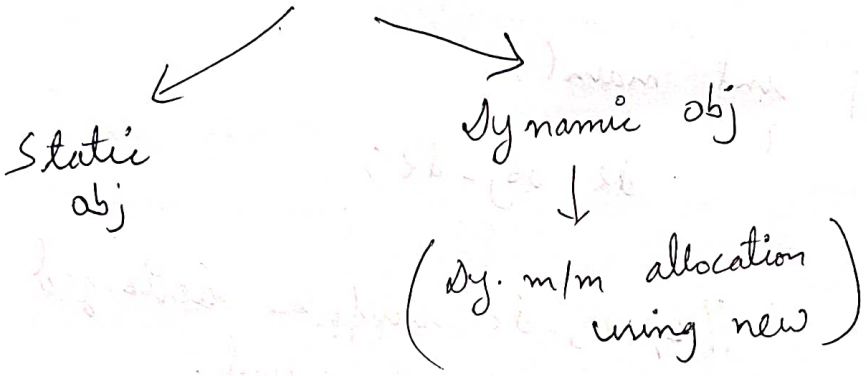
Virtual Destructors

(unit-4)

- obj can be created in 2 ways



- Destructors behave differently for static & dy. obj.

```
B ~base() { "b dest called" }
↓
d1 ~d1() { "d1 " " " }
↓
d2 ~d2() { "d2 " " " }
```

- Let us consider 4 diffent cases:

Case-1 static obj is created.

- If a static obj is created, then there r 2 behaviour of this obj.

1) It will be destroyed by OS when prog. finishes
2) Dest^{r} of all classes will be called from bottom to top.

//When a der class obj is destroyed, then all dest^r s called. Explained in unit-3 //

eg. int main()
{
    d2 obj-d2;
}
// 1) obj-d2 will be destroyed.
// 2) All dest^r called.

Case-2) Dy. obj is created.

- A dy. obj is allocated m/m by the user, whereas a static obj is allocated m/m by the OS.
- So OS will auto^y destroy destroy a static obj.
  But dy. obj will not be destroyed auto^y by OS.
- In the next code, the behaviour of dy-obj is as follows.

1) The obj will not be destroyed
2) No dest will be called.

```
int main()
{
    base *bp;
    bp = new d2;
}
```

dy. obj of d2 is created.

it is assigned to a base ptr.

// 1) obj not destroyed
2) No dest called. So there will be no o/p.

Note :-

- Base ptr is being used to allocate a der class dy. obj bcoz it needs to be used polymorphically
- i.e to support RT polymorphism base ptr is reqd.

Case-3) · Dy. obj is created.
· It is destroyed by user using delete operator.

- The behaviour of dy. obj in this case :

1) obj will be destroyed (using delete)
2) Only base dest will be called.
(Not all dest called)

Q) Why do we need a Virtual dest (VD)?

A) When a dy. obj is destroyed using delete operator, then it has an "undefined" behaviour.

↓

- Dest of all classes r not called.
  Dest of only base class is called.
- To resolve this undefined behaviour VD r used.

```
//eg. of non VD

int main()
{
    base *p;
    p = new d2;
    delete p;
//m/m is released for dy. obj using
delete. It results in undefined
behaviour.
}
//1) obj is destroyed (using delete)
  2) only base dest called.
     All dest shud have been called.
```

Ex

Case-4 VD r used

- Base class must declare the dest. as virtual.
- Dy obj r created & destroyed using delete.
- The behaviour will be:

1) obj will be destroyed
2) All dest. will be called in correct order (bottom to top) beoz VD r used.

Q) what r VD?

A) VD r used to ensure that all dest r called when a ~~der obj class obj is~~ der class dy. obj is released / destroyed using delete.

```
//in this case assume VD r used in base
  class
  // virtual ~base ( ) { "B dest called"; }

int main ( )
{
      base *p;
       p = new d2;
       delete p;
}// 1) obj is destroyed.
   2) All dest r called.
```

Summary

Q) What r VD?

A) • Dest r declared as vir using virtual keyword in base class destructor - Not needed in der class.

• VD allows calling of dest in correct order when a dy. obj is destroyed.

• When a non vir dest is used then delete opr has undefined behaviour. i.e only base dest. may be called.

Q) When do you use a VD?

A When a dy. obj is created, it is used polymorphically means base ptr is used for a der class obj. and this obj is destroyed using delete.

• So if these three ~~cond^n~~ condition r fullfilled then VD must be used.

1) dy. obj is created
2) It is used polymorphically
3) m/m is released using delete.

• If no VD is used → then undefined behaviour (only base dest called)

• If VD is used → all dest called.

Q) Extra quest:-

what if an static obj is destroyed using delete operator?

eg int main()
{
d2 obj_d2;
//static obj.

base *p = &obj_d2;
//p pointing to static obj.

delete p;
// release m/m of a static obj

}//In this case abnormal prog termination will occur.

- A user can't release a static obj using delete operator.
- OS must release " " " .