C++

Object Oriented Programming

procedure / structure

↓

Subprogram

funtion subroutine

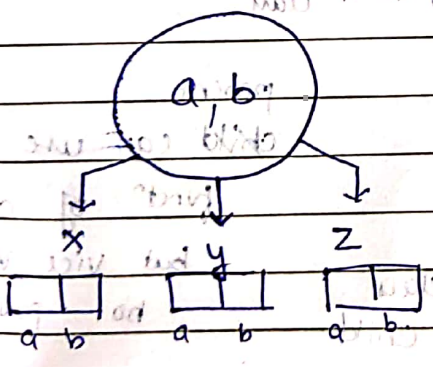
It emphasizes on two major parts

- Class
- Object

Class :- * user defined datatype
* it is similar to structure variable
* it is a group of different type of data and function. And function performs the operation on respective object data.

Object :- * its run time entity of class
* it is similar to structure variable.

Structure

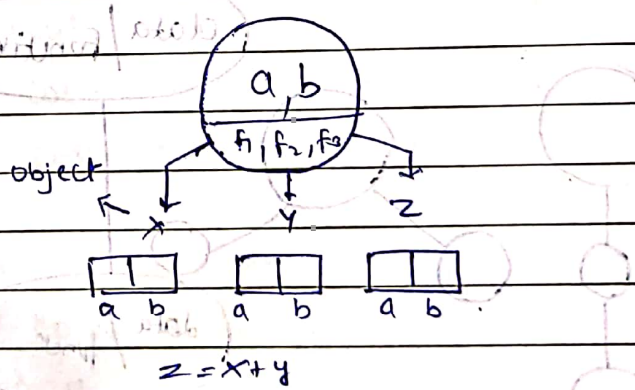


x.a x.b . . .

z.a = x.a + y.a

a must be of primary type
(Not array, string)

Class



z = x + y

Primary as well as secondary data can be added
(string array. . . .)

Data hiding :- No one can perform any operation on data without a pure knowledge of function.

In OOP '+' can be used in different way at different place.

The way by which group data & function together is known as data incapculation. (as a combined entity)

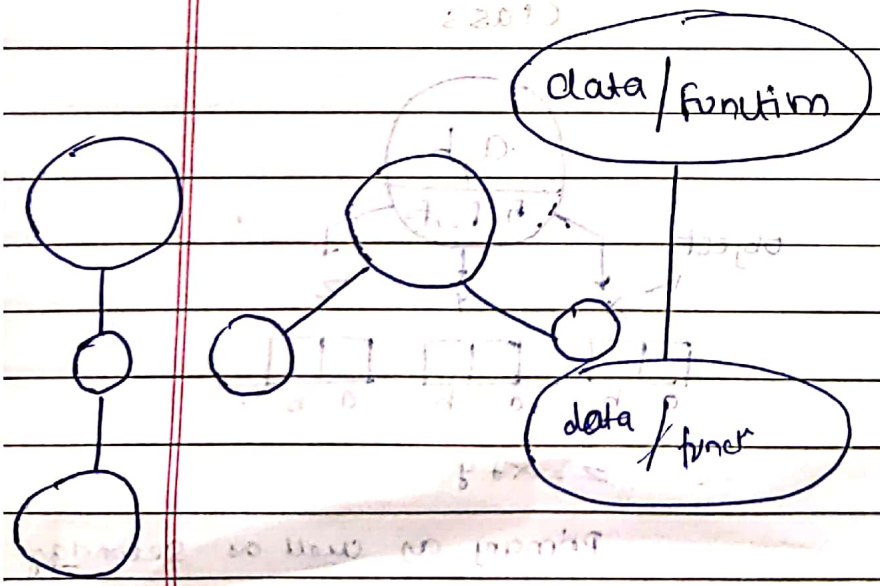
Polymorphism :- One thing used in different way

Eg area(r);
In C++ area(l,w);
area(a,b,c);

→ single function used for different purposes.

In C, areac(r);
arear(l,w);
areat(a,b,c);

Inheritance



parent class

parent child can use data/func of child but vice versa not possible

class child

Input / Output

Data flow in two direction
input device → prog
prog → output device

cout << double quoted value / variable name

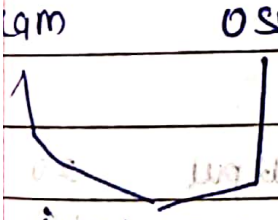
it is an object of std. class ostream
It represent the std. output device

```
cout << a;
cout << b;
cout << c;
```

```
cout << a << b << c;
```

102030

```
cout << a << " ";
cout << b << "\n";
cout << c << "\n";
```



for changing line

```
cout << a << "\n";
cout << b << endl;
```

end of line (manipulator)

If name of local & global are same then global can't be used in main body.

```
a=10;
void main(){
a=20;
}
```

output ⇒ 20

```
#
#include <iostream.h>
void main(){
        clrscr();
        int a, b, c;
        cout << "Enter 2 numbers :";
        cin >> a >> b;
        c = a+b;
        cout << "a=" << a << endl;
        getch();
}
```

Scope Resolution

Eg

```
#
#
int a=10;
void main(){
        clrscr();
        int a=20;
        cout << a << endl;
        cout << ::a << endl;
        getch();
}
```

Output 20
10

:: → for global variable.

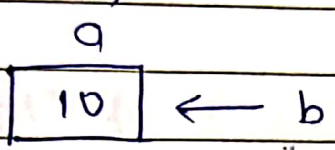
** Local & global both can be used in c++ even if their names are same by concept of scope resolution.

Refrence Variable (Secondary Name / Alias)

→ Another name of existing variable.

datatype &varname = existing var;
↓
Reference operator

```
int a = 10;
```




```
int &b = a;
```

One memory being used by two names (No overwrite)

```
void main(){
    clrscr();
    int a = 20;
    int &b = a;
    b = 100;
    cout << a << endl;      → 100
    cout << &a << endl;     } Returns same address
    cout << &b << endl;
    getch();
}
```

int x → value
int *x → ref.
int &x → Ref variable

Syntax ⇒ pass by value
= —"— by variable

Swaping Using Ref. variable

Q&A

```
#
#
void swap (int &, int &);
void main()
{
 clrscr();
 int a,b;
cin >> a >>b ;
cout << a <<" "<< b<< endl ;
 swap(a,b);

cout <<a<<" "<< b <<endl;
 getch();
}
void swap ( int &x , int &y )
{
    int t ;
    t=x ;
    x = y;
    y = t ;
}
```

function defined before main body (C++)

```
#
float convert ( float x ) {
        float y ;
    y = 9 * x/5 + 32 ;
        return y ;
}
void main() {
        clrscr();
    float c, f;
    cin >> c;
    f = convert ( c );
    cout << f;
    getch();
}
```

Inline function :- A function which performs the process at the same line from where we call.

N.B. It will work only for very less line of codes (2/3 line)

Copy & pasting & paste at the place of calling.
Saves the shifting time.

inline float convert (float x)

FUNCTION OVERLOADING

Functim dellared more than once by same name. Each declaration, there must be a difference between either no. of parameters or type of parameters.

```
#include <iostream.h>
void show();
void show (char);
void show (char, int);
void show (int);
void main()
{
    show();
    show('*');
    show('t', 30);
    show (50);
    getch();
}
void show()
{
    cout << " 0 argument ";
}
void show (char x)
{
    cout << x << " 1 argument";
}
void show (char x, int n) {
    cout << n << x << "2 argument"; }
void show (int) {
    cout << x << " 1 argument";
```

M II → function declares once, define once but calling is no. of ways.

→ default arguments / optional argument :-
it is not compulsory to use these arguments while calling.

→ The basic role of default values during function calling :-
If u miss or not pass any argument, so, function declaration argument values automatically pass on to function def.

Eg :-

```
#include <iostream.h>
#include <conio.h>
void show (char='?', int=20);
void main()
{
show(.);
show('*');
show('+', 35);
getch();
}
void show(char x, int n)
{
cout << x << " " << n << " 2 argument\n";
}
```

default value → optional argument.

compulsory → default

void show (char , int = 30);

show ('*');
show ('*', 20);

& void show (char = '*', int); // default-value missing
show (, 30); ✗ → R.

show (30);
↳ R to L in def.
↳ L to R in calling

Assignment :-

Write a program :-

to ~~are~~ calc. the area of circle, rect, triangle using function overloading.
(switch case)

a) using 3 declaration & definition
b) using only 1 decl. & defn.

Data hiding diff b/w C++ & struct

Class & Object

class classname
↙ keyword ↘ name of datatype as class

① Keyword :- private
↳ it is a keyword and access modifier if we define anything under private its only used within the class.

Qa :- #
#

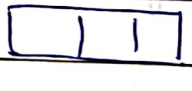
```
class product {
    private:
        int pno;
        int pqty;
        float pcost;
};
```

```
void main() {
    product x, y, z; //object dec
    cout << sizeof(x);
    x.pno = 101;
```

To be continue (Pg)

② public :- keyword & access modifier
if anything ~~of~~ defined under public it can be used within the class as well as outside of the class but through an object.

```
Struct product
{
    int pno;
    int pqty;
    float pcost;
};
struct product x;
```



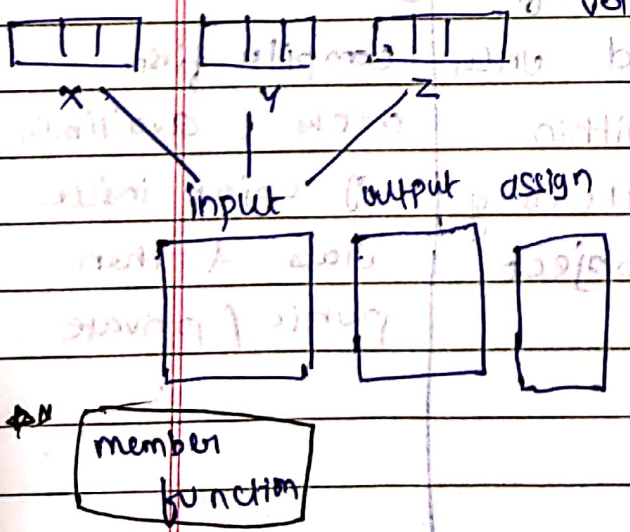
memory

compiler first checks & availibility of object inside class & then public / private.

Qeq #

```
class product
{
private :
    int pno ;
    int pqty ;
    float pcost ;          // cost per unit
public :
    void input ()
    {
        cin >> pno >> pqty >> pcost ;
                calling object data
    }
    void output ();
        {
            cout << pno << " " ;
            cout << pqty << " " ;
            cout << pcost << endl ;
        }
    ~~void assign () { cin >> y.assign (102, 30, 90.5)~~
    void assign (int a, int b, float c) { pno = a ;
                                          pqty = b ;
                                          pcost = c ; }
};
void main () {
    clrscr ();
    product x, y, z ;  // object declaration
    x.input ();
    y.output ();
    y.assign (102, 30, 90.5);
    y.output ();
    t1 = x.total ();
    t2 = y.total ();
    t3 = t1 + t2 ;
```

x.input ();
↓
calling object



Total cost of n objects

Date
Page

convincing is making functions

Make small functions

```
float total ()
{
    float t;          // local variable
    t = pqty * pcost;
    return t;
}
```

```
cout << t1 << " " << t2 << " " << t3;
getch();
}
```

y input() function is under private section then.

write

```
public:
    void call()
    { input();
    }
```

// Make a new function

// x.input

x.call

ign (102, 30, 905);

Eg #
#

```
class measure
{
private :
        int feet;
        float inch;
public :
    void getm()
    {
        cin >> feet >> inch;
    }
void setm (int f, float i) {
        feet = f, inch = i;
    void shown()
    {
cout << feet << " " << inch << endl;
    }
void add (measure x, measure y) {
        feet = x.feet + y.feet;
    inch = x.inch + y.inch;
    if (inch >= 12) {
            feet ++;
            inch = inch - 12;
    } }
};
void main () {
    clrscr();
    measure m1, m2, m3;
```

```
m1.getm();
m2.setm(2,7);
m3.addm(m1,m2);
m3.showm();
getch();
}
```

We can add multiple objects.
3 or more

To show sum of m_1, m_2, m_3 object in m4.

```
measure m1, m2, m3, m4;
m4.addm(m1,m2);
m4.addm(m4,m3);
m4.showm();
```

To add multiple objects

```
clrscr();
measure m[5], s;
int i;
for(i=0; i<5; i++){
    m[i].getm();
}
s.setm(0,0);
for(i=0; i<5; i++){
    s.add(s,m[i]);
```

function → class public

m3 = m1.add(m2)

calling + passing

classmate

Date
Page

argument

s.showm();
getch();
}

Method 2

measure add (measure x)
{
measure t;
t.feet = feet + x.feet;
t.inch = inch + x.inch;
if (t.inch >= 12){
t.feet ++;
t.inch = t.inch - 12;
}
return t;
}

m3 = m1.add(m2);

For adding arrays

s = s.add(m[i]);

possible way

adding objects

object.add (object, object);
object = object.add (object);

d1 - calling
d2 - pass

if (a>b) Comparison

m1 comp (m2)

de class
cont.)

```
int comp (measure x) {
        float d1, d2;
d1 = feet * 12 + inch;
d2 = x.feet * 12 + x.inch;
if (d1 > d2)
return 1;
else
return 0;
```

calling

pass

only 2 values returned. which helps in executing if Statement if (1/0).

```
void main() {
    clrscr();
measure m1, m2;
int k;
m1.getm();
m2.getm();
k = m1.comp (m2);
if (k==1) {
        cout << "m1 largest \n";
        m1.showm();
}
else
{
    k = m2.comp (m1);
if (k==1) {
        cout << "m2 largest \n";
    m2.showm();
}
else { cout << "Equal \n";
}
```

Comparing multiple objects

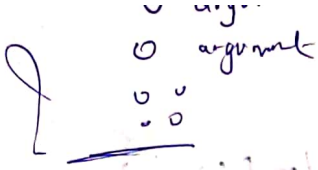
```
void main() {
    clrscr();
    measure m[5], t;
    int i, p;
    for (i=0; i<5; i++) {
        m[i].getm();
    }
    t = m[0];
    p = 0;
    for (i=1; i<5; i++) {
        if (m[i].comp(t)) {
            t = m[i];
            p = i;
        }
    }
    t.showm();
    m[p].showm();
    cout << p << endl;
```

const/dest.

features of constructor

- It is special function which calls automatically whenever an object declared or occupied a memory.
- The name of constructor is same as name of class.
- Constructor mostly define in public section of the class.
- A constructor not having any return type.
- Once we define a constructor, we can't skip or stop the execution of constructor.
- class having more than one constructor is called constructor overloading, In that case, we declare an object a no. of ways.

By default, all class having one constructor called as default constructor which perform nothing but once we define our own constructor there is no existance of default constructor.



⇒

```
measure()
{
    feet = 0;
    inch = 0;
cout << " 0 argument \n";
}
measure (int f, float i) {
    feet = f;
    inch = i;
cout << " 2 argument \n";
};  measure (float x) {
    feet = x;  inch = (x + feet) * 12;
    cout << " 1 argument \n";
};
void main() {
    clrscr();
    measure m1;
    measure m2 (6, 9);
    measure m3 (6, 5); // or measure m3 = 6.5;
    m1.showm();
    m2.showm();
    m3.showm();
    getch();
}
```

6.5 feet

0.5 feet

Compiler not allows to pass an object into the constructor.

~~If memory is~~

✗ measure (measure x) { → memory is created again & again process goes ∞.

feet = x.feet;

inch = x.inch;

measure x = m3;

✓ measure (measure &x) {

feet = x.feet;

inch = x.inch;

cout << " copy constructor \n";

};

Void main ().{

clrscr();

measure m4(m3);

m4.show();

Distructor :- It is a special functim which calls automatically whenever the object destroy from memory.

→ The name of distructor is same as name of class but followed by a symbol tilde (~)

→ Distructor also defined in public section of the class.

→ Distructor is not having any return type. Once we define the distructor it's not possible to stop skip the executim of distructor.

→ Distructor can't overload.

```
~measure() {
    cout << feet << " " << inch << endl;
}

void call () {
    measure m1;
    measure m2 (6,9);
    measure m3 = 6.5;
    measure m4(m3);
    m1.showm();
    m2.showm();
    m3.showm();
    m4.showm();
}
```

Last in first out

```
void main ()
{
clrscr ();
call ();
getch ();
```

Member Function defined outside of class

eg

return type | classname :: | functionname (parameters)

→ is member

eg

```
class measure {
    private :
        int feet;
        float inch;
    public :
        void getm(); // dec.
};
```

Together

```
void measure :: getm();
cout << " feet " << " " << inch << endl;
```

Normal

when defined outside of class.

body written outside class

Operator Overloading

Rules

(1) After overloading of any existing operator the syntax of overloaded operator remains same.

(2) Overloaded operator works on object only if operator implement in a form of function.

→ return type operator (operator symbol) (parameter)
{
}

(3) The no. of parameters less than one to the type of operator

unary → 0 → obj ++ ;
call

binary → 1
obj = obj + obj ;
call argument

Eg

```
void operator ++ () {
    inch ++ ;
    if (inch >= 12) {
        feet ++ ;  inch -= 12 ;
    }
}

void main () {
    distcr ();
    measure m1 ;
    m1.getm () ; m1 ++ ;
```

```
// void inc () {
}

m1.inc () ;
call
++ m1 ;
```

Eg a

obj call

m2 = m1.inc();

measure inc () {
measure t;
inch ++;
if (inch >= 12) {
feet ++;
inch -= 12;
}
t.feet = feet; t.inch = inch;
return t;
}

obj call

// m2 = m1++;
m2 = ++m1;

measure operator ++ ( ) {
measure t;
inch ++;
if (inch >= 12) {
feet ++; inch -= 12; }
t.feet = feet; t.inch = inch;
return t;
}

Assignment

String class

1. Overload an operator (++) on string class such that each character of string increase by 1

hello
ifmmp

2. y = -x;

y = x.reverse();

```
#Include <iostream>
#include <string.h>
using namespace std;

class strin {
        private: char a[20];
public:
        void get(){
                cin>>a;
        }
        void operator++() {
                int k, i;
        k = strlen(a);
        for(i=0; i<k; i++){
                a[i]++;
        }
        cout<<a;
        }

};
        int main() {
                strin b;
                b.get(); ~~b++;~~ ++b;
                }
```

Binary operator

~~m3 = m1.add(m2);~~

m3 = m1 + m2;

call — argument

m3 = m1.add(m2);
= m1 + m2;

measure operator + (measure x)
{
measure t;
t.feet = feet + x.feet;
t.inch = inch + x.inch;

if (t.inch >= 12) {
t.feet++;
t.inch = -12;
}
return t;
}

void main() {
measure m1, m2, m3;
m1.getm();
m2.getm();
m3 = ~~m1.add(m2)~~ m1 + m2; // m3 = m1.add(m2)
m3.showm();

For adding 3 entries

① m4 = m1 + m2;
m4 = m4 + m3;

② m4 = m1 + m2 + m3
(m1 + m2 — nameless return object)

void showm() { cout << feet << inch; }

m1.add(6.5);

→ m4 = m1 + 6.5;
call arg

True when constructor of same type is present.

Object add with primary data type (int, float, ...) .. only if class having a one argument constructor of that primary datatype

✓ Const : measure(float x) {
feet = x; inch = (x - feet) * 12;

→ m4 = 6.5 + m1;

Currently, incorrect syntax because in binary overloaded left operand always call overloaded operator so it should be always an object.

→ Logical (>, <)

```
void int operator > (measure x) {
    float d1, d2;
    d1 = feet * 12 + inch;
    d2 = x.feet * 12 + inch;
    if (d1 > d2)
        return 1;
    else
        return 0;
}
void main() {
    measure m1, m2;
    m1.getm();
    m2.getm();
    if (m1 > m2) {
        cout << "m1 largest";
        m1.showm();
    }
```

```
else if (m2 > m1) {
        cout << "m2 largest";
        m2.showm();
    }
    else {
        cout << "equal";
    }
```

bool → char → short int → int → unsigned int → long → unsigned → long long → float → double → long double

(Implicit) (Auto. by user)

Type casting

1. Pre → Pre (Pre = primary)
2. Pre → User defined (object)
3. User defined (object) → Pre
4. Object → Object

1) Predefine → Predefine.

```
int a;
float b;
{ a = b; or b = a; } → Implicit type casting
```

Syntax: `b = (float) a;` → Explicit type casting (datatype)

Syntax: `b = float(a);` → Special routine (like a function)

2) Predefine → Userdefine (Object)

Syntax: `object = var;` (var → primary datatype var.)

Method (1) By constructor

```
Main body :- measure m1 = 6.5; // measure m1(6.5);
class :- measure (float x) {
            feet = x;
            inch = (x - feet) * 12;
         }
```

② by function

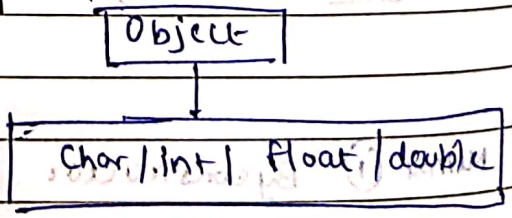
```
measure m1;
m1.assign(6.5);        // call
void assign (float x) {
    feet = x;
    inch = (x-feet) *12;
}
```

③ by operator

```
measure m1;
m1 = 6.5;              // call
   └ pass as an argument
void operator = (float x) {

}
```

3) userdefine (object) → Predefine



Main

```
measure m1 = 6.5;
float k;
k = m1.convert();
cout << k;
```

Class

```
float convert() {
    float t;
    t = feet + inch/12;
    return t;
}
```

To overload any data type as a function
(int / sec.)

```
operator datatype ()
{

}
```

Eg: class : operator float ()

```
{
    float t;
    t = feet + inch/12.0;
    return t;
}
```

```
measure m1 = 6.5;
float k;
→ k = m1; → implicit
or
→ k = float (m1);
        ↳ explicit.
```

If we overload any data type as an operator within class it means that class object assigned to overloaded data type.

4) Object → Object

destination — rupees — rs — convert — x

source — dollar — dl — convert — y

```
x.convert(y);
x = y.convert();
```

classmate

Date ______
Page ______

M1 function mode
at source class
const →
const →

```
Class rupees {
        private :
                float rs;
        public :
                rupees () {
                        rs = 0;
                }
                rupees (float r) {
                        rs = r;
                }
                void getrs () {
                        cin >> rs;
                }
                void showrs () {
                        cout << rs << endl;
                }
};
```

source
L
The value we have got initialis

rupees x ; [x: rs = 3.]

dollor y ; [y: d1 = 5]

```
Class dollor {
        private :
                float d1;
        public :
                dollor() {
                        d1 = 0;
                }
                dollor (float d) {
                        d1 = d;
                }
                void getd1 () {
                        cin >> d1;
                }
                void showd1 () {
                        cout << d1 << endl;
```

x = y.convert(); classmate Date Page

source

```
rupees convert() {
    float k;
    k = dl * 80;
    rupees t(k);        // rupees t;
    return t;           // t.setrs(k);
}
};
```

object

t → 400 rs

```
void main() {
    clrscr();
    rupees x;
    dollor y;
    y.getdl();
    x = y.convert();
    y.showdl();
    x.showrs();
    getch();
}
```

x = y.convert(); → (destination data type)

operator rupees() {

implicit → x = y; ✓

explicit → x = rupees(y); ✓

```
    float k;
    k = dl * 80;
    rupees t(k);
    return t
}
```

y.showdl();
x.showrs();
}

f(n) at desitinam

2

```
class dollor {
    private :
        float dl;
    public :
        dollor() {
            dl = 0;
        }
        dollor (float d) {
            dl = d;
        }
        void getdl() {
            cin >> dl;
        }
        void showdl () {
            cout << dl << endl;
        }
        float retdl () {
            return dl;
        }
};
```

```
class rupees {
    private :
        float rs;
    public :
        rupees() {
            rs = 0;
        }
        rupees (float r) {
            rs = r;
        }
```

Conversion function should be made at class which is defined at last.

```
void getrs () {
        cin >> rs;
    }
void show rs() {
        cout << rs << endl;
    }
/* void operator = (dollor p)*/
← void convert (dollor p) {
        float k;
        k = p.retdl();
        rs = k * 80;
    }
};

void main() {
    drser();
    rupees x;
    dollor y;
    y.getdl();
    x.convert(y);   // x = y;

    y.showdl();
    x.show rs();
    getch();
}
```

y: 5 dl.

p: 5 dl

when (x = y)

destination

↑

x = y ; .
y = x ;

} possible in one program

source

(must be defined in function with comes at last)

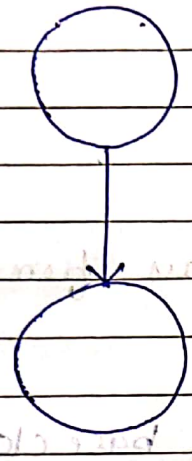
```
class rupees {
    };
class dollor {
    operator rupees () {
        x = y
    }
    void operator = (rupees p) {
        y = x
    }
}
```

OR //

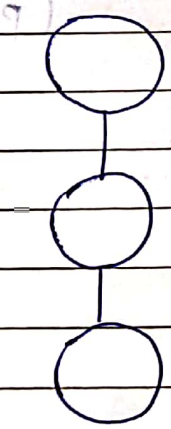
```
class dollor {
    };
class rupees {
    void operator = (dollor p) {
        x = y
    }
    operator dollor () {
        y = x
    }
};
```

Camlin | Page
Date / /

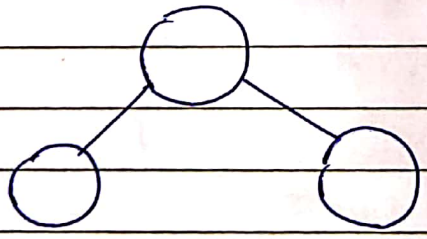
Inheritance :- How to use the feature (data & function) of one class to another class by defining a relation between the classes as a parent child.

i> 

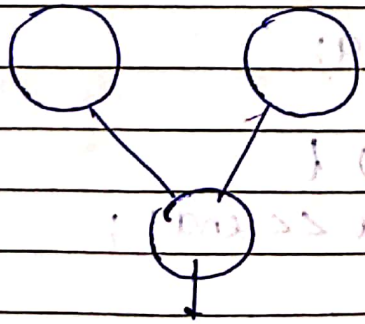
⇒ indicates single level inheritance

ii> 

⇒ indicates Multi level inheritance

iii> 

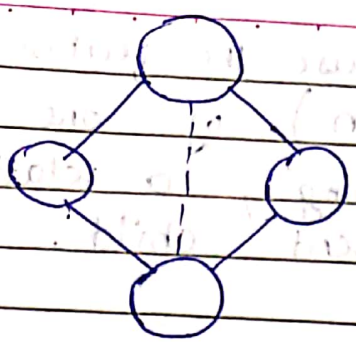
→ Hierarchical Inheritance.

iv> 

⇒ Multiple Inheritance

class having more than one parent.

uses features of two classes.

v)  Hybrid inheritance

i) Single level inheritance

Syntax to define a derived class from any existing class

```
class classname : public/private baseclass
                                  (parent)
{
}
```

Q#

```
class alpha
{
private:
    int a;
public:
    void geta() {
        cin >> a;
    }
    void showa() {
        cout << a << endl;
    }
```

```
int xeta() {
    return a;
}

class beta : public alpha
{
private:
    int b, c;
public:
    void getb() {
        cin >> b;
    }
    void showb() {
        cout << b << endl;
    }

void sum() {
    c = b + xeta(); // acc. to calling object i.e. x.
    cout << c << endl;
}
```

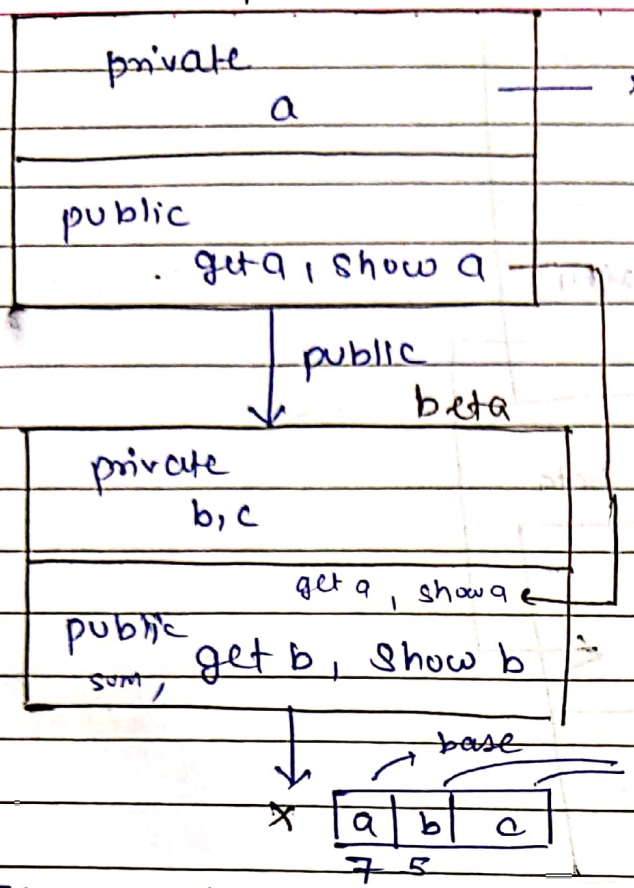
```
};

void main() {                clrscr();
        beta x;
      x.geta();
     x.getb();
    x.showa();
    x.showb();
     x.sum();
}
```

Camlin | Page
Date / /

acc to obj

Basic

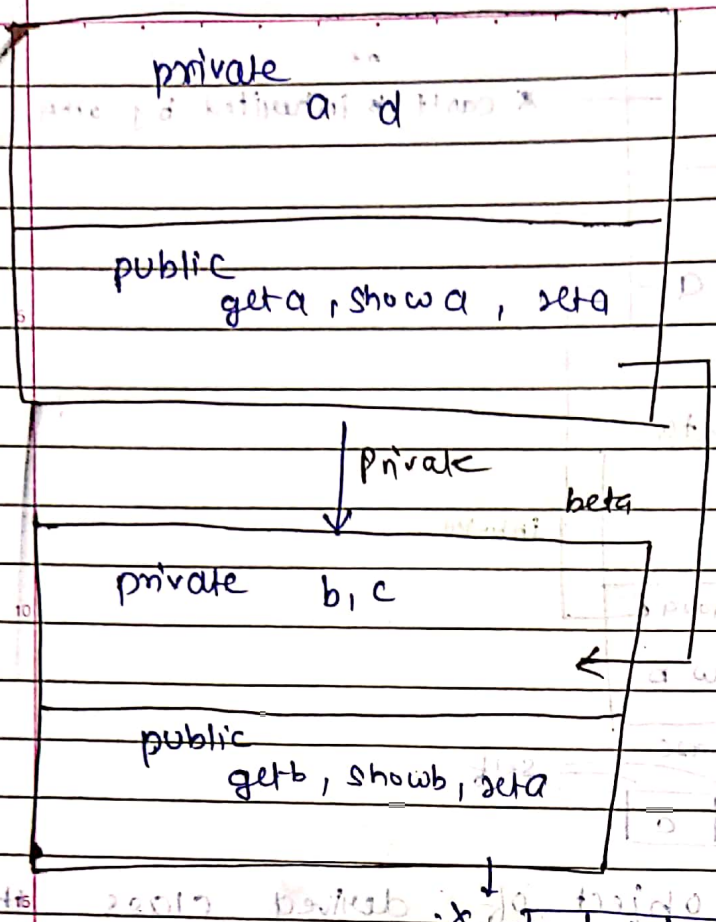


1. If we declare an object of derived class the total no. of data for derived class object is sum of data of base classes & data of own class.

2. Base class private feature never inherit by derived class.

3. All public feature of base class inherit in derived class public section.

alpha



b) ~~Same as public~~ a | b | c

Same) 1) If we

2) Private feature of base class never inherit in derived class.

3) Base class public feature inherit in derived class in private section.

removed from main

```
x.geta();
x.showa();
```

```
void getbb() {
    geta();  ✓ & added
    cin >> b;
void showb() {
    showa(); ✓ added
    cout << b << endl;
```

```
base class :: feature
// alpha :: geta();
```

private

Scanned with CamScanner

Date / /

① During compilation time its already decided, an object can call which function from which class.

That concept is called early binding.

② When derived class object call any function first it gives the preference of their own class function / feature than to their base class. // function overiding

③ → Base Class & Derived Class having a function with same Name

Special case

→ Derive class member function hides the visibility of base class inherit function.

```
class alpha ⇒
        void input () {
            cout << "base";
        }

Class beta ⇒ void input () {
            cout << " Derived \n";
        }

        x.input ();
```

Output → Derived

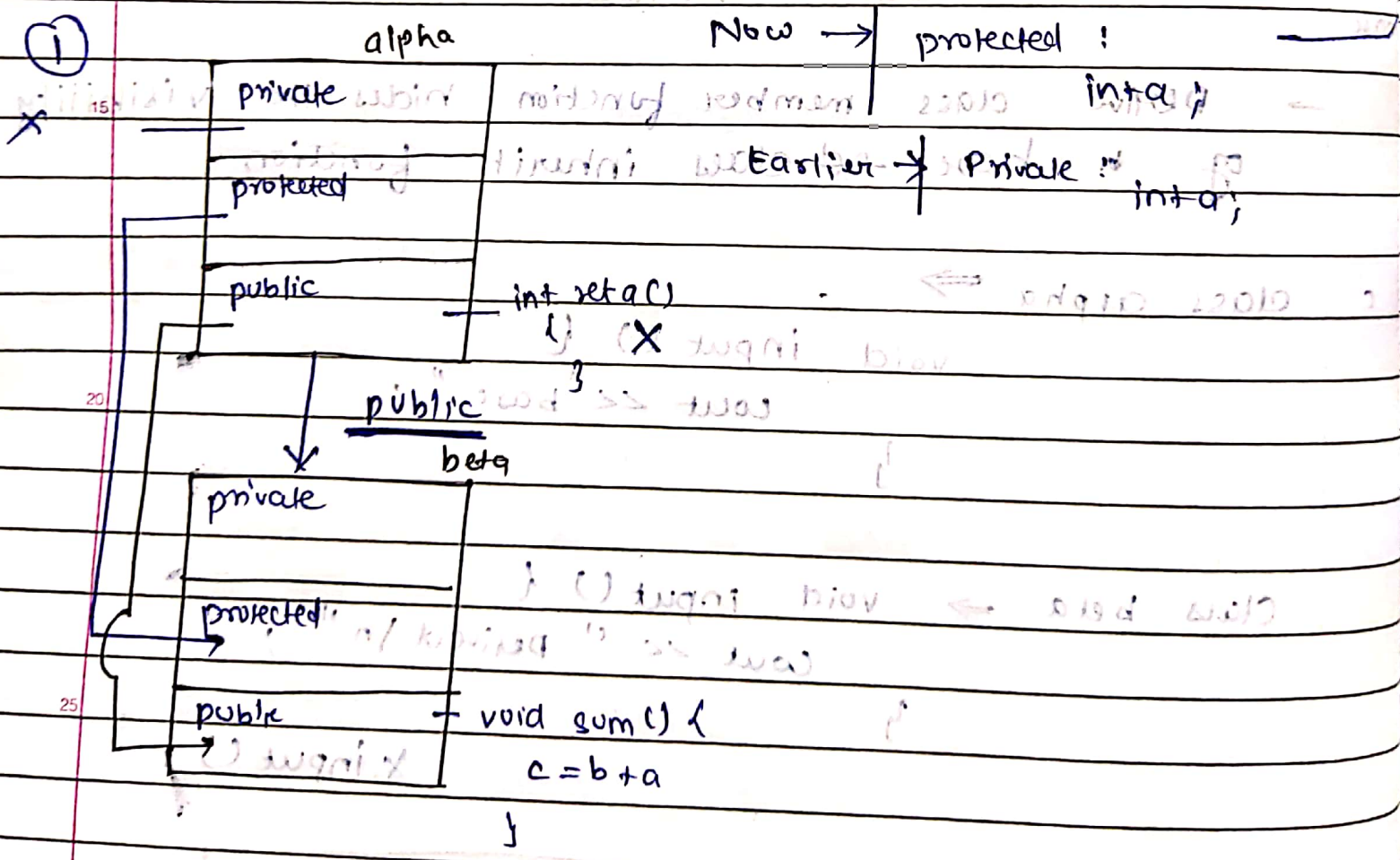
Role No. (2)

How to use base class, when name of function is same

```
void input() {                    scope resolution
    alpha::input();  ---->  // base class :: feature
    cout << "derived \n";
}
```

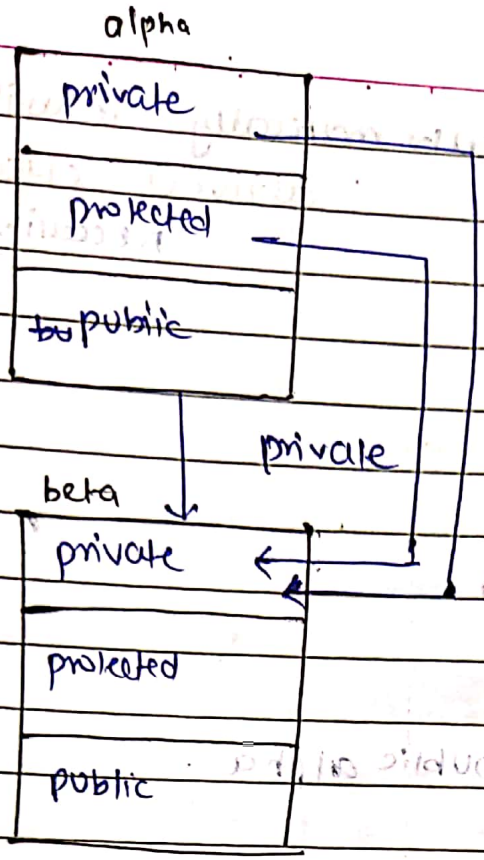
Protected :- It is a keyword and access modifier if, if, anything define under protected its use within the class as well as in their immediate derived class.

(1)



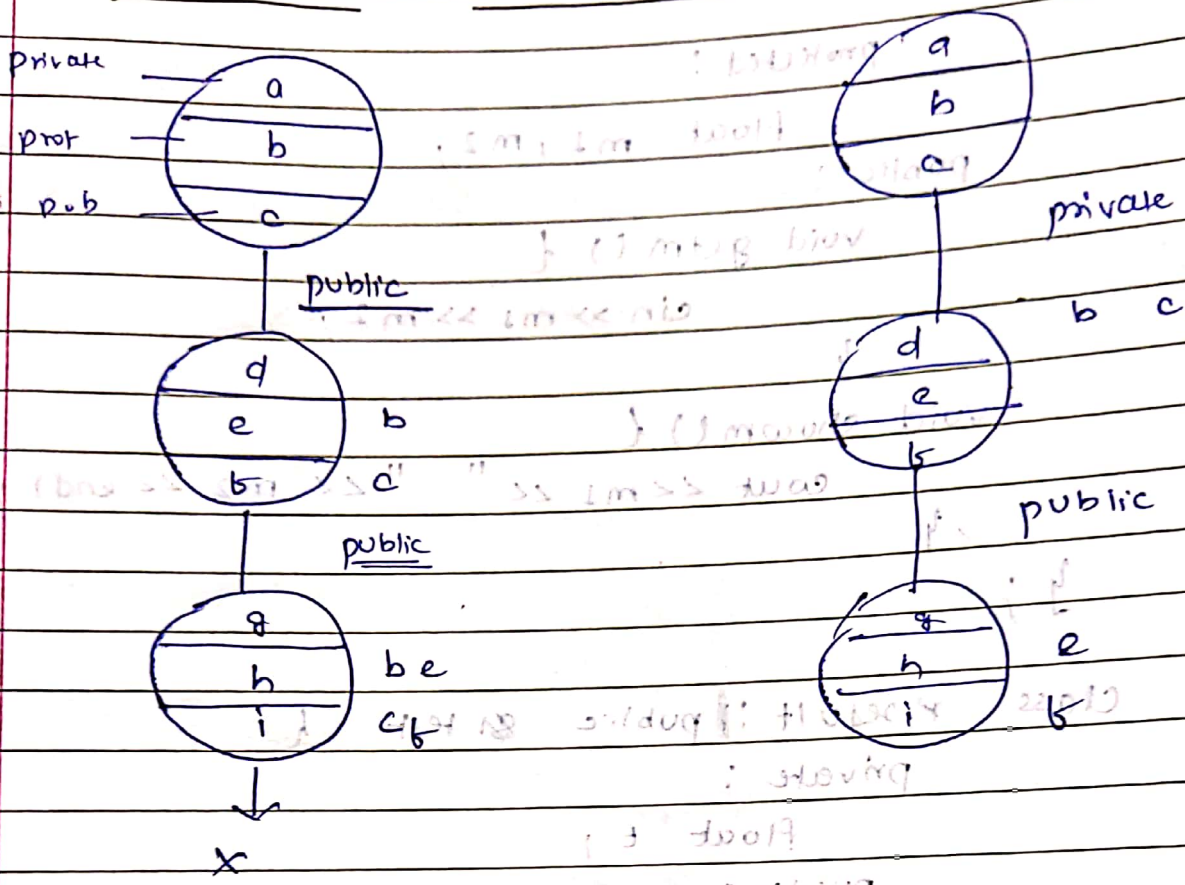
protected : ⟶ automatically inherited to derived class (no necessity to define)
int a;
public :
void geta() {
cin>>a;
};
void showa() {
cout << a << endl;
}
};
class beta : public alpha.
{
private:
int b, c;
public :
void getb() {
cin >> b;
}
void showb() {
cout << b << endl;
}
void sum() {
c = a + b;
cout << c << endl;
}
~~void x;~~ };
void main() {
clrscr();
beta x;
x.geta();
x.getb();
x.showa();

(2)



	Own class	Derived class	Outside of class (Derived class object)
private	✓	✗	✗
protected	✓	✓ (public → protected)	✗
	✓	✓ (public → public)	✗
public	✓	✓ (private → private)	✓
	✓	✓ (public → private)	✗

Multilevel Inheritance



Eg.

```
#
#
class student {
    protected :
        int rno;
        char name [10];
public :
    void getst() {
        cin >> rno >> name;
    }
    void showst(){
        cout << rno << " " << name << endl;
    }
};
```

```
class test : public student {
    protected :
        float m1, m2;
public :
    void getm () {
        cin >> m1 >> m2;
    }
void showm () {
    cout << m1 << "  " << m2 << endl;
}
};
class result : public test {
    private :
        float t;
    public :
    void total () {
        t = m1 + m2;
        cout << t << endl;
    }
};

void main () {
    clrscr();
    result x;
    x.getst ();
    x.getm ();
    x.showst ();
    x.showm ();
    x.total ();

    getch();
```

private public private

public private private

Camlin Page
Date / /

Assignment Change the relation between different classes and write code.

object.baseclass :: feature

x.test :: show(); // if show function is present in both base classes i.e

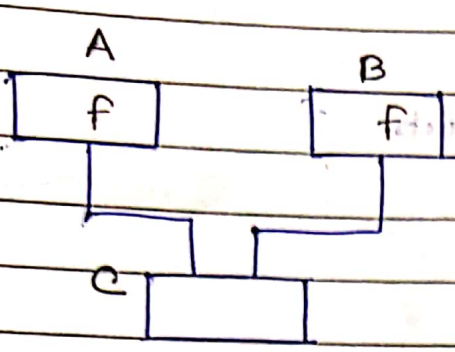
private private

```
#include <iostream>
using namespace std;
class student {
        private :
                int rno;
                char name[10];
        public :
                void getst() { cin >> rno >> name ;
                }
                void showst() {
                        cout << rno << " " << name << endl;
                }
};

class test : private student {
                private:
                        float m1, m2;
        public :
                void getm() {
                        cin >> m1 >> m2 ;
                }
                void showm() {
                        cout << m1 << " " << m2 ;
                }
void a1() { getst(); }
void a2() { showst(); }
```

```
    }
int retm2(){
        return m2;
    }
};
class result{ private test{
    private:
        float t;
public:
    void p1(){
        a1();
        getm();
    }
void p2(){ a2();
        show();
    }
void total(){
        t = retm1() + retm2();
    cout << t << endl;
    }
};
int main(){
    result x;
    x.p1();
    x.p2();
    x.total();
}
```

Multiple Inheritance



f → name of function

object of C class

```
x.f();
x.A::f();
x.B::f();
```

```
class A {
};
class B {
};
class C : public A, public B
{
}
```

BASE CLASS HAVING FEATURE WITH SAME NAME BUT NOT PRESENT IN DERIVED CLASS

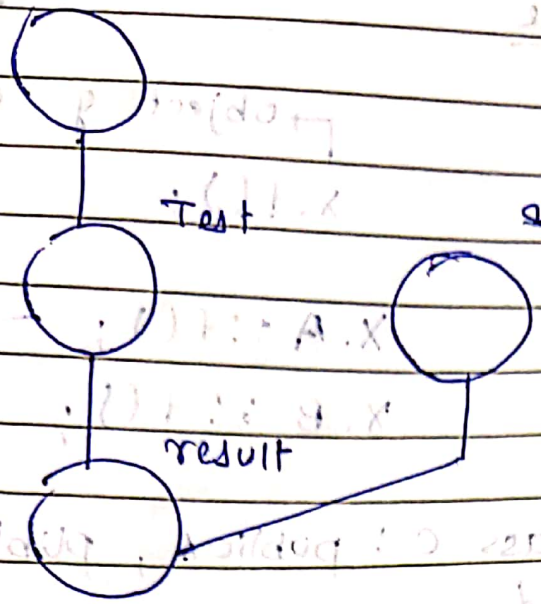
#

```
class student {
protected:
int rno;
char name;
```

⇒

Camlin Page
Date / /

No change
same as multilevel)




```
class student {
        same as multilevel example
};
class test : public student {
};
class sports {
protected :
        float sp;
public :
    void getsp () {
        cin >> sp;
    }
void showsp () {
        cout << sp << endl;
}
};
```

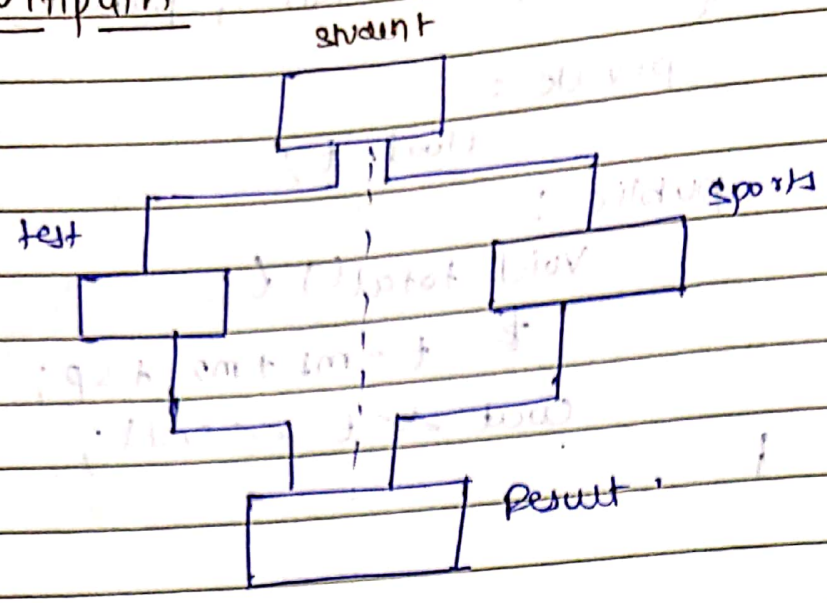
```
class result : public test , public sports
{
    private:
        float t;
    public :
        void total() {
            t = m1 + m2 + sp;
            cout << t << endl;
        }
};

void main() {
    clrscr();
    result x;
    x.getst();
    x.getm();
    x.getsp();
    x.showst();
    x.showm();
    x.showsp();
    x.total();
    getch();
}
```

object.base class::fature()

x.test::show(); // if show function is present in both the base classes i.e test & sports.

Multipath




```
class student {

};
class test : public student
{

}
class sports : public student
(

};
class result : public test, public sports
{

};
```

virtual

virtual

can be used at any of two posn i.e before public or after public.

The top base class feature present in multiple copies for bottom derived class.

ambiguity

To know over

Camlin Page
Date / /

* All intermediate class should behave as single class for the relation of top base class with bottom derived class.

This can be achieved through a Virtual Base class.

→ Add keyword virtual (before or after public/private) in intermediate class.

Constructor and Inheritance

Case (1) No constructor present in base class & derived class.

↓ then

If we declare an object of derived class its called default constructor of derived class & default constructor of base class.

Case (2) - Constructor present in 'base' class only.

↓ then

If we declare an object of derived class it's called default constructor of derived class and two & argument constructor of base class.

⇒ Done

Camlin | Page
Date / /

Q6a

#
#

```
conio
class alpha {
    private : int a;
public :
    alpha () {
        cout << " 0 alpha \n";
        a = 0;
    }
    alpha (int a1) {
        cout << 1 " alpha \n";
        a = a1;
    }
};

class beta : public alpha {
    private : int b;
};

void main() {
    clrscr();
    beta x;
    getch();
}
```

if not present gives error.

Output :-

0 alpha.

Camlin | Page
Date / /

Case (3)

* Constructor present only in derived class

If we declare an object of derived class call specific constructor of derived class and default constructor of base class

#

```
class alpha {
    private : int a;
};
class beta : public alpha {
    private:
        int b;
    public :
        beta() {
            b = 0;
            cout << " 0 beta \n";
        }
        beta (int bt) {
            b = bt;
            cout << " 1 beta \n";
        }
};

void main() {
    clrscr();
    beta x, y(10);
    getch();
}
```

Output:

0 beta
1 beta.

Date / /

Case (4) Constructor present in both class.

a) If we declare an object of derived class its call specific constructor of derived class and zero argument constructor of base class

b) Base class constructor execute first.

Code

Case (2) alpha class

Case (3) beta class.

```
void main()
{
    clrscr();
    beta x, y(10);
}
```

Output

0 alpha
0 beta
10 alpha
1 beta

b) To call specific constructor of base class

Code

case (2) base class;

```
class beta : public alpha
{
private:
    int b;
```

We can call constructor forcely / explicitly in inheritance.

```
public :
    beta() : alpha(10)
        {
        b = 0;
        cout << " 0 beta \n" ;
    }
    beta (int b1) {
        b = b1 ;
        cout << " 1 beta \n" ;
    }
};
void main() {
    clrscr();
    beta x,y (10);
    getch();
    }
```

```
beta() : alpha()
    { alpha(10)
}
}
beta(int b1) : alpha ()
    { alpha (20)
    alpha ( b1 )
}
beta (int p, int q) : alpha(p)
{
    b = q ;
}
```

pointer
Structure pointer
dynamic memory a
file handling.