PAGE No.
DATE / /

Assignment - 21

Q1] Why we can not create object of such a class which contains pure virtual function?

Ans

1] If a class contains a pure virtual function in it then, we cannot create object of that class.

2] We cannot create the object because the VTABLE of that class is incomplete. As VTABLE is a part of object our object is also incomplete.

3] ~~If object is created for a class which contains Pure virtual function in it, then the~~

3] It is compulsory to provide definition inside the derived, which is the abstract in the base class

4] If the derived class is unable to provide the definition, then the compiler will generate error.

5] ~~If object~~ If we try to create object for a class which contains Pure virtual function in it then, the reserved slot for the address of definition in the derived class in the VTABLE might receive another address which would not be safe.

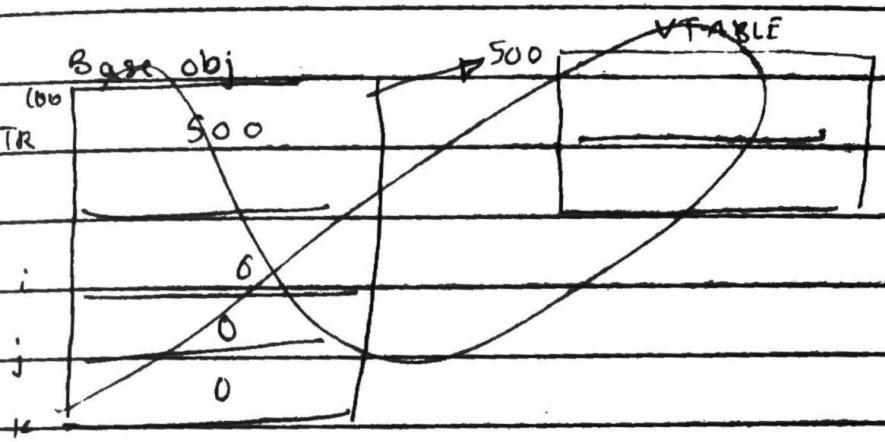
Q2] What is meant by Pure virtual function?

Ans

1] If a class contains pure virtual function then that class is considered as abstract class.

3] If a function is Pure virtual, it means it is a function without body, i.e abstract function.

5] Inside the derived class we provide the definition of the ~~addition~~ abstract function.



(6i] What happens if base class contains virtual function ~~vde~~ under private access specifier?

Ans

1] Any private characteristic or behaviour of a class cannot be accessed outside the class.

2] If a base class pointer tries to access derived class, but the derived class will only have the publicly inherited ~~the~~ behaviours (functions).

2] Since virtual method will also be private, it cannot be inherited by the derived class

PAGE No.
DATE / /

Q4] What is meant by Abstract method & Concrete method

Ans A Pure Virtual function is recognized as Abstract Method
It is recognized as abstract when the function precedes with virtual keyword & proceeds with = 0.

Eg: Virtual void fun = 0.

A function which has its own definition is called as Concrete method.

Q5] Explain below syntax & draw its diagrammatic representation

```
Class Base
{
    public:
        int i;
        float f;
        virtual void gun()= 0;
        virtual void run() = 0;

        virtual void run()          // 1000
        {
            cout << " Base Run";
        }
};
```

DATE / /

```
class Derived : public Base
{
public:
    int i;
    double d;

    virtual void sun()    // 3000
    {
        cout << "Derived sun";
    }
    void fun()            // 3000
    {
        cout << "Derived fun";
    }
    virtual void gun()    // 4000
    {
        cout << "Derived gun";
    }
    virtual void mun()    // 5000
    {
        cout << "Derived mun";
    }
};
int main()
{
    Base * bp = NULL;
    Derived dobj;
    bp = &dobj;
    bp -> fun();
    bp -> gun()
    bp -> sun()
    bp -> run()
    bp -> mun();
    return 0;
}
```

DATE / /

1] The code uses 3 object oriented paradigm
i] Encapsulation: Binding of characteristics & behaviour together
ii] Inheritance : Reusability
iii] Polymorphism : Single name & multiple behaviours

2] In our code we use Run-time polymorphism which performs Late Binding.

3] Binding simply is connecting Function call to its body.

4] To achieve Run-time polymorphism our code should contain:
1] ~~Single~~ At least single Level Inheritance
2] Redefinition of function (across classes)
3] UPCASTING : A small capacity pointer points to a large data.

5] All of these features are satisfied in our code.

6] Base class is the abstract class because it contains pure virtual function in it

7] A function without body is Pure virtual / abstract function.

8] In derived class we provide definition of that abstract methods / function.

9] Base pointer is initialised to NULL to avoid segmentation faults.

10] Then the Base pointer (small capacity) points to a (large data) derived object.

PAGE No.
DATE / /

since derived object inherits from base object.

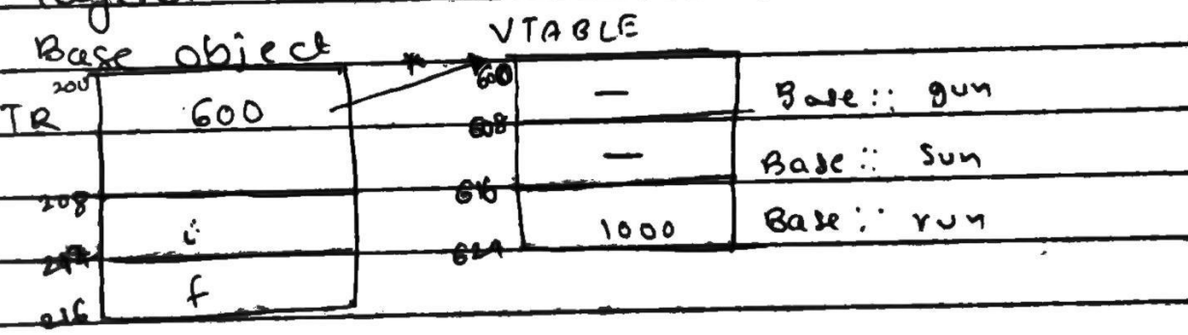
~~Base object~~

If a class containing virtual function or if a class is derived from such a class which containing virtual function in it then the first 4/8 bytes of memory is reserved for a VPTR.

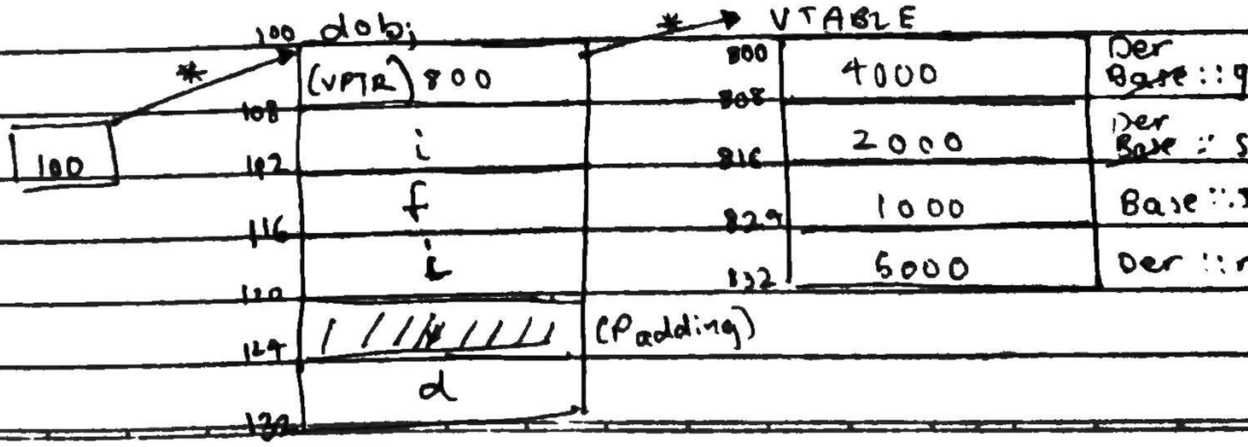
13] This VPTR points to the base address of a VTABLE. The VTABLE holds address of all the virtual functions in it.

14] These addresses are stored sequentially in the sequence they are defined.

15] object layout



16] when object gets inherited it gets inherited along with the VTABLE but has new address



PAGE No.
DATE / /

16] The use of virtual keyword is necessary.

17] Without virtual keyword the Base pointer calls the method of its type (i.e. Base) instead of calling method of type to which it points.

18] This could've generated error because the VTABLE in the Base class is incomplete due to abstract methods in it.

19] When we use the virtual keyword, the compiler checks the pointed type (i.e. derived) & checks if the derived class has redefined the Base class method i.e. abstract method. If yes, then it overrides the Base class abstract method with the concrete method in the derived class.

20] Thus the empty spaces in VTABLE is filled with addresses of the definitions of abstract method in the derived classes.

21] Gun() & Sun() are Pure Virtual functions, while Run() is a virtual function.
Run() does not have redefinition in the derived class hence, bp -> run() will call method from Base class

22] Padding gets added

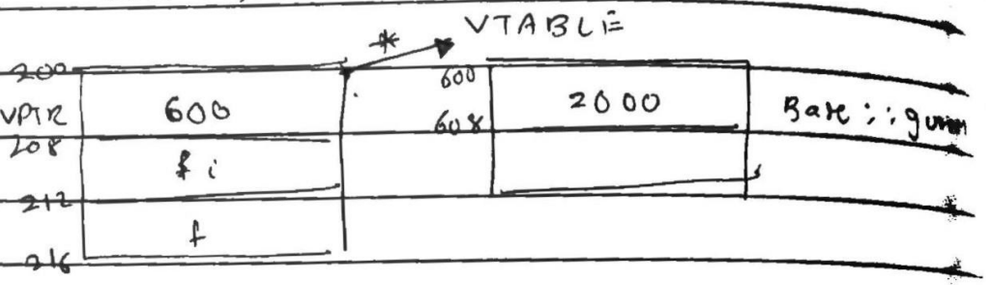
PAGE No.
DATE / /

Draw the memory layout & draw the assembly instruction of function calls.

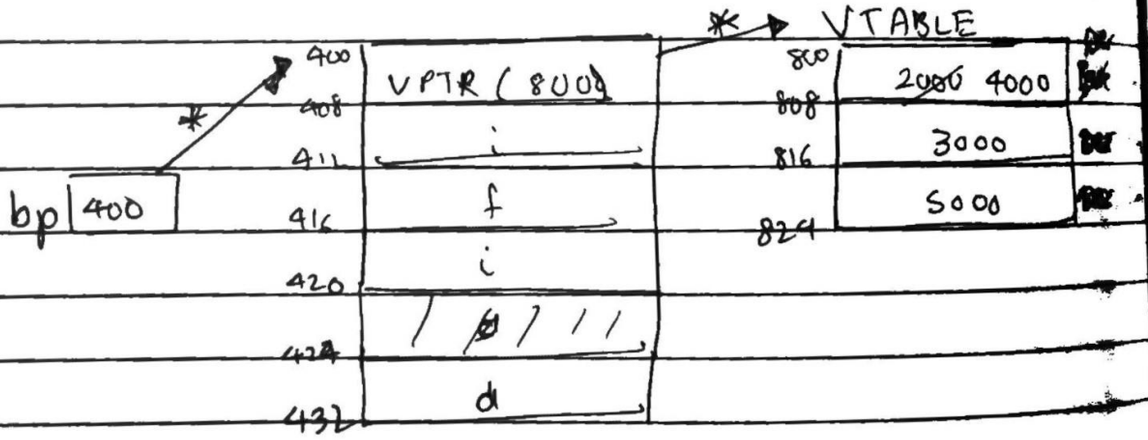
```
class Base
{
    public:
        int i;
        float f;
    void fun()          //1000
    {
        cout<<" Base fun";
    }
    virtual void gun()      //2000
    {
        cout<<" Base gun";
    }
};
class derived : public base
{
    public:
        int i;
        double d;
        virtual void fun()      //3000
        {
            cout<<" Derived fun";
        }
        void gun()          //4000
        {
            cout<<" Derived gun";
        }
        virtual void sun() {    //5000
        {   cout<<" Derived sun";
        }
};
```

```
int main()
{
    Base *bp = new Derived;
    bp -> gun();

    return 0;
}
```

Base object



Derived object on heap



1] When we call

bp -> Gun()

2] when we call any OOP method the address of the object gets stored inside the ESI register. This It is considered as 'this' pointer in assembly which holds address of the object

PAGE No.
DATE / /

Due to PUSH ESI

Due to the PUSH ESI, the address of object gets stored in ESI register. Now our ESI contains 400 in it.

4] Now the first 4 bytes of the object gets fetched & stored inside the EAX register (Accumulator)

MOV EAX, [ESI] i.e. MOV EAX, 800

This statement says store the first 4/8 bytes of to which the ESI (this pointer) points to

5] Now the compiler will check the sequence & get position of gun function from the base class.

6] As the gun function is defined at first index, the compiler will call the function from first index of VTABLE

7] To call the function we use

CALL [EAX + 0]

0 stands for the position in the VTABLE

This statement says call the function which is at 800 +0 & square brackets give dereference the address.

Therefore 4000 will be called i.e Derived gun().

PAGE No.
DATE / /

8] All calculations are performed at time of compilation & also the Base class methods are overridden by derived class methods if they virtual at the time of compilation. But which method is at call i.e. ith index

[EAX + i]

↑ i.e. index of the ith virtual method in the VTABLE

gets called at run-time due to which

it is called as run-time polymorphism.

Q7] Draw the objects layout of below system

```
class Base1
{
    public:
        int i;
        float f;

        virtual void fun () = 0;
        virtual void sun () = 0;
        virtual void run ()          // 1000
        { }
};
class Base2
{
    public:
        int j;
        float g;
```

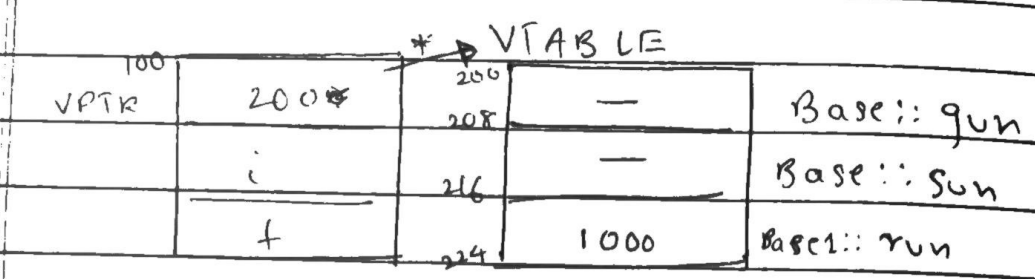
PAGE No.
DATE / /

```
virtual void mun() = 0;
virtual void fun() = 0;

void fun()                    // 2000
{}
};

class derived : public base1, base2
{
public:
    int i;
    double d;

    void sun()                // 3000
    {}
    void fun()                // 4000
    {}
    void gun()                // 5000
    {}
    void mun()                // 6000
    {}
};
int main()
{
    derived dobj;
    return 0;
}
```

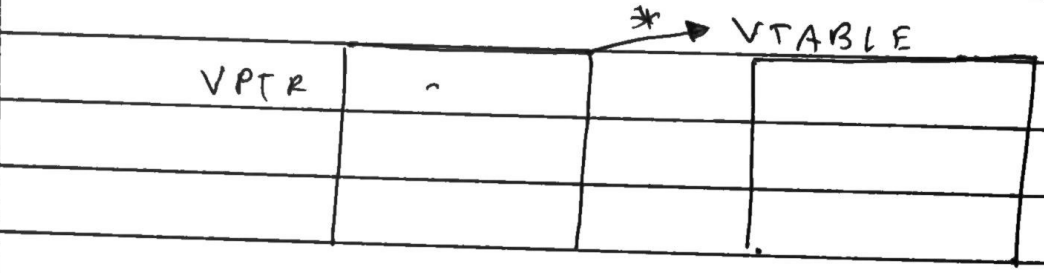
Base 1 object

~~Base 2 object~~

~~dobj~~



Base 2 object



PAGE No.
DATE / /

(Q8) What are the ways in which we can achieve upcasting in object oriented language?

Ans) UPCASTING : When a small capacity pointer points to a large data is called as upcasting

Upcasting can be achieved through : pointer or using reference

```
Base *ptr = &dobj;    // Pointer

Base &bobj = dobj;    // Referencing
Base &bobj = &dobj
```

Q9] What is the use of pure virtual function?

Ans -

1] If you want to present only an interface for derived class.

2] You don't want anyone to create object of that class.

3] only upcast it so that its interface can be used.

Q10] Can we create a pointer of a class which contains pure virtual function in it?

Ans

1] We cannot create object of a class which contains pure virtual function in it but we can create pointer of the abstract class for upcasting.