**Practical-1**

**Aim:** Demonstrate the usage of Constructor and Destructor.

**i)** Define a class **data** with data member **acct\_no, balance** containing constructor

**data** to initialize data member and a member function **display** for output.

**Algorithm:**

Step 1: Start

Step 2: Call function data( ) for object D1

Step 3: Call function data( ) for object D2

Step 4: Call function data( ) for object D3

Step 5: Call function display( ) for object D1

Step 6: Call function display( ) for object D2

Step 7: Call function display( ) for object D3

Step 8: Stop

Function data ( )

Step 1: Read acct\_no, balance

Step 2: Return control to the calling function

Function display( )

Step 1: Display acct\_no, balance

Step 2: Return control to the calling function

**Flowchart:**

**Class:** data

**Data Members:**

acct\_no

balance

**Member Functions:**

data(long int a, long int b)

void display( )

Function data ( ) Function display( )

Read acct\_no, balance

Read acct\_no, balance

Call function display( ) for object D3

Call function display( ) for object D2

Call function display( ) for object D1

Call function data( ) for object D3

Call function data( ) for object D2

Call function data( ) for object D1

**Practical-2**

**Aim:** Program to demonstrate usage of a constructor and Destructor function. Declare a

class with public data member count. The class containing one constructor and

destructor to maintain updated information about active objects i.e.

1. No of objects created.
2. ii) No of objects Destroyed.

**Algorithm:**

Step 1: Start

Step 2: count=0

Step 3: Call function alpha( ) for object A

Step 4: Call function alpha( ) for object B

Step 5: Call function alpha( ) for object C

Step 6: Call function alpha( ) for object D

Step 7: Call function alpha( ) for object E

Step 8: Call function ~alpha( ) for object E

Step 9: Call function ~alpha( ) for object D

Step 10: Call function ~alpha( ) for object C

Step 11: Call function ~alpha( ) for object B

Step 12: Call function ~alpha( ) for object A

Step 13: Stop

Function alpha( )

Step 1: count=count+1

Step 2: Print count

Step 3: Return control to the calling function

Function display( )

Step 1: Print count

Step 2: count=count-1

Step 3: Return control to the calling function

**Flowchart:**

**Class:** alpha

**Data Members:**

count

**Member Functions:**

alpha( )

~alpha( )

Function alpha( ) Function ~alpha( )

Print count

count=count+1

Print count

count=count-1

count=0

Call function alpha( ) for object A

Call function alpha( ) for object E

Call function alpha( ) for object D

Call function alpha( ) for object C

Call function alpha( ) for object B

Call function ~alpha( ) for object E

Call function ~alpha( ) for object A

Call function ~alpha( ) for object B

Call function ~alpha( ) for object D

Call function ~alpha( ) for object C

**Practical-3**

**Aim:** Program to accept the distance between city **1st & 2nd**, city **2nd & 3rd**. calculate the

distance between city **1st & 3rd**. Define a class **road** with private data member **d1, d2, d3** containing member function getdata to accept values of **d1, d2** and **calculate** for

calculating distance.

**Algorithm:**

Step 1: Start

Step 2: Call function data getdata( ) for R

Step 3: Call function calculate( ) for R

Step 4: Call function putdata( ) for R

Step 5: Stop

Function getdata ( )

Step 1: Read d1, d2

Step 2: Return control to the calling function

Function calculate( )

Step 1: d3=d1+d2

Ste[ 2: Return control to the calling function

Function putdata( )

Step 1: Display d1, d2, d3

Step 2: Return control to the calling function

**Flowchart:**

**Class:** road

**Data Members:**

d1, d2, d3

**Member Functions:**

getdata( )

calculate( )

putdata( )

Function getdata( ) Function calculate( ) Function putdata( )

Print d1, d2, d3

Read d1, d2

d3=d1+d2

Call function data getdata( ) for R

Call function calculate( ) for R

Call function putdata( ) for R

**Practical-4**

**Aim:** Demonstrate the use of operators overloading (string manipulation: + for concatenation

and relational operators for alphabetical comparison).

**Algorithm:**

Step 1: Start

Step 2: Call function input( ) for S1

Step 3: Call function input( ) for S2

Step 4: Call function operator>(S2) for S1

Step 5: Call function operator+(S2) for S1

Step 6: Call function disp( ) for S3

Step 7: Stop

Function input ( )

Step 1: Read str

Step 2: Return control to the calling function

Function display( )

Step 1: Print str

Ste[ 2: Return control to the calling function

Function operator+( )

Step 1: temp.str=str+c.str

Step 2: Return control to the calling function

Function operator>( )

Step 1: if str>c.str

Yes a) Return 1 to the calling function

else

No b) Return 0 to the calling function

**Flowchart:**

**Class:** string

**Data Members:**

Str

**Member Functions:**

input( )

disp( )

operator+( )

operator>( )

Function input( ) Function disp( )

Print str

Read str

Function operator+( ) Function operator> ( )

Yes

If str > c.str ?

temp.str=str

Return 1

temp.str=temp.str+c.str

No

Return 0

Return temp to the calling procedure

Call function input( ) for S1

Call function input( ) for S2

Call function operator>(S2) for S1

Call function operator+(S2) for S1

Call function disp( ) for S3

**Practical-5**

**Aim:** In a bank N depositor deposit the amount, write a program to find total amount

deposited in the bank. Declare a class deposit with private data member **Rupee** and

**Paisa** containing member function **getdata, putdata.**

**i) Use array of objects**

**ii) Use Operator ‘+’ overloading.**

**Algorithm:**

Step 1: Start

Step 2: Call function deposit( ) for D[0] to D[9] Each

Step 3: Call function deposit( ) for SUM

Step 4: i=0

Step 5: if i<=9

Yes a) Call function getdata( ) for D[i]

b) i=i+1

c) goto step 5

No d) goto step 6

Step 6: i=0

Step 7: if i<=9

Yes a) Call function putdata( ) for D[i]

b) Call function operator+(D[i] ) for SUM

c) i=i+1

d) goto step 7

No e) goto step 8

Step 8: Call function putdata( ) for SUM

Step 9: Stop

Function deposit( )

Step1: Rupee=0, Paisa=0

Step2: Return control to the calling function

Function getdata ( )

Step 1: Read Rupee, Paisa

Step 2: Return control to the calling function

Function putdata( )

Step 1: Print Rupee, Paisa

Ste[ 2: Return control to the calling function

Function operator+( )

Step 1: temp.Paisa=Paisa+c.Paisa

Step2: if (temp.Paisa>=100)

Yes i) temp.Rupee=temp.Paisa/100

ii) temp.Paisa=Remainder of (temp.Paisa÷100)

iii) goto Step 3

Step 3: temp.Rupee=temp.Rupee+Rupee+c.Rupee

Step 4: Return temp to the calling function

**Flowchart:**

**Class:** deposit

**Data Members:**

Rupee, Paisa

**Member Functions:**

deposit( )

getdata( )

putdata( )

operator+( )

Function deposit( ) Function getdata( )

Read Rupee, Paisa

Rupee=0 , Paisa=0

Function putdata( ) Function operator+( )

temp.Paisa=Paisa+c.Paisa

Print Rupee, Paisa

Yes

If

temp.paisa>=100?

temp.Rupee=temp.Paisa/100

temp.Paisa=temp.Paisa%100

temp.Rupee=Rupee+c.Rupee

Return temp to the calling procedure

No

Call function putdata( ) for SUM

Call function deposit( ) for D[0] to D[9] Each

Call function deposit( ) for SUM

Call function getdata( ) for D[i]

i=0

If i<=9

?

Yes

No

i=0

Call function putdata( ) for D[i]

If i<=9

?

Yes

No

i=i+1

Call function operator+(D[i]) for SUM

i=i+1

**Practical-6**

**Aim:** Declare class event and accept time of first event and second event and find the

difference between 1st and 2nd event. Containing public member function **getdata** and

**display** with private data member **hour**, **minute, second and total.**

**i) Use Operator ‘-‘ overloading.**

**Algorithm:**

Step 1: Start

Step 2: Call function gettime( ) for E1

Step 3: Call function gettime( ) for E2

Step 4: E3=Call function operator-(E2) for E1

Step 5: Call function puttime( ) for E3

Step 6: Stop

Function gettime ( )

Step 1: Read hour, minute, second

Step 2: Return control to the calling function

Function operator-( )

Step 1: temp.second=second-c.second

Step 2: if temp.second<0?

Yes i) minute=minute-1

ii) temp.second=60+temp.second

iii) goto step 3

Step 3: temp.minute=minute-c.minute

Step 4: if temp.minute<0?

Yes i) hour=hour-1

ii) temp.minute=60+temp.minute

iii) goto step 5

Step 5: temp.hour=hour-c.hour

Step 6: Return temp and transfer the control to the calling function

Function putdata( )

Step 1: Print hour, minute, second

Step 2: Return control to the calling function

**Flowchart:**

**Class:** road

**Data Members:**

hour, minute, second

**Member Functions:**

gettime( )

puttime( )

operator-( )

Function gettime( )

Function gettime( ) Function puttime( )

Print hour, minute, second

Read hour, minute, second

Function operator-( )

temp.second=second-c.second

Yes

If temp.second<0 ?

minute=minute-1

No

temp.second=60+temp.second

temp.minute=minute-c.minute

No

If temp.minute<0 ?

Yes

hour=hour-1

temp.minute=60+temp.minute

temp.hour=hour-c.hour

Return temp

Call function data gettime( ) for E1

Call function data gettime( ) for E2

E3=Call function operator-(E2) for E1

Call function puttime( ) for E3

**Practical-7**

**Aim:** Program to demonstrate **Single Inheritance**. Declare a class **B** and derive publically

class **D** from **B**.

**i)** The class **B** contains private data member **a**, public data member **b** with member

function **get\_ab, get\_a, show\_a.**

**ii)** The derived class **D** contains data member **c** with member function **mul** and **display.**

**Algorithm:**

Step 1: Start

Step 2: Call function get\_ab( ) for object

Step 3: Call function mul( ) for object

Step 4: Call function display( ) for object

Step 5: Stop

Function get\_ab ( )

Step 1: Read a,b

Step 2: Return control to the calling function

Function get\_a( )

Step 1: Return a along with the control to the calling function

Function show\_a( )

Step 1: Print a

Step 2: Return control to the calling function

Function mul( )

Step 1: c=call to function get\_a( )\*b

Step 2: Return control to the calling function

Function display( )

Step 1: Call to function show\_a( )

Step 2: Print b, c

Step 3: Return control to the calling function

**Flowchart:**

**Class:** B

**Data Members:**

a, b

**Member Functions:**

get\_ab( )

get\_a( )

show\_a( )

**Class:** D: public B

**Data Members:**

c

**Member Functions:**

mul( )

display( )

Function B::get\_ab( ) Function B::get\_a( ) Function B::show\_a( )

Return a

Read a,b

Print a

Function D:: mul( ) Function D:: display( )

c=Call function get\_a( )\*b

c=Call function get\_a( )\*b

Print b,c

Call function get\_ab( ) for **object**

Call function mul( ) for **object**

Call function display( ) for **object**

**Practical-8**

**Aim:** Program to demonstrate **Multiple Inheritances. Declare class M** and **N** and derive

publically class **P** from **M** and **N.**

**i)** Declare a class **M** with protected data member **m** and public member function

**get\_m.**

**ii)** Declare a class **N** with protected data member **n** containing member function

**get\_n.**

**iii)** Declare class **P** containing member function **display.**

**Algorithm:**

Step 1: Start

Step 2: Call function display( ) for object

Step 3: Stop

Function get\_m ( )

Step 1: Read m

Step 2: Return control to the calling function

Function get\_n( )

Step 1: Read n

Step 2: Return the control to the calling function

Function display( )

Step 1: Call to function get\_m( )

Step 2: Call to function get\_n( )

Step 3: Print m,n

Step 4: Return control to the calling function

**Flowchart:**

**Class:** M

**Data Members:**

m

**Member Functions:**

get\_m( )

**Class:** N

**Data Members:**

n

**Member Functions:**

get\_n( )

**Class:** P: public M, public N

**Member Functions:**

display( )

Function M :: get\_m( ) Function N::get\_n( ) Function P::display( )

Print m,n

Call function get\_m( )

Call function get\_n( )

Read n

Read m

Call function get\_ab( ) for **object**

**Practical-9**

**Aim:** Program to demonstrate Multilevel Inheritance. Declare a class **student** and derive

publically a class **test** and derive publically class **result** from class **test.**

**i)** The class student contains protected data member **roll\_number** with public

member functions **get\_number** and **put\_number.**

**ii)** The class **test** containing protected data member **sub1, sub2** with public member function **get\_marks** and **put\_marks.**

**iii)** The class **result** contains data member **total** and public member function **display.**

**Algorithm:**

Step 1: Start

Step 2: Call function get\_number( ) for R

Step 3: Call function get\_marks( ) for R

Step 4: Call function display( ) for R

Step 5: Stop

Function get\_number ( )

Step 1: Read roll\_number

Step 2: Return control to the calling function

Function put\_number( )

Step 1: Print roll\_number

Step 2: Return the control to the calling function

Function get\_marks ( )

Step 1: Read sub1, sub2

Step 2: Return control to the calling function

Function put\_marks( )

Step 1: Print sub1, sub2

Step 2: Return the control to the calling function

Function display( )

Step 1: total=sub1+sub2

Step 2: Call to function put\_number( )

Step 3: Call to function put\_marks( )

Step 4: Print total

Step 5: Return control to the calling function

**Flowchart:**

**Class:** student

**Data Members:**

roll\_number

**Member Functions:**

get\_number( )

put\_number( )

**Class:** test: public student

**Data Members:**

sub1, sub2

**Member Functions:**

get\_marks( )

put\_marks( )

**Class:** result: public test

**Data Members:**

total

**Member Functions:**

display( )

Function student :: get\_number( ) Function student :: put\_number( )

Print roll\_number

Read roll\_number

Function test :: get\_marks( ) Function test :: put\_marks( )

Print sub1, sub2

Read sub1,sub2

Function result :: display( )

total=sub1+sub2

Call function put\_number ( )

Call function put\_marks ( )

Print total

Call function get\_number( ) for R

Call function get\_marks( ) for R

Call function display( ) for R

**Practical-10**

**Aim:** Program to demonstrate Hierarchical Inheritance. Declare a class **Side** and derive

publically class **Square** from base class **side** and also derive publically class **cube**

from base class **side.**

**i)** Class **Side** contains protected data member **L** with a member function **set\_values.**

**ii)** Class **Square** contains member function **sq.**

**iii)** Class **Cube** contains member function **cub.**

**Algorithm:**

Step 1: Start

Step 2: Call function set\_values( ) for object1

Step 3: Call function sq( ) for object1

Step 4: Call function set\_values( ) for object2

Step 5: Call function cub( ) for object2

Step 6: Stop

Function set\_values ( )

Step 1: Read L

Step 2: Return control to the calling function

Function sq( )

Step 1: Print L\*L

Step 2: Return the control to the calling function

Function cub( )

Step 1: Print L\*L\*L

Step 2: Return control to the calling function

**Flowchart:**

**Class:** Side

**Data Members:**

L

**Member Functions:**

Set\_values( )

**Class:** Sphere: public Side

**Member Functions:**

sq ( )

**Class:** Cube: public Side

**Member Functions:**

cub( )

Function side :: set\_values( ) Function sphere :: sq( )

Print L\*L

Read L

Function cube :: cub( )

Print L\*L\*L

Call function set\_values( ) for object1

Call function sq( ) for object1

Call function set\_values( ) for object2

Call function cub( ) for object1

**Practical-11**

**Aim:** Program to demonstrate usage of normal virtual function and pure virtual Function

with abstract class.

**Algorithm:**

Step 1: Start

Step 2: Let bptr point object B

Step 3: Call function message( ) using bptr

Step 4: Call function show( ) using bptr

Step 5: Call function disp( ) using bptr

Step 6: Stop

Function base::message( )

Step 1: Print “This is base class message”

Step 2: Return control to the calling function

Function base::show( )

Step 1: Print “Show Base”

Step 2: Return the control to the calling function

Function base::disp( )

Step 1: Return control to the calling function

Function deriv::message( )

Step 1: Print “This is derived class message”

Step 2: Return control to the calling function

Function deriv::show( )

Step 1: Print “Show Derived”

Step 2: Return the control to the calling function

Function deriv::disp( )

Step 1: Print “Display Derived”

Step 2: Return control to the calling function

**Flowchart:**

**Class:** Base

**Member Functions:**

message( )

virtual show( )  
virtual disp( )

**Class:** Deriv

**Member Functions:**

message( )

show( )  
disp( )

Function base :: message( ) Function base :: show( )

Print “Show Base”

Print “This is base calss message”

Function base :: disp( )

Empty Function

Function deriv :: message( ) Function deriv :: show( )

Print “This is derived class message”

Print “Show Derived”

Function deriv :: disp( )

Print “Display Derived”

Let bptr point object D

Call function message( ) using bptr

Call function show( ) using bptr

Call function disp( ) using bptr