- Virtual Functions
- Calling VF using base class reference

★★★ V.imp

(unit-4)

Q) What are virtual fun?

A) • A VF is declared by using virtual keyword in base class
eg virtual void fun();

- VF allows a base ptr/ref to call a fun in derived class.
- Normally a base ptr can't call a fun in derived class. It can only call a fun in base class. // Explained in last topic

Q) How are VF used? When do you use a VF?

A)
```
B   fun { "Base fun called" }
↓
d1  fun { " d1 fun called" }
↓
d2  fun { " d2 fun Calle " }
```

```
fun()
fun()
```

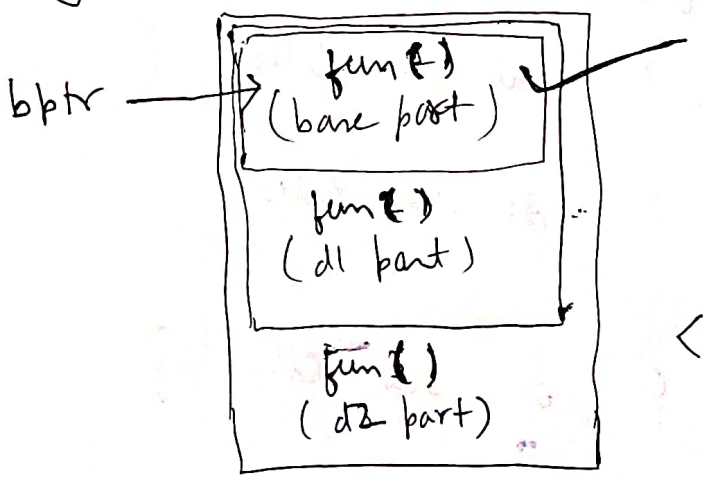
★

- VF - r always used when you are overriding a fun & calling that fun thru base ptr.

- VF, fun overriding, base ptr are combinedly used to implement a VF.

- Ex of a ~~VF~~ non VF

- Suppose base class contains a fun called fun();
- This fun is overridden/redefined by d1 & d2, with same param.
- Now d2 will have 3 fun().
  two r inherited from base, d1
- Suppose a bptr is pointing to a d2 obj.



<see the struct. of classes in previous page>

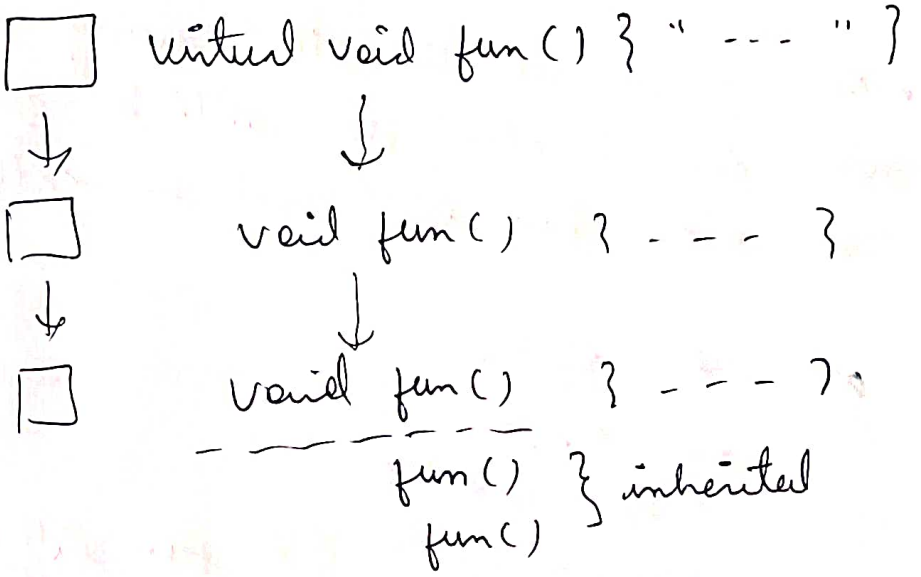
obj of d2

• when this ptr calls the fun, then base class fun will be called.

```
d2 obj-d2;
base *bp = &obj-d2;
  bp -> fun( ); // base fun will be
                   called.
```

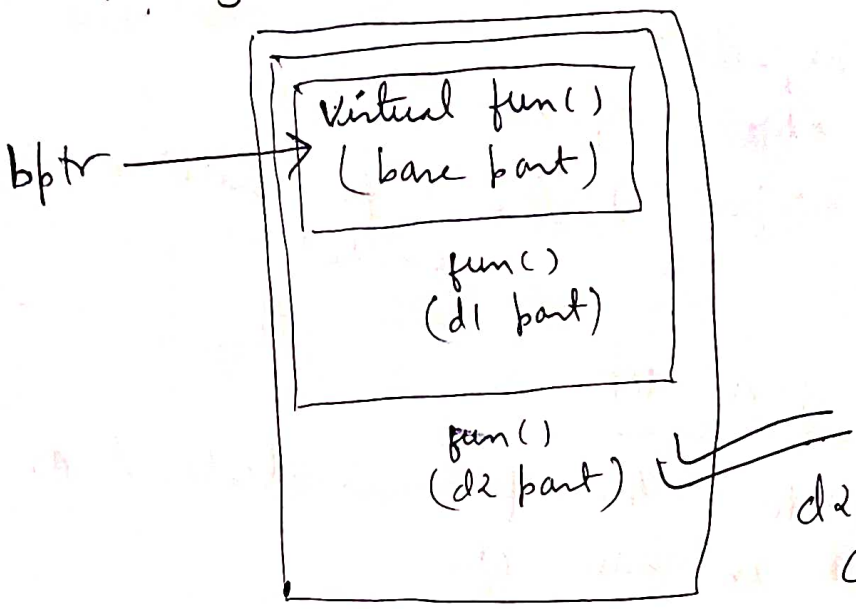
Example of a VF

• Now suppose the fun is declared as virtual in base class.



• In this eg., fun is declared as virtual in base class.
• d2 contains three fun, 2 r inherited

• Now suppose a bptr is pointing to d2 obj.



• Now if the fun is called using a bptr then derived class fun will be called. Base class fun will not be called.

eg
```
d2 obj_d2;
base *bptr;
  bptr = &obj_d2;
bptr -> fun(); // d2 fun will be called.
```

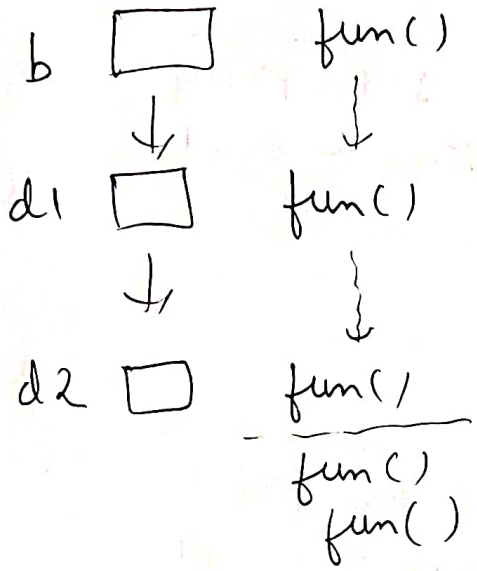
Same rule applies to ~~std~~ base ref.

```
d2 obj_d2;
base &ref = obj_d2; // create ref.
  ref.fun(); // d2 fun called.
```

★ Here VF is called using a base class reference.

Understanding the difference between Vir Fun & Non VF.

- Suppose a fun() is overridden by d1 & d2 class.



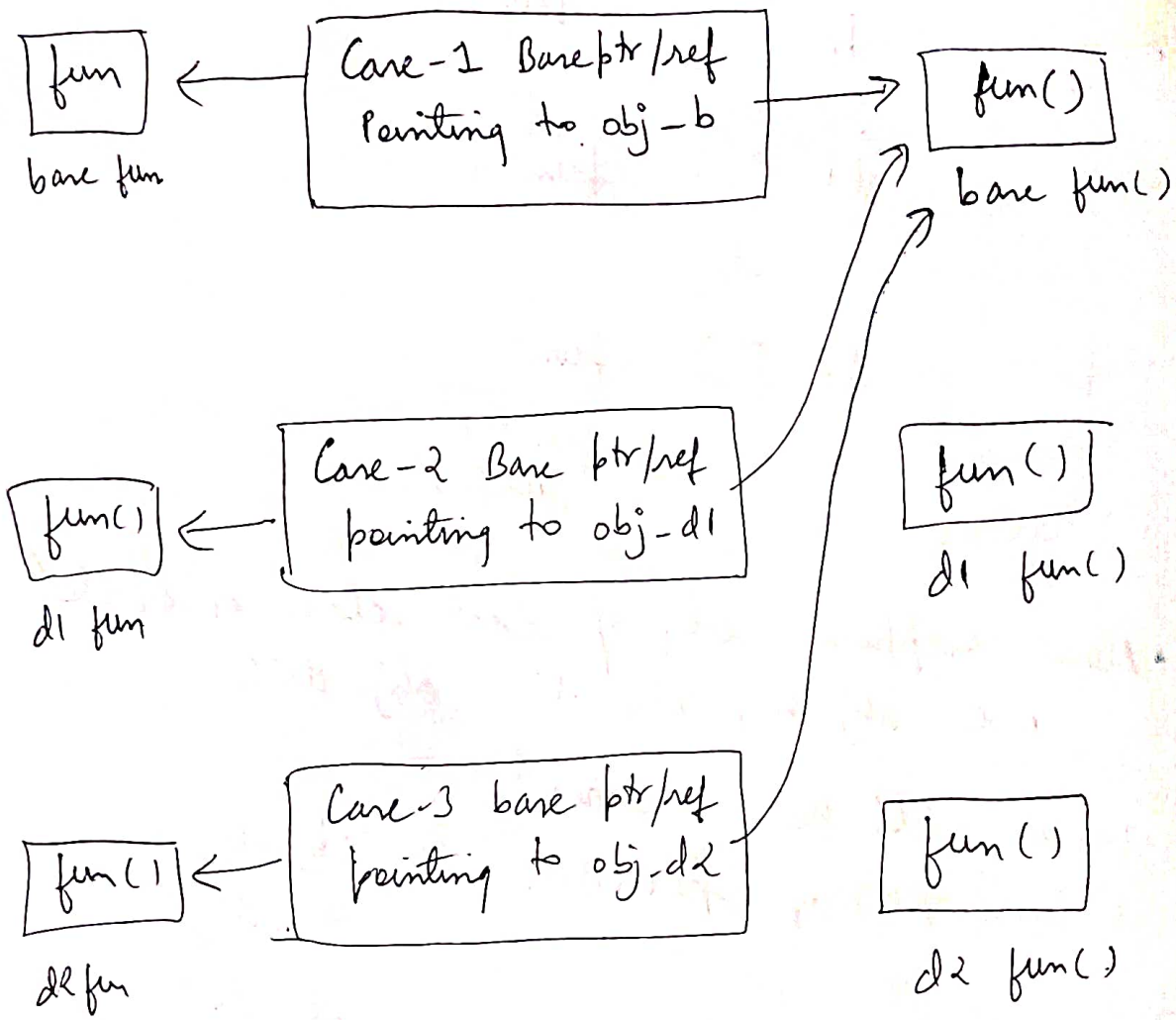
- Now suppose obj of each class is created. i.e obj-b, obj-d1, obj-d2;

- There will be 3 cases:
  1) base ptr/ref pointing to obj-b;
  2) " " " " " obj-d1;
  3) " " " " " obj-d2;

- In each case fun is called using base ptr/ref.

- If fun is non virtual, then base fun will be called in each case. It doesn't matter wh. obj is being pointed to (base, d1, d2)

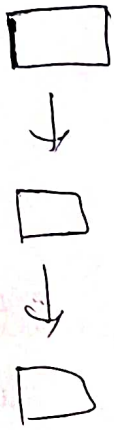
• If the fun is ~~the~~ vir, then each obj own fun() will be called. So VF means that each obj's own fun will be called rather than base's fun()



Note: Vir fun, fun overriding, baseptr ment be combinedly used.

- There is no sense in following cases :-

1) A fun is declared as virtual in base class, but it's not overridden.



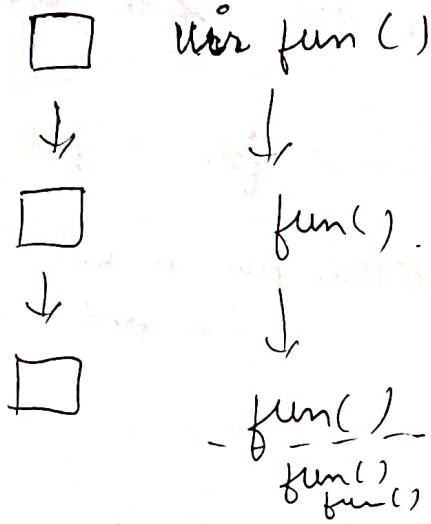
virtual fun() {---}

//This fun is not overridden (redefined) in derived clases.

- In this case, a fun is declared as Vir in base class, but it's not overridden in d1 & d2.
- Now when a bptr is pointing to d2 obj, it will have only 1 fun() wch is the base class fun. So that fun will be called.
- Declaring a fun as vir means you want to call a derived class fun using base ptr but there is no der class fun in this case, so base fun will be called.

2) A fun is declared as ~~fr~~ vir & it is also overridden. But it is not called using a ptr/ref. Instead it is called using an obj.

eg



- In this eg, the fun is called using obj directly.

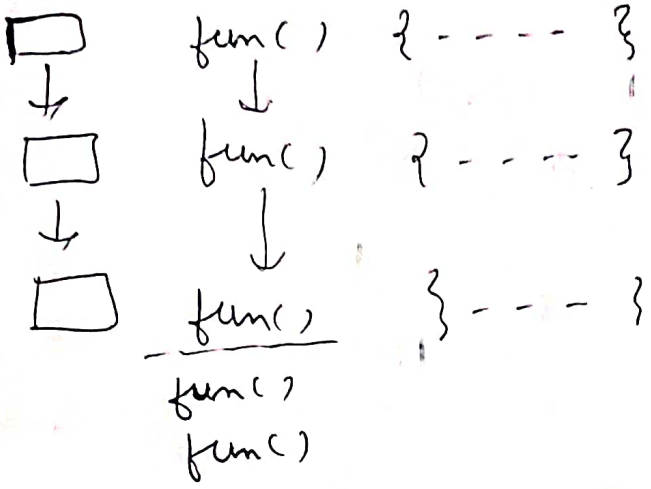
```
d2 obj-d2;
obj-d2.fun(); // No obj used.
```
ptr/ref

- Here d2's fun is called. This behaviour is same as using bptr to call a VF.

- But, the difference is that bptr also allows runtime polymorphism, wh. can't be achieved using obj.

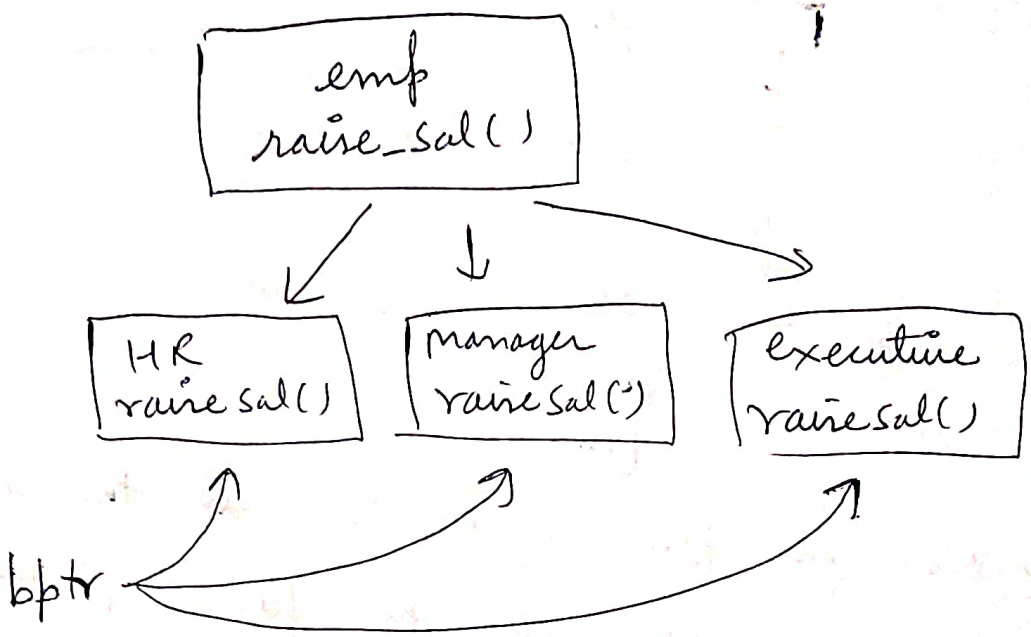
- So bptr shud be used to call VF.

3) A fun is overridden & bptr is also used, but it is not declared as vir.



- Non vir fun r created in this eg.
- When a bptr calls the fun( ), then base fun( ) will be called.
- But we want to call the derived class fun( ). So fun( ) must be declared as ~~non~~ VF to do so.

Practical Example of a VF



- Suppose there is a class called emp. It is derived by HR, Manager, executive
- There is a fun() called raise_sal() in emp class. It is overridden by HR, mgr, executive classes.
- Each class will have a diff implementation of the raise sal() fun, becoz salary criteria may be diff for diff employee.
- When a bptr pts. to diff obj, then each obj's own raise_sal() fun shud be called Base's raise_sal() fun shud not be called.
- Therefore, this fun is declared as Virtual in this case.