Day 08

25 March 2025 08:30 AM

Friend function

Dynamic :→ Run Time

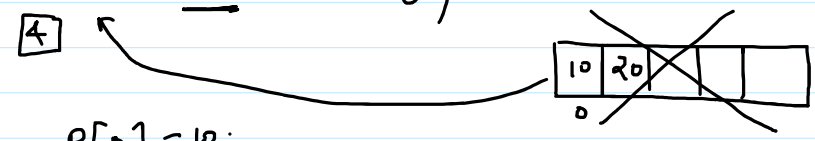
new

delete

Base Add = new data-type [size]

n | 5 int *p;

p = new int [n];



p[0] = 10;

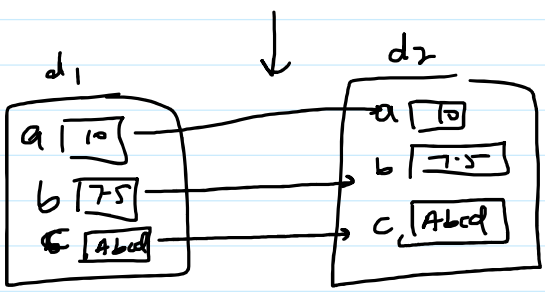
p[1] = 20;

delete []address

delete [] p;

shallow copy :→

shallow copy



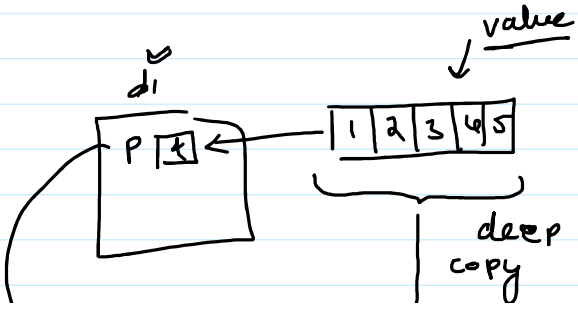
✔ data d1 (-----);

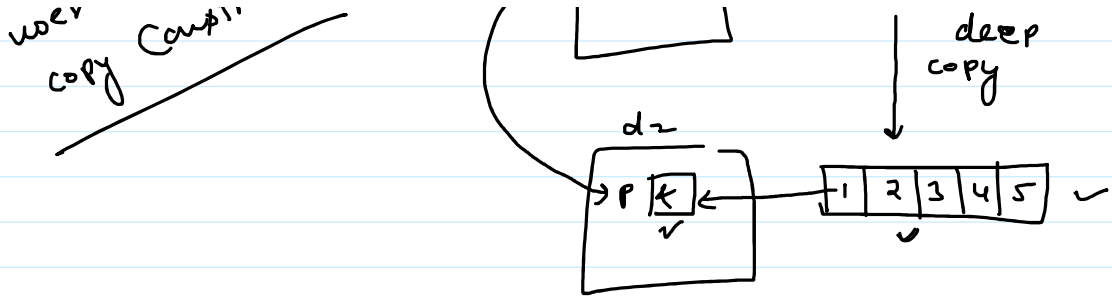
data d2 = d1;

copy

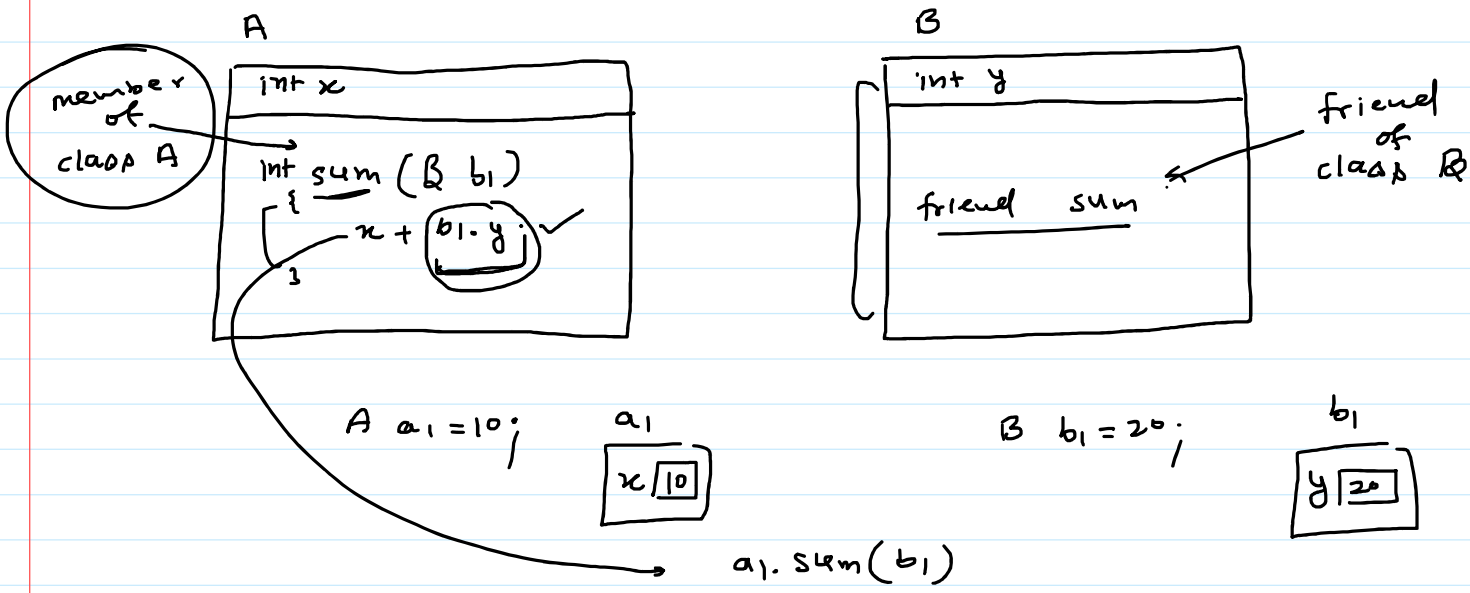
deep Copy :→

user defined copy constructor





friend function :→



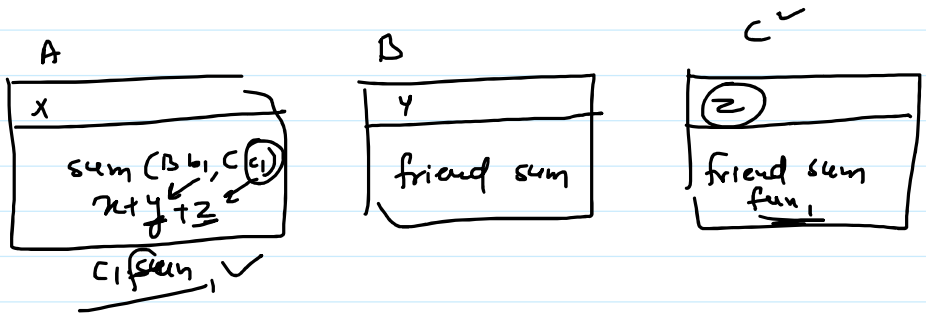

```
#include<iostream>
using namespace std;
class B;
class A{
    int x;
    public:
        A(int x1=0)
        {
            x=x1;
        }
        int sum(B);     //member function declaration
};
class B{
    int y;
    public:
        B(int y1=0)
        {
            y=y1;
        }
        friend int A::sum(B);       //friend function
declaration
};
int A::sum(B b1)
{
    return x+b1.y;
}
int main()
{
```

```
    A a1=10;
    B b1=20;
    cout<<a1.sum(b1);
    return 0;
}
```

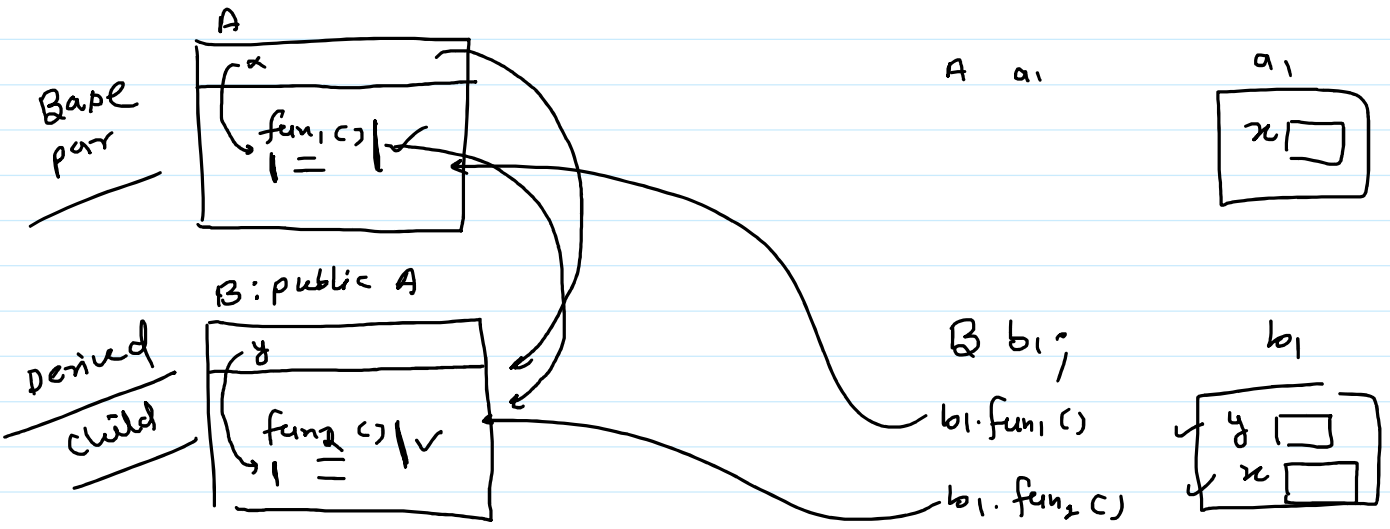
```
#include<iostream>
using namespace std;
class B;
class A{
    int x;
    public:
        A(int x1=0)
        {
            x=x1;
        }
        friend int sum(A,B);     //friend function declaration
};
class B{
    int y;
    public:
        B(int y1=0)
        {
            y=y1;
        }
        friend int sum(A,B);       //friend function declaration
};
int sum(A a1,B b1)
{
    return a1.x+b1.y;
}
int main()
{
    A a1=10;
    B b1=20;
    cout<<sum(a1,b1);
    return 0;
}
```

```
#include<iostream>
using namespace std;
class B;
class A{
    int x;
    public:
        A(int x1=0)
        {
            x=x1;
        }
        int sum(B);      //member function declaration
        int mult(B);     //member function declaration
};
class B{
```

```
    int y;
    public:
        B(int y1=0)
        {
            y=y1;
        }
        friend A;
};
int A::sum(B b1)
{
    return x+b1.y;
}
int A::mult(B b1)
{
    return x*b1.y;
}
int main()
{
    A a1=10;
    B b1=20;
    cout<<a1.sum(b1)<<endl;
    cout<<a1.mult(b1)<<endl;
    return 0;
}
```



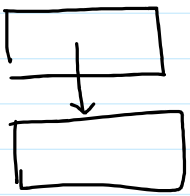
Inheritance :→



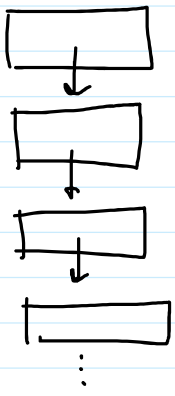
```
#include<iostream>
using namespace std;
class A{
    int x;
    public:
        void fun1()
        {
            x=10;
            cout<<"Fun1 of class A\n";
        }
};
class B:public A{
    int y;
    public:
        void fun2()
        {
            y=20;
            cout<<"Fun2 of class B\n";
        }
};
int main()
{
    A a1;
    B b1;
    a1.fun1();
    // a1.fun2();
    b1.fun2();
    b1.fun1();
    return 0;
}
```

Types of Inheritance :→

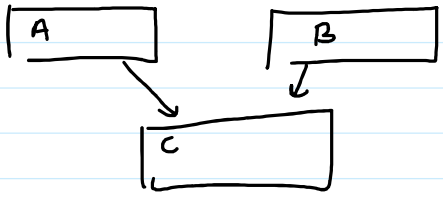
① Single Level



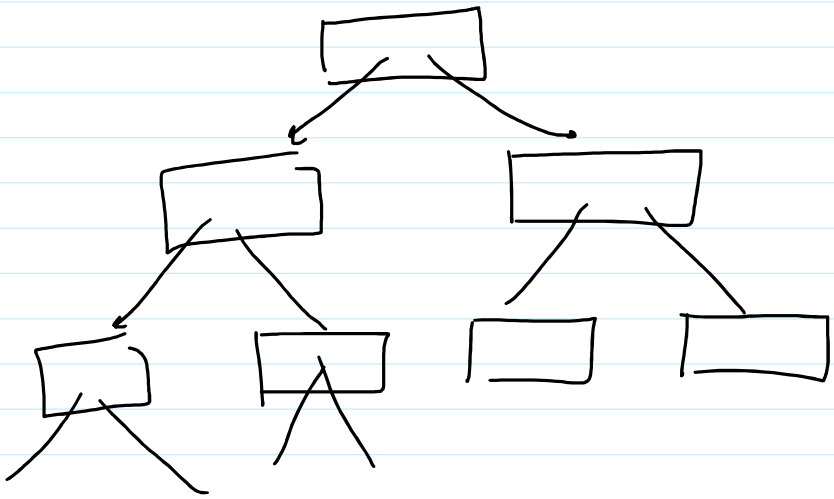
② multi Level



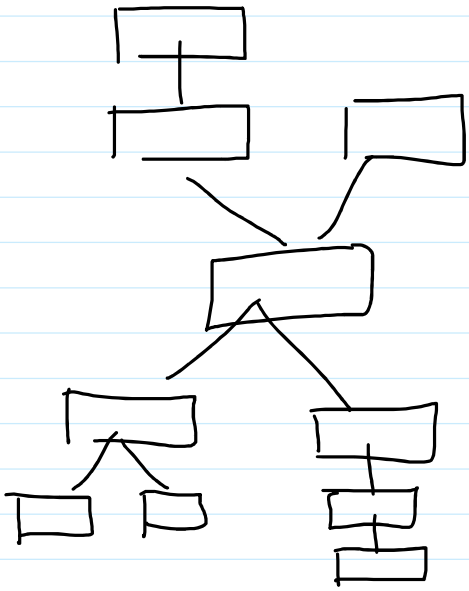
③ multiple Inheritance

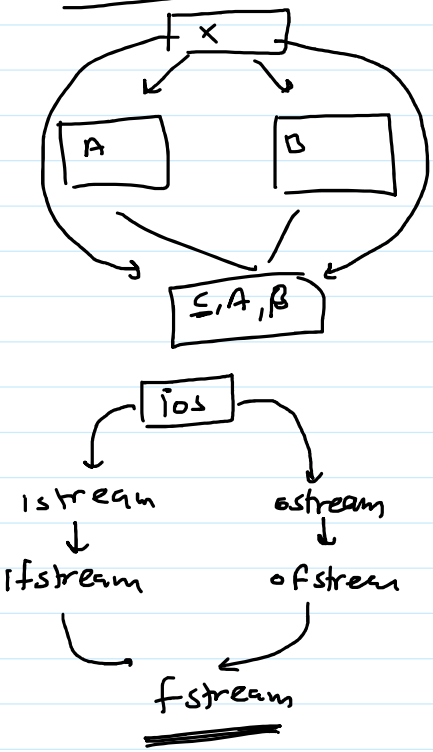


④ Hierarchical Inheritance (tree)



⑤ Hybrid Inheritance ⇒ multi path

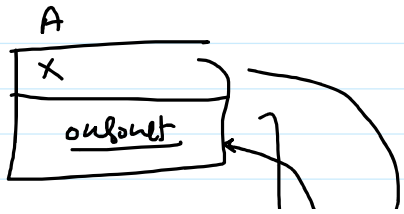


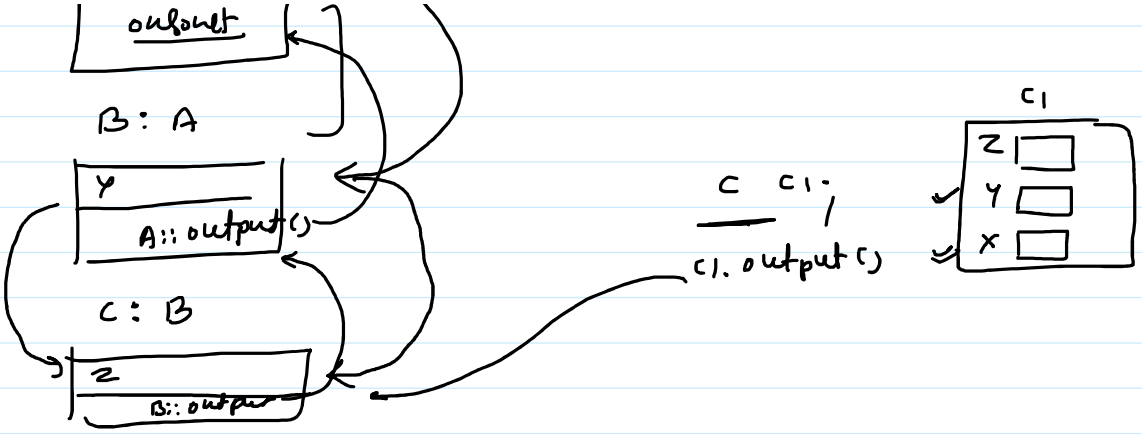


① Single level :-

```
#include<iostream>
using namespace std;
class A{
    int x;
    public:
        A()
        {
            cout<<"Class A constructor\n";
        }
        A(int x1)
        {
            x=x1;
            cout<<"Class A parameterized constructor
\n";
        }
        void output()
        {
            cout<<"Class A\n";
        }
};
class B:public A{
    int y;
    public:
        B():A()
        {
            cout<<"Class B constructor\n";
        }
        B(int x1, int y1):A(x1)
        {
            y=y1;
            cout<<"Class B parameterized constructor
\n";
        }
        void output()
        {
            A::output();
            cout<<"Class B\n";
        }
};
int main()
{
    B b1;
    B b2(10,20);
    b1.output();
    // b1.A::output();
    return 0;
}
```

multi level :-

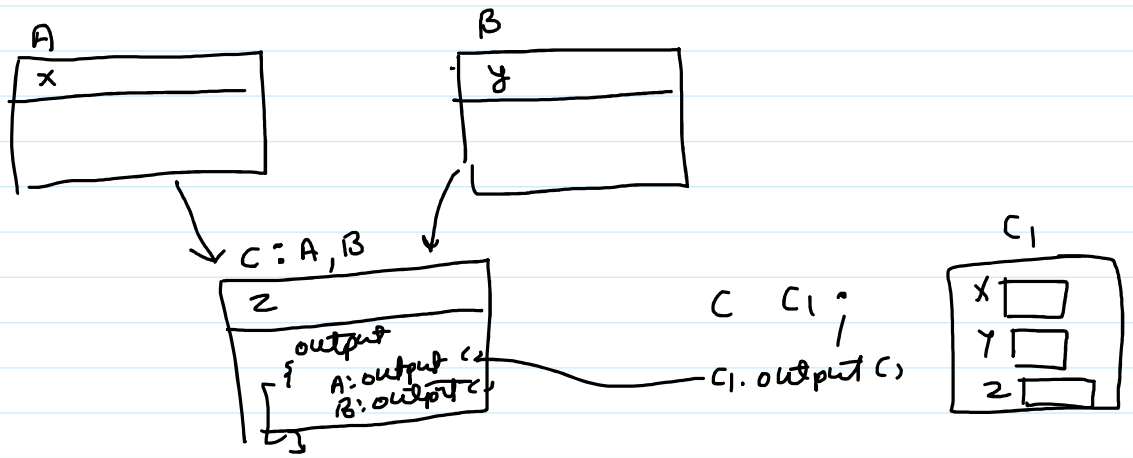





```
#include<iostream>
using namespace std;
class A{
    int x;
    public:
        A()
        {
            x=0;
            cout<<"Class A constructor\n";
        }
        A(int x1)
        {
            x=x1;
            cout<<"Class A parameterized constructor\n";
        }
        void output()
        {
            cout<<"X = "<<x<<endl;
        }
};
class B:public A{
    int y;
    public:
        B():A()
        {
            y=0;
            cout<<"Class B constructor\n";
        }
        B(int x1, int y1):A(x1)
        {
            y=y1;
            cout<<"Class B parameterized constructor\n";
        }
        void output()
        {
            A::output();
            cout<<"Y = "<<y<<endl;
        }
};
class C:public B{
    int z;
    public:
        C():B()
        {
            z=0;
            cout<<"Class C constructor\n";
```

```
        }
        C(int x1, int y1, int z1):B(x1,y1)
        {
            z=z1;
            cout<<"Class C parameterized constructor\n";
        }
        void output()
        {
            B::output();
            cout<<"Z = "<<z<<endl;
        }
};
int main()
{
    C c1;
    C c2(10,20,30);
    c1.output();
    c2.output();
    return 0;
}
```

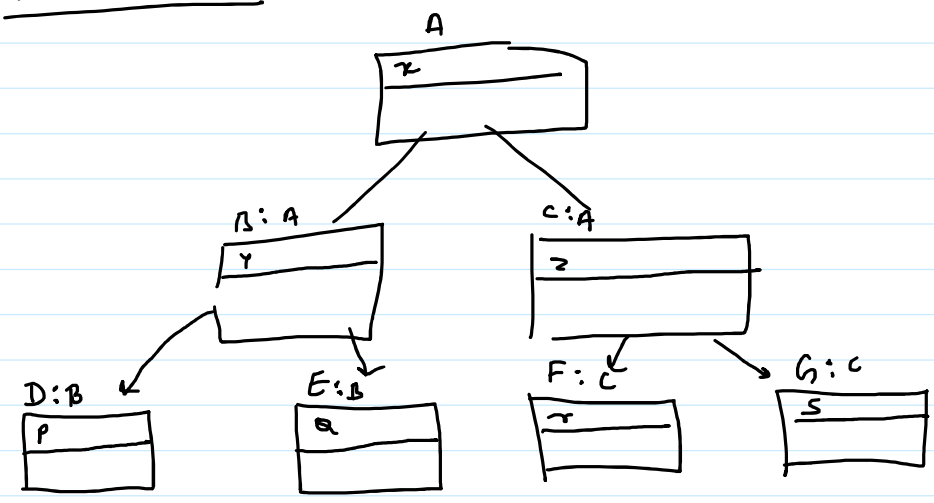
multiple Inheritance :→




```
#include<iostream>
using namespace std;
class A{
    int x;
    public:
        A()
        {
            x=0;
            cout<<"Class A constructor\n";
        }
        A(int x1)
        {
            x=x1;
            cout<<"Class A parameterized constructor\n";
        }
        void output()
        {
            cout<<"X = "<<x<<endl;
        }
};
class B{
    int y;
```

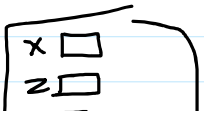
```
    public:
        B()
        {
            y=0;
            cout<<"Class B constructor\n";
        }
        B(int y1)
        {
            y=y1;
            cout<<"Class B parameterized constructor\n";
        }
        void output()
        {
            cout<<"Y = "<<y<<endl;
        }
};
class C:public A,public B{
    int z;
    public:
        C():A(),B()
        {
            z=0;
            cout<<"Class C constructor\n";
        }
        C(int x1, int y1, int z1):B(y1),A(x1)
        {
            z=z1;
            cout<<"Class C parameterized constructor\n";
        }
        void output()
        {
            A::output();
            B::output();
            cout<<"Z = "<<z<<endl;
        }
};
int main()
{
    C c1;
    c1.output();
    C c2(10,20,30);
    c2.output();
    return 0;
}
```

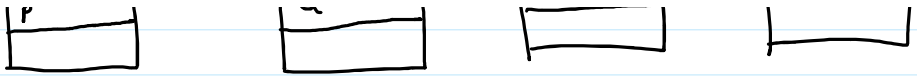
Hierarchical

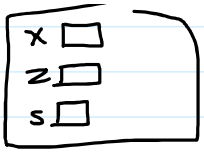


G g1 ;

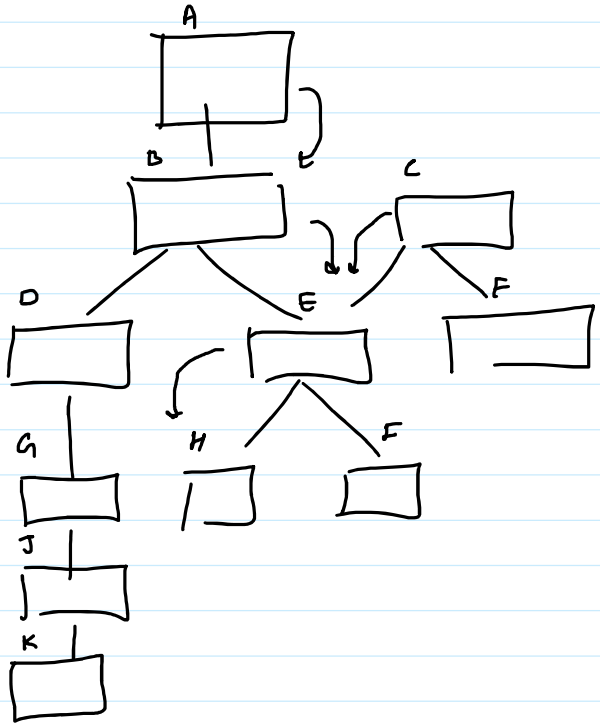
g1

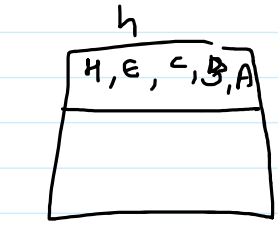






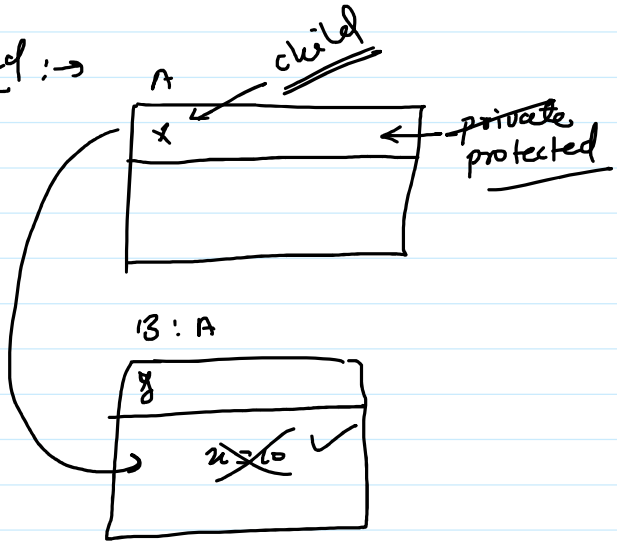
Hybrid :-





H h1 ;

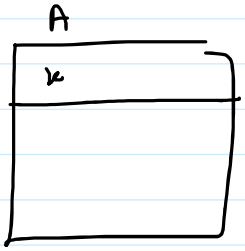
Protected :→

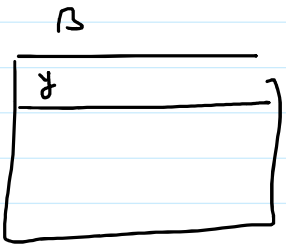



```
#include<iostream>
using namespace std;
class A{
    protected:
        int x;
    public:9
        A()
        {
            x=0;
            cout<<"Class A constructor\n";
```

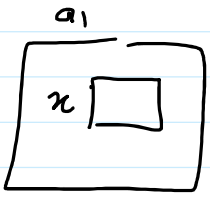
```
        }
        A(int x1)
        {
            x=x1;
            cout<<"Class A parameterized constructor\n";
        }
        void output()
        {
            cout<<"X = "<<x<<endl;
        }
};
class B:public A{
    int y;
    public:
        B():A()
        {
            y=0;
            cout<<"Class B constructor\n";
        }
        B(int x1,int y1):A(x1)
        {
            y=y1;
            cout<<"Class B parameterized constructor\n";
        }
        void output()
        {
            cout<<x<<endl;
            cout<<"Y = "<<y<<endl;
        }
};
int main()
{
    B b1;
    b1.output();
    return 0;
}
```

object slicing :→

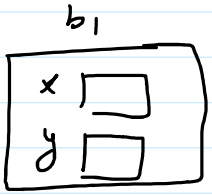




A a1;



B b1;



✓ a1 = b1;

✗ b1 = a1;

A *pa;

pa = &a1; ✓

✓ `Pa = &b1;` ✗ `Pb = &a1;` ✗ `*Pb;` ✓ `Pb = b1;`

par obj = child obj ✓

par pointer = & child obj ✓

static members :→

Types of member variables :→

① Instance member variable (object)

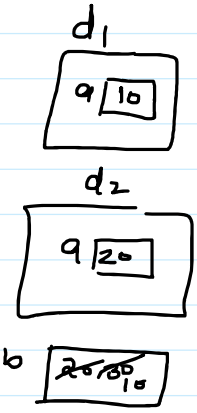
② static member variable (class)

class data
{ public:
int a; ← Instance
static int b; ← static
};

int data::b; ← static member variable
✓ memory

int main()
{
data d1, d2;

d1.a = 10;
d1.b = 20;
b = 10;
d2.b = 30;
d2.a = 20;
data::b = 100;



count objects of a class :—

```
#include<iostream>
using namespace std;
class data{
    public:
    static int count;
        data()
        {
            count++;
```

```
        }
        void output()
        {
            cout<<count;
        }
};
int data::counts;
int main()
{
    data d[10];
    cout<<data::count;
    return 0;
}
```

Instance	static
① No keyword	① static keyword
② Access by an object d1.a	② Access object & class d1.b data::b
③ many copies — obj	③ single copy
④ Default → Garbage	④ Default → zero