Classes and objects :class discussion programs

1. Program on Creation of simple class structure(Single student)

```
#include <iostream>
using namespace std;

class Student
{
  public:
     int id;//data member (also instance variable)
     string name;//data member(also instance variable)
};
int main()
{
   Student s1; //creating an object of Student
   s1.id = 201;
   s1.name = "Lahari";
   cout<<s1.id<<endl;
   cout<<s1.name<<endl;
   return 0;
}
```

2. Initialize and Display data through methods.

```
#include <iostream>
using namespace std;

class Student
{
  public:
     int id;//data member (also instance variable)
     string name;//data member(also instance variable)

     void insert(int i, string n)
      {
         id = i;
         name = n;
      }

     void display()
      {
         cout<<id<<" "<<name<<endl;
      }
};

int main()
{
   Student s1; //creating an object of Student
   Student s2; //creating an object of Student
   s1.insert(201, "Sonoo");
   s2.insert(202, "Nakul");
   s1.display();
   s2.display();
   return 0;
}
```

Class Activity1:

1. Nihal is trying to calculate the area and volume of the rectangular room help Nihal to write a cpp program to calculate the roomarea and volume with the help of valid data members and member functions.

Program:

```
// Program to illustrate the working of
// objects and class in C++ Programming

#include <iostream>
```

```
using namespace std;

// create a class
class Room {

  public:
   double length;
   double breadth;
   double height;

   double calculateArea() {
      return length * breadth;
   }

   double calculateVolume() {
      return length * breadth * height;
   }
};

int main() {

   // create object of Room class
   Room room1;

   // assign values to data members
   room1.length = 42.5;
   room1.breadth = 30.8;
   room1.height = 19.2;

      // calculate and display the area and volume of the room
       cout << "Area of Room =  " << room1.calculateArea() << endl;
      cout << "Volume of Room =  " << room1.calculateVolume() << endl;

   return 0;
        }
```

Assignment questions

1. 1. C++ program to read time in HH:MM:SS format and convert into total seconds using class.

2. Define a class to represent a bank account. Include the following members:

Data members

1. Name of the depositor
2. Account number
3. Type of account
4. Balance amount in the account

Member functions

1. To assign initial values
2. To deposit an amount

	3. To withdraw an amount after checking the balance 4. To display name and balance Write a main program to test the program.
3. Modify the program 2 for handling 10 customers.	4. Write a class to represent a vector (a series of float values).Include member functions to perform the following tasks. a) To create the vector b) To modify the value of a given element c) To multiply by a scalar value d) To display the vector in the form (10, 20, 30,…) Write a program to test your class.

Chapter2:class discussion programs	

1. Scope resolution operator in C++
#include <iostream>
using namespace std;
 	
char c = 'a'; // global variable (accessible to all functions)
 	
int main()
{
  char c = 'b'; // local variable (accessible only in main function)
 	
  cout << "Local variable: " << c << "\n";
  cout << "Global variable: " << ::c << "\n"; // Using scope resolution operator
 	
  return 0;
  }
	2. Scope resolution operator in C++
	#include <iostream>
	using namespace std;
	class Game
	{
	  public:
	  void play(); // Function declaration
	};
	
	// function definition outside the class
	
	void Game::play()
	 {
	  cout << "Function defined outside the class.\n";
	 }
	
	int main()
	{
	  Game g;
	  g.play();
	
	  return 0;
	}

3. // C++ program to	**4. C++ program to demonstrate**

demonstrate public

```
// access modifier

#include<iostream>
using namespace std;

// class definition
class Circle
{
	public:
		double radius;

		double compute_area()
		{
			return
3.14*radius*radius;
		}

};

// main function
int main()
{
	Circle obj;

	// accessing public datamember
outside class
	obj.radius = 5.5;

	cout << "Radius is: " <<
obj.radius << "\n";
	cout << "Area is: " <<
obj.compute_area();
	return 0;
}
```

private

```
// access modifier

#include<iostream>
using namespace std;

class Circle
{
	// private data member
	private:
		double radius;

	// public member function
	public:
		double compute_area()
		{ // member function can
access private
			// data member
radius
			return
3.14*radius*radius;
		}

};

// main function
int main()
{
	// creating object of the class
	Circle obj;

	// trying to access private data
member
	// directly outside the class
	obj.radius = 1.5;

	cout << "Area is:" <<
obj.compute_area();
	return 0;
}
```

5. C++ program to demonstrate private access modifier

```
#include<iostream>
using namespace std;

class Circle
{
	// private data member
	private:
		double radius;

	// public member function
```

7. C++ program to demonstrate protected access modifier

```
#include <bits/stdc++.h>
using namespace std;

// base class
class Parent
{
	// protected data members
	protected:
	int id_protected;

};
```

```
        public:
                void compute_area(double
r)
                { // member function can
access private
                        // data member
radius
                        radius = r;

                        double area =
3.14*radius*radius;

                        cout << "Radius is:
" << radius << endl;
                        cout << "Area is: "
<< area;
                }

};

// main function
int main()
{
        // creating object of the class
        Circle obj;

        // trying to access private data
member
        // directly outside the class
        obj.compute_area(1.5);
        return 0;
}
```

```
// sub class or derived class
class Child : public Parent
{


        public:
        void setId(int id)
        {

                // Child class is able to
access the inherited
                // protected data
members of base class

                id_protected = id;

        }

        void displayId()
        {
                cout << "id_protected is: "
<< id_protected << endl;
        }
};

// main function
int main() {

        Child obj1;

        // member function of the
derived class can
        // access the protected data
members of the base class

        obj1.setId(81);
        obj1.displayId();
        return 0;
}
```

Class Activity

1. Wrt a cpp program to create a class KLETECH employee with private data member as a salary and public member functions as getsalary and setsalary. Read the salary at compile time and print.

```
#include <iostream>
using namespace std;

class Employee {
  private:
    // Private attribute
    int salary;
```

```
  public:
    // Setter
    void setSalary(int s) {
      salary = s;
    }
    // Getter
    int getSalary() {
      return salary;
    }
};

int main() {
  Employee myObj;
  myObj.setSalary(50000);
  cout << myObj.getSalary();
  return 0;
}
```

Class Conduction programs

1. Default Constructor

```
#include <iostream>
using namespace std;
class Employee
 {
   public:
       Employee()
       {
          cout<<"Default Constructor Invoked"<<endl;
       }
};
int main(void)
{
   Employee e1; //creating an object of Employee
   Employee e2;
   return 0;
}
```

2.Parameterised constructor

```
#include <iostream>
using namespace std;
class Employee {
  public:
     int id;//data member (also instance variable)
     string name;//data member(also instance variable)
     float salary;
     Employee(int i, string n, float s)
      {
         id = i;
         name = n;
         salary = s;
      }
     void display()
      {
         cout<<id<<" "<<name<<"  "<<salary<<endl;
      }
};
int main(void) {
   Employee e1 =Employee(101, "Sonoo", 890000); //creating an object of Employee
   Employee e2=Employee(102, "Nakul", 59000);
   e1.display();
   e2.display();
   return 0;
```

	} **Output:** 101 Sonoo 890000 102 Nakul 59000

3.copy
#include <iostream>
using namespace std;
class A
{
public:
int x;
A(int a) // parameterized const ructor.
{
x=a;
}
A(A &i) // copy constructor
{
x = i.x;
}
};
int main()
{
A a1(20); // Calling the paramete rized constructor.
A a2(a1); // Calling the copy con structor.
cout<<a2.x;
return 0;
}
	4. Destructors
	#include <iostream>
	using namespace std;
	class Employee
	{
	public:
	Employee()
	{
	cout<<"Constructor Invoked "<<endl;
	}
	~Employee()
	{
	cout<<"Destructor Invoked" <<endl;
	}
	};
	int main(void)
	{
	Employee e1; //creating an object of Employee
	Employee e2; //creating an object of Employee
	return 0;
	}

4. Nested classes: nesting of member functions
 	
using namespace std;
class set
{
int m,n;
public:
void input();
void display();
int largest();
};
int set :: largest()
{
if(m >= n)
return(m);
else
return(n);
}
void set :: input()	

```
{
cout << "Input value of m and n"<<"\n";
cin >> m>>n;
}
void set :: display()
{
cout << "largest value=" << largest() <<"\
n";
}

int main()
{
set A;
A.input();
A.display();

return 0;
}
```

Class Activity

Static data members

```
#include <iostream>
#include<string.h>

using namespace std;
class Student {
  private:
  int rollNo;
  char name[10];
  int marks;
  public:
  static int objectCount;
  Student() {
    objectCount++;
  }

  void getdata() {
    cout << "Enter roll number: "<<endl;
    cin >> rollNo;
    cout << "Enter name: "<<endl;
    cin >> name;
    cout << "Enter marks: "<<endl;
    cin >> marks;
  }

  void putdata() {
    cout<<"Roll Number = "<< rollNo <<endl;
    cout<<"Name = "<< name <<endl;
    cout<<"Marks = "<< marks <<endl;
```

```
    cout<<endl;
  }
};
int Student::objectCount = 0;
int main(void) {
  Student s1;
  s1.getdata();
  s1.putdata();
  Student s2;

  s2.getdata();
  s2.putdata();
  Student s3;

  s3.getdata();
  s3.putdata();
  cout << "Total objects created = " << Student::objectCount << endl;
  return 0;
```

```
Enter roll number: 1
Enter name: Mark
Enter marks: 78
Roll Number = 1
Name = Mark
Marks = 78

Enter roll number: 2
Enter name: Nancy
Enter marks: 55
Roll Number = 2
Name = Nancy
Marks = 55

Enter roll number: 3
Enter name: Susan
Enter marks: 90
Roll Number = 3
Name = Susan
Marks = 90
Total objects created = 3
```

Class discussion programs

	// C++ program to demonstrate the	**Output:**

working of friend function

```
#include <iostream>
using namespace std;

class Distance {
   private:
      int meter;

      // friend function
      friend int addFive(Distance);

   public:
      Distance() : meter(0) {}

};

// friend function definition
int addFive(Distance d) {

   //accessing private members from
the friend function
   d.meter += 5;
   return d.meter;
}

int main() {
   Distance D;
   cout << "Distance: " <<
addFive(D);
   return 0;
}
```

Distance: 5

// C++ program to demonstrate the working of friend class

```
#include <iostream>
class A {
private:
        int a;

public:
        A() { a = 0; }
        friend class B; // Friend Class
};

class B {
```

Output:
A::a=0

```
private:
        int b;

public:
        void showA(A& x)
        {
                // Since B is friend of
A, it can access
                // private members of
A
                std::cout << "A::a="
<< x.a;
        }
};

int main()
{
        A a;
        B b;
        b.showA(a);
        return 0;
}
```

// **single inheritance**	**Output**

#include <iostream>
 	
using namespace std;
 	
class base //single base class
 	
{
 	
public:
 	
int x;
 	
void getdata()
 	
{
 	
cout << "Enter the value of x = "; cin >> x;
 	
}
 	
};
 	
class derive : public base //single derived class
 	
{
 	
private:
 	
int y;
 	
public:
 	
void readdata()
 	
{
 	
cout << "Enter the value of y = "; cin >>
	Enter the value of x = 3
	
	Enter the value of y = 4
	
	Product = 12

```
y;
  }
  void product()
  {
    cout << "Product = " << x * y;
  }
};

int main()
{
  derive a;     //object of derived class
  a.getdata();
  a.readdata();
  a.product();
  return 0;
}        //end of program
```

2.C++ Program to Inherit a Student class from Person Class printing the properties of the Student

```
#include <iostream>
#include <conio.h>
```

Output:

Input data

First Name: Harry

Last Name: Potter

Gender: Male

```
using namespace std;

class person    /*Parent class*/

{

 private:

    char fname[100],lname[100],gender[10];

 protected:

    int age;

 public:

    void input_person();

    void display_person();

};

class student: public person    /*Child class*/

{

 private:

    char college_name[100];

    char level[20];

 public:

    void input_student();

    void display_student();

};
```

```
Age: 23

College: Abc International College

Level: Bachelors


Display data

First Name : Harry

Last Name  : Potter

Gender     : Male

Age        : 23

College    : Abc International College

Level      : Bachelors
```

```
void person::input_person()

{

   cout<<"First Name: ";

   cin>>fname;

   cout<<"Last Name: ";

   cin>>lname;

   cout<<"Gender: ";

   cin>>gender;

   cout<<"Age: ";

   cin>>age;

}

void person::display_person()

{

   cout<<"First Name : "<<fname<<endl;

   cout<<"Last Name  : "<<lname<<endl;

   cout<<"Gender     : "<<gender<<endl;

   cout<<"Age        : "<<age<<endl;

}

void student::input_student()

{

   person::input_person();
```

```
    cout<<"College: ";
    fflush(stdin);
    gets(college_name);
    cout<<"Level: ";
    cin>>level;
}
void student::display_student()
{
    person::display_person();
                    cout<<"College                    :
"<<college_name<<endl;
    cout<<"Level      : "<<level<<endl;
}
int main()
{
    student s;
    cout<<"Input data"<<endl;
    s.input_student();
    cout<<endl<<"Display data"<<endl;
    s.display_student();
    getch();
```

```
    return 0;

}
```

3. C++ program to create and display properties of a typist from a staff using Single Inheritance.

```
#include<iostream>

#include<conio.h>

using namespace std;

class staff

{

   private:

       char name[50];

       int code;

   public:

       void getdata();

       void display();

};



class typist: public staff

{

   private:

       int speed;
```

Output

```
Enter data

Name:Roger Taylor

Code:13

Speed:46

Display data

Name:Roger Taylor

Code:13

Speed:46
```

```
    public:
        void getdata();
        void display();
};
void staff::getdata()
{
    cout<<"Name:";
    gets(name);
    cout<<"Code:";
    cin>>code;
}

void staff::display()
{
    cout<<"Name:"<<name<<endl;
    cout<<"Code:"<<code<<endl;
}

void typist::getdata()
{
    cout<<"Speed:";
```

```
    cin>>speed;
}

void typist::display()
{
    cout<<"Speed:"<<speed<<endl;
}

int main()
{
    typist t;
    cout<<"Enter data"<<endl;
    t.staff::getdata();
    t.getdata();
    cout<<endl<<"Display data"<<endl;
    t.staff::display();
    t.display();
    getch();
    return 0;
}
```

// **multiple inheritance**
 	
using namespace std;
 	
class A
 	
{
 	
public:
 	
int x;
 	
void getx()
 	
{
 	
cout << "enter value of x: "; cin >> x;
 	
}
 	
};
 	
class B
 	
{
 	
public:
 	
int y;
 	
void gety()
 	
{
 	
cout << "enter value of y: "; cin >> y;
 	
}
 	
};
	Output: enter value of x:10
	enter value of y:10
	Sum =20

class C : public A, public B //C is derived from class A and class B
 	
{
 	
public:
 	
void sum()
 	
{
 	
cout << "Sum = " << x + y;
 	
}
 	
};
 	
int main()
 	
{
 	
C obj1; //object of derived class C
 	
obj1.getx();
 	
obj1.gety();
 	
obj1.sum();
 	
return 0;
 	
} //end of program	

Ambiguity in Multiple Inheritance #include <iostream>	**Do the changes** sample.A::display();

```
#include <conio.h>

using namespace std;

class A

{

  public:

    void display()

    {

        cout <<"This is method of A";

    }

};



class B

{

  public:

    void display()

    {

        cout <<"This is method of B";

    }

};
```

```
sample.B::display();
```

class C: public A, public B
 	
{
  public:
};
 	
int main()
{
  C sample;
  sample.display(); /*causes ambiguity*/
  getch();
  return 0;
}	

C++ program to display *petrol*'s data using Multiple Inheritance from *fuel* and *liquid* #include <iostream> #include <conio.h>	**Enter data** **Specific gravity: 0.7** **Rate(per liter): $0.99**

```
using namespace std;



class liquid

{

   float specific_gravity;

   public:

      void input()

      {

         cout<<"Specific gravity: ";

         cin>>specific_gravity;

      }

      void output()

      {

         cout<<"Specific gravity:
"<<specific_gravity<<endl;

      }

};



class fuel
```

Displaying data

Specific gravity: 0.7

Rate(per liter): $0.99

```
{
    float rate;
    public:
        void input()
        {
            cout<<"Rate(per liter): $";
            cin>>rate;
        }
        void output()
        {
            cout<<"Rate(per liter):
$"<<rate<<endl;
        }
};


class petrol: public liquid, public fuel
{
    public:
        void input()
```

```
        {
            liquid::input();
            fuel::input();
        }
        void output()
        {
            liquid::output();
            fuel::output();
        }
};


int main()
{
    petrol p;
    cout<<"Enter data"<<endl;
    p.input();
    cout<<endl<<"Displaying data"<<endl;
    p.output();
    getch();
```

return 0; } 	

// hierarchial inheritance
 	
#include <iostream>
 	
using namespace std;
 	
class A //single base class
 	
{
 	
public:
 	
int x, y;
 	
void getdata()
 	
{
 	
cout << "\nEnter value of x and y:\n"; cin >> x >> y;
 	
}
 	
};
 	
class B : public A //B is derived from class base
 	
{
	Output
	
	Enter value of x and y:
	
	2
	
	3
	
	Product= 6
	
	Enter value of x and y:
	
	2
	
	3
	
	Sum= 5

```
    public:
            void product()
            {
                cout << "\nProduct= " << x * y;
            }
};
class C : public A //C is also derived from
class base
{
    public:
            void sum()
            {
        cout << "\nSum= " << x + y;
            }
};
int main()
{
    B obj1;         //object of derived class B
    C obj2;         //object of derived class C
```

```
    obj1.getdata();

    obj1.product();

    obj2.getdata();

    obj2.sum();

    return 0;

}  //end of program
```

C++ program to create Employee and Student inheriting from Person using Hierarchical Inheritance

```
#include <iostream>

#include <conio.h>



using namespace std;



class person

{

    char name[100],gender[10];

    int age;

    public:

        void getdata()
```

Output

```
Student

Enter data

Name: John Wright

Age: 21

Gender: Male

Name of College/School: Abc Academy

Level: Bachelor



Displaying data

Name: John Wright

Age: 21

Gender: Male
```

```
        {
            cout<<"Name: ";
            fflush(stdin);  /*clears input stream*/
            gets(name);
            cout<<"Age: ";
            cin>>age;
            cout<<"Gender: ";
            cin>>gender;
        }
        void display()
        {
            cout<<"Name: "<<name<<endl;
            cout<<"Age: "<<age<<endl;
            cout<<"Gender: "<<gender<<endl;
        }
};
class student: public person
{
    char institute[100], level[20];
```

```
Name of College/School: Abc Academy
Level: Bachelor
Employee
Enter data
Name: Mary White
Age: 24
Gender: Female
Name of Company: Xyz Consultant
Salary: $29000


Displaying data
Name: Mary White
Age: 24
Gender: Female
Name of Company: Xyz Consultant
Salary: $29000
```

```
    public:
        void getdata()
        {
            person::getdata();
            cout<<"Name of College/School: ";
            fflush(stdin);
            gets(institute);
            cout<<"Level: ";
            cin>>level;
        }
        void display()
        {
            person::display();
            cout<<"Name of College/School:
"<<institute<<endl;
            cout<<"Level: "<<level<<endl;
        }
};
class employee: public person
```

```
{
    char company[100];
    float salary;
    public:
        void getdata()
        {
            person::getdata();
            cout<<"Name of Company: ";
            fflush(stdin);
            gets(company);
            cout<<"Salary: Rs.";
            cin>>salary;
        }
        void display()
        {
            person::display();
            cout<<"Name of Company:
"<<company<<endl;
            cout<<"Salary: Rs."<<salary<<endl;
```

```
        }
};
int main()
{
    student s;
    employee e;
    cout<<"Student"<<endl;
    cout<<"Enter data"<<endl;
    s.getdata();
    cout<<endl<<"Displaying data"<<endl;
    s.display();
    cout<<endl<<"Employee"<<endl;
    cout<<"Enter data"<<endl;
    e.getdata();
    cout<<endl<<"Displaying data"<<endl;
    e.display();
    getch();
    return 0;
```

// **multilevel inheritance**	**Output**

#include <iostream>
 	
using namespace std;
 	
class base //single base class
 	
{
 	
public:
 	
int x;
 	
void getdata()
 	
{
 	
cout << "Enter value of x= "; cin >> x;
 	
}
 	
};
 	
class derive1 : public base // derived class from base class
 	
{
 	
public:
 	
int y;
 	
void readdata()
 	
{
 	
cout << "\nEnter value of y= ";
	Enter value of x= 2
	
	Enter value of y= 3
	
	Enter value of z= 3
	
	Product= 18

```
cin >> y;

        }

};

class derive2 : public derive1   // derived
from class derive1

{

        private:

        int z;

        public:

        void indata()

        {

        cout << "\nEnter value of z= "; cin
>> z;

        }

        void product()

        {

            cout << "\nProduct= " << x * y *
z;

        }

};

int main()
```

```
{
    derive2 a;      //object of derived class
    a.getdata();
    a.readdata();
    a.indata();
    a.product();
    return 0;
}                       //end of program
```

2. C++ program to create a programmer derived from employee which is himself derived from person using Multilevel Inheritance

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


class person
{
   char name[100],gender[10];
   int age;
```

Output:

```
Enter data
Name: Karl Lens
Age: 31
Gender: Male
Name of Company: Dynamic Info
Salary: $21000
Number of programming language known: 4
Displaying data
Name: Karl Lens
```

public:
 	
void getdata()
 	
{
 	
cout<<"Name: ";
 	
fflush(stdin); /*clears input stream*/
 	
gets(name);
 	
cout<<"Age: ";
 	
cin>>age;
 	
cout<<"Gender: ";
 	
cin>>gender;
 	
}
 	
void display()
 	
{
 	
cout<<"Name: "<<name<<endl;
 	
cout<<"Age: "<<age<<endl;
 	
cout<<"Gender: "<<gender<<endl;
 	
}
 	
};
 	
class employee: public person
	Age: 31
	
	Gender: Male
	
	Name of Company: Dynamic Info
	
	Salary: $21000
	
	Number of programming language known: 4

```
{
    char company[100];
    float salary;
    public:
        void getdata()
        {
            person::getdata();
            cout<<"Name of Company: ";
            fflush(stdin);
            gets(company);
            cout<<"Salary: Rs.";
            cin>>salary;
        }
        void display()
        {
            person::display();
            cout<<"Name of Company:
"<<company<<endl;
            cout<<"Salary: Rs."<<salary<<endl;
```

```
    }

};

class programmer: public employee

{

   int number;

   public:

      void getdata()

      {

         employee::getdata();

         cout<<"Number of programming
language known: ";

         cin>>number;

      }

      void display()

      {

         employee::display();

         cout<<"Number of programming
language known: "<<number;

      }

};
```

```
int main()

{

   programmer p;

   cout<<"Enter data"<<endl;

   p.getdata();

   cout<<endl<<"Displaying data"<<endl;

   p.display();

   getch();

   return 0;

}
```

```
// hybrid inheritance

#include <iostream>

using namespace std;

class A

{

        public:

        int x;

};

class B : public A
```

Output

```
Sum= 14
```

```
{
        public:
        B()     //constructor to initialize x in base class A
        {
          x = 10;
        }
};
class C
 {
        public:
        int y;
        C()   //constructor to initialize y
        {
           y = 4;
      }
};
class D : public B, public C   //D is derived from class B and class C
{
```

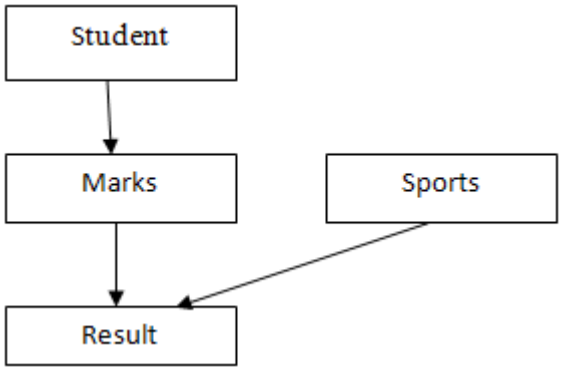
```
        public:

        void sum()

        {

            cout << "Sum= " << x + y;

        }

};



int main()

{

    D obj1;         //object of derived class D

        obj1.sum();

        return 0;

}                   //end of program
```

Hybrid inheritance

A new scheme for evaluation of student's performance is formulated that gives weightage for sports. Extend the inheritance relation depicted in the figure below such that the result, class also inherits properties of sports class. Write an interactive program to model this relationship. Which type of inheritance this model belongs to?

include<iostream.h>

#include<conio.h




```
class stu

{ //First base Class//

        int id;

        char name[20];

        public: //If not declared, data members are by default defined as private//



        void getstu(){

                cout << "Enter stuid, name";

                cin >> id >> name;

        }

};



class marks: public stu{//derived class//

        protected: //without this command,
```

```
data members will not be available next//

        int m, p, c;// without 'protected:' command, m1, m2, & m3 are private members//

        public:

        void getmarks(){

                cout << "Enter 3 subject marks:";

                cin >> m >> p >> c;

        }

};



class sports{//Second base class//

        protected:

        int spmarks;

        public:

        void getsports(){

                cout << "Enter sports marks:";

                cin >> spmarks;

        }

};



class result : public marks, public
```

```
sports{//Derived class by hybrid inheritance//
        int tot;
        float avg;
        public :
        void show(){}
                tot=m+p+c;
                avg=tot/3.0;
                cout << "Total=" << tot << endl;
                cout << "Average=" << avg << endl;
                cout << "Average + Sports marks =" << avg+spmarks;
        }
};
void main(){
        result r;//object//
        r.getstu();
        r.getmarks();
        r.getsports();
        r.show();
        getch();
```

```
};
```

1. Constructor in Derived class

```
#include <iostream>
using namespace std;
class alpha
{
   int x;
public:
   alpha(int i)
   {
      x=i;
      cout<<"alpha initialized";
   }
   void show_x(void)
   {
      cout<<"x="<<x;
   }
};
class beta
{
   float y;
public:
   beta(float j)
   {
      y=j;
      cout<<"beta initialized";

   }
    void show_y(void)
   {
      cout<<"y="<<y;
   }
};

class gamma:public beta,public alpha
{
   int m,n;
public:
   gamma(int a,float b,int c,int d):
   alpha(a),beta(b)
   {
      m=c;
      n=d;
      cout<<"gamma initialized";
   }
    void show_mn(void)
   {


         cout<<"m="<<m;
      cout<<"n="<<n;
}
};
```

```
int main()
{
   gamma g(5,10.75,20,30);
   g.show_x();
   g.show_y();
   g.show_mn();
   return 0;
}
```

Base and derived class constrctors

2. **Base class Default Constructor in Derived class Constructors**

```
class Base

{

   int x;

   public:

   // default constructor

   Base()

   {

      cout << "Base default constructor\n";

   }

};



class Derived : public Base

{

   int y;

   public:
```

OUTPUT:
Base default constructor
Base default constructor
Derived default constructor
Base default constructor
Derived parameterized constructor

```
    // default constructor
    Derived()
    {
        cout << "Derived default constructor\n";
    }
    // parameterized constructor
    Derived(int i)
    {
        cout << "Derived parameterized constructor\
n";
    }
};


int main()
{
    Base b;
    Derived d1;
    Derived d2(10);
}
```

3. Base class Parameterized Constructor in Derived class Constructor.

```
class Base
```

OUTPUT:

Base Parameterized Constructor

Derived Parameterized Constructor

```
{
    int x;
    public:
    // parameterized constructor
    Base(int i)
    {
        x = i;
          cout << "Base Parameterized Constructor\
n";
    }
};


class Derived : public Base
{
    int y;
    public:
    // parameterized constructor
    Derived(int j):Base(j)
    {
        y = j;
                  cout << "Derived Parameterized
Constructor\n";
    }
```

```
};



int main()

{

   Derived d(10) ;

}
```

Virtual base calss

```
#include<iostream>
#include<conio.h>
using namespace std;
class ClassA
{
      public:
      int a;
      ClassA()
      {
         cout<<"class A"<<endl;
      }
       void getfun()
      {
         cout<<"inside A"<<endl;
      }
};

class ClassB : virtual public  ClassA
{
      public:
      int b;
        ClassB()
      {
         cout<<"class B"<<endl;
      }
       void getfun()
      {
         cout<<"inside B"<<endl;
      }
};
class ClassC : virtual public  ClassA
{
      public:
      int c;
        ClassC()
      {
         cout<<"class C"<<endl;
      }
```

```
            void getfun()
            {
               cout<<"inside C"<<endl;
            }
      };

      class ClassD : public ClassB, public
ClassC
      {
            public:
            int d;
              ClassD()
            {
               cout<<"class D"<<endl;
            }
          void getfun()
            {
              cout<<"INSIDE D"<<endl;
            }
      };

      int main()
      {

                  ClassD obj;
//Statement 1
                  obj.a = 100;
//Statement 2

                  obj.b = 20;
                  obj.c = 30;
                  obj.d = 40;
obj.getfun();
                  cout<< "\n A : "<< obj.a;
                  cout<< "\n B : "<<
obj.b;
                  cout<< "\n C : "<< obj.c;
                  cout<< "\n D : "<<
obj.d;

           }
```