```

Output :-


PS D:\9. Tutorial of C++> cd
Value of data1 is 10
Value of data2 is 20
Value of data3 is 200
Value of data1 is 10
Value of data2 is 20
Value of data3 is 200
PS D:\9. Tutorial of C++>

Protected Access Modifier :

```
#include<iostream>

using namespace std;


class Base{

    protected:

    int a;

    private:

    int b;

};


/*

                      Public Derivation     Private Derivation     Protected Derivation

1. Private members    Not inherited         Not inherited         Not inherited

2. Protected members  Protected             Private               Protected

3. Public members     Public                Private               Protected

*/


class Derived : protected Base{

};


int main(){

    Base b;

    Derived d;

//cout<<b.a; -->This will not work since a is protected in both Base as well as Derived Class.


return 0;

}
```

Multilevel Inheritance :

```
#include<iostream>

using namespace std;


class Student{

    protected:

    int rollNo;

    public:

    void setRollNo(int);

    void getRollNo();

};


void Student :: setRollNo(int r){

    rollNo = r;

}


void Student :: getRollNo(){

    cout<<"Roll number of student is : "<<rollNo<<endl;

}


class Exam : public Student{

    protected:

    float maths, physics;

    public:

    void setMarks(float, float);

    void getMarks();

};


void Exam :: setMarks(float m, float p){

    maths = m;

    physics = p;

}


void Exam :: getMarks(){

    cout<<"The marks obtained in Maths are : "<<maths<<endl;
```

```
    cout<<"The marks obtained in Physics are : "<<physics<<endl;
}


class Result : public Exam{
    public:
    void display(){
        getRollNo();
        getMarks();
        cout<<"Your result is : "<<(maths + physics)/2<<"%"<<endl;
    }
};


int main(){
    /*
    Notes:- If we are inheriting B from A and C from B : [A --> B --> C]
    1. 'A' is the base class for 'B' and 'B' is the base class for 'C'.
    2. A --> B --> C is called Inheritance Path.
    */
    Result Aman;
    Aman.setRollNo(21);
    Aman.setMarks(94.0, 96.0);
    Aman.display();

return 0;
}
```

Output :-

```
PS D:\9. Tutorial of C++> cd "d:\9. Tutorial
Roll number of student is : 21
The marks obtained in Maths are : 94
The marks obtained in Physics are : 96
Your result is : 95%
PS D:\9. Tutorial of C++>
```

Multiple Inheritance :

```
#include<iostream>

using namespace std;


class Base1{

    protected:

    int base1int;

    public:

    void setBase1int(int b){

        base1int = b;

    }

};


class Base2{

    protected:

    int base2int;

    public:

    void setBase2int(int b){

        base2int = b;

    }

};


class Base3{

    protected:

    int base3int;

    public:

    void setBase3int(int b){

        base3int = b;

    }

};


class Derived : public Base1, public Base2, public Base3{

    public:

    void show(){

        cout<<"The value of Base1 is "<<base1int<<endl;
```

```
        cout<<"The value of Base2 is "<<base2int<<endl;
        cout<<"The value of Base3 is "<<base3int<<endl;
        cout<<"The sum of these value is "<<(base1int + base2int + base3int)<<endl;
    }
};

int main(){
    Derived Aman;
    Aman.setBase1int(02);
    Aman.setBase2int(11);
    Aman.setBase3int(2004);
    Aman.show();

return 0;
}
```

Output :-

```
PS D:\9. Tutorial of C++> cd "d:\9.
The value of Base1 is 2
The value of Base2 is 11
The value of Base3 is 2004
The sum of these value is 2017
PS D:\9. Tutorial of C++>
```

Ambiguity Resolution :

```cpp
#include<iostream>

using namespace std;


class Base1{

    public:

    void greet(){

        cout<<"How are you ? "<<endl;

    }

};


class Base2{

    public:

    void greet(){

        cout<<"Kaise ho ? "<<endl;

    }

};


class Derived : public Base1, public Base2{

    public:

    void greet(){

        Base2 :: greet();

    }


};


class B{

    public:

    void say(){

        cout<<"Hey friend..!"<<endl;

    }

};


class D : public B{

    // D's new say() method will override base class's say() method.
```

```
    public:
    void say(){
        cout<<"Hello my beautiful friends..!"<<endl;
    }
};


int main(){
    Base1 base1obj;
    Base2 base2obj;
    base1obj.greet();
    base2obj.greet();

    Derived obj;
    obj.greet();

    B b;
    b.say();

    D d;
    d.say();

return 0;
}
```

Output :-

```
PS D:\9. Tutorial of C++> cd "d:\9.
How are you ?
Kaise ho ?
Kaise ho ?
Hey friend..!
Hello my beautiful friends..!
PS D:\9. Tutorial of C++>
```

Virtual Base Class :

```
#include<iostream>

using namespace std;


/*

Student --> Test

Student --> Sports

Test --> Result

Sports --> Result

*/


class Student{

    protected:

    int rollNo;

    public:

    void setNumber(int n){

        rollNo = n;

    }

    void printRollNo(){

        cout<<"Your roll number is : "<<rollNo<<endl;

    }

};


class Test : virtual public Student{

    protected:

    float maths, physics;

    public:

    void setMarks(float m, float p){

        maths = m;

        physics = p;

    }

    void printMarks(){

        cout<<"Your result is here : "<<endl;

        cout<<"Marks of Maths is : "<<maths<<endl;

        cout<<"Marks of physics is : "<<physics<<endl;
```

```
    }
};

class Sports : virtual public Student{
    protected:
    float score;
    public:
    void setScore(float s){
        score = s;
    }
    void getScore(){
        cout<<"Your PT score is : "<<score<<endl;
    }
};

class Result : public Test, public Sports{
    private:
    float total;
    public:
    void display(){
        total = maths + physics + score;
        printRollNo();
        printMarks();
        getScore();
        cout<<"Your total result is : "<<total<<endl;
    }
};

int main(){
    Result Aman;
    Aman.setNumber(21);
    Aman.setMarks(95, 97);
    Aman.setScore(85);
    Aman.display();
```

```
return 0;

}
```

Output :-

```
PS D:\9. Tutorial of C++> cd "d:\9. Tutorial
Your roll number is : 21
Your result is here :
Marks of Maths is : 95
Marks of physics is : 97
Your PT score is : 85
Your total result is : 277
PS D:\9. Tutorial of C++>
```

Constructor In Derived Class :

```
#include<iostream>

using namespace std;


/*

Case1:

class B : public A{

    // Order of the execution of constructor --> first A() then B()

};


Case2:

class A : public B, public C{

    // Order of the execution of constructor --> first B() then C() and then A()

    // Constructor of B is called first because it written for first.

};


Case3:

class A : public C, public B{

    // Order of the execution of constructor --> first C() then B() and then A()

    // Constructor of C is called first because it written first.

};


Case4:

Class A : public B, virtual public C{

    // Order of the execution of constructor --> first C() then B() and then A()

    Constructor of B is called first as virtual is more prefrable.

};

*/

class Base1{

    int data1;

    public:

    Base1(int d){

        data1 = d;

        cout<<"Base1 class constructor is called\n";

    }
```

```
    void Base1printData(){
        cout<<"The value of the data is : "<<data1<<endl;
    }
};

class Base2{
    int data2;
    public:
    Base2(int d){
        data2 = d;
        cout<<"Base2 class constructor is called\n";
    }
    void Base2printData(){
        cout<<"The value of the data is : "<<data2<<endl;
    }
};

class Derived1 : public Base1, public Base2{
    int Ddata1, Ddata2;
    public:
    Derived1(int a, int b, int c, int d) : Base1(a), Base2(b){
        Ddata1 = c;
        Ddata2 = d;
        cout<<"Derived class constructor is called\n";
    }
    void printDdata1(){
        cout<<"The value of the Ddata1 is : "<<Ddata1<<endl;
        cout<<"The value of the Ddata2 is : "<<Ddata2<<endl;
    }
};

class Derived2 : public Base2, public Base1{
    int Ddata11, Ddata22;
    public:
    Derived2(int a, int b, int c, int d) : Base1(a), Base2(b){
```

```
        Ddata11 = c;
        Ddata22 = d;
        cout<<"Derived class constructor is called\n";
    }
    void printDdata2(){
        cout<<"The value of the Ddata1 is : "<<Ddata11<<endl;
        cout<<"The value of the Ddata2 is : "<<Ddata22<<endl;
    }
};

int main(){
    Derived1 Aman(1, 2, 3, 4);
    Aman.Base1printData();
    Aman.Base2printData();
    Aman.printDdata1();

    Derived2 Nikku(1, 2, 3, 4);
    Nikku.Base1printData();
    Nikku.Base2printData();
    Nikku.printDdata2();

return 0;
}
```

Output :-

```
PS D:\9. Tutorial of C++> cd "d:\9. Tutorial
Class }
The value of the data is : 2
The value of the Ddata1 is : 3
The value of the Ddata2 is : 4
Base2 class constructor is called
Base1 class constructor is called
Derived class constructor is called
The value of the data is : 1
The value of the data is : 2
The value of the Ddata1 is : 3
The value of the Ddata2 is : 4
PS D:\9. Tutorial of C++>
```

Initialization List In Constructor :

```cpp
#include <iostream>

using namespace std;


class Test{

    int a;

    int n;


public:

    Test(int v, int r) : a(v), n(r)

    // Test(int v, int r) : a(v), n(v + r)

    // Test(int v, int r) : a(v), n(3 * r)

    // Test(int v, int r) : a(v), n(a + r)-->This will also run as value of 'a' is available.

    // Test(int v, int r) : n(r), a(v + n)

    {

        cout << "Constructor executed\n";

        cout << "The value of 'a' is : " << a << endl;

        cout << "The value of 'n' is : " << n << endl;

    }

};


int main(){

    Test Me(12, 21);


    return 0;

}
```

Output :-

```
PS D:\9. Tutorial of C++> cd "d:\9.
ns }
Constructor executed
The value of 'a' is : 12
The value of 'n' is : 21
PS D:\9. Tutorial of C++>
```