NEW EDITION

OBJECT ORIENTED PROGRAMMING LABORATORY

(III semester of B.Tech)

As per the curricullam and syllabus

of

Bharath Institute of Higher Education & Research

OBJECT ORIENTED PROGRAMMING LABORATORY

Bharath
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Declared as Deemed - to - be - University under section 3 of UGC Act 1956)
ACCREDITED WITH 'A' GRADE BY NAAC

PREPARED BY
Mr.D.Paramesh

Bharath
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Declared as Deemed-to-be University under section 3 of UGC Act, 1956)
(Vide Notification No. F.9-5/2000 - U.3, Ministry of Human Resource Development, Govt. of India, dated 4th July 2002)

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

LAB MANUAL

SUBJECT NAME: OBJECT ORIENTED PROGRAMMING LABORATORY

SUBJECT CODE:BCS3L2

Regulation 2015

(2015-2016)

BCS3L2	OBJECT ORIENTED PROGRAMMING LAB	L	T	P	C
	Total Contact Hours - 30	0	0	3	2
	Prerequisite –Fundamental of Computing and Programming, Object Oriented Programming using C++,C.				
	Lab Manual Prepared by – Dept. of Computer Science & Engineering				

OBJECTIVES: This lab manual demonstrates familiarity with various concepts of OOPS.

COURSE OUTCOMES (COs)

CO1	Demonstrate class object concepts by using C++.
CO2	Develop programs using inheritance and polymorphism.
CO3	Demonstrate the significance of constructors and destructor.
CO4	Implement function and operator overloading using C++.
CO5	Construct generic classes using template concepts.
CO6	Implement the concept of file handling.

MAPPING BETWEEN COURSE OUTCOMES & PROGRAM OUTCOMES
(3/2/1 INDICATES STRENGTH OF CORRELATION) 3- High, 2- Medium, 1-Low

COs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	2	3	3		3	2	2	2	3	3	3	3		3	
CO2									2					3	
CO3		3	3	2	2		3		3		3	3		3	
CO4	2					1		2		2				3	
CO5			2		3		2		2		2	2	2	3	
CO6	1	2	2		2	2		1	2	2	3			3	

Category	Professional Core (PC)
Approval	37th Meeting of Academic Council, May 2015

LIST OF EXPERIMENTS:

1. Programs Using Functions
- Functions with default arguments
- Implementation of Call by Value, Call by Address and Call by Reference
2. Simple Classes for understanding objects, member functions and Constructors
- Classes with primitive data members
- Classes with arrays as data members
- Classes with pointers as data members – String Class
- Classes with constant data members, Classes with static member functions
3. Compile time Polymorphism
- Operator Overloading including Unary and Binary Operators, Function Overloading
4. Runtime Polymorphism
- Inheritance ,Virtual functions
- Virtual Base Classes, Templates
- File Handling-Sequential access, Random access.

[BCS3L2]-[OBJECT ORIENTED PROGRAMMING LABORATORY]

LIST OF EXPERIMENTS

PROGRAMMING IN C++

1. Program using Functions
   - Functions with Default arguments, Friend Function, Inline Function
   - Implementation of Call by Value, Address, Reference
2. Simple classes for understanding objects, member functions, constructors, destructor, copy constructor.
   - Classes with Primitive Data Members
   - Classes with Arrays as Data Members
   - Classes with Pointers as Data Members
   - Classes with Constant Data Members
   - Classes with Static Member Functions
3. Compile Time Polymorphism
   - Operator Overloading
   - Function Overloading
4. Run Time Polymorphism
   - Various Forms of Inheritance
   - Virtual Functions
   - Virtual Base Classes
   - Templates

CONTENT

EX. NO.1A **DEFAULT ARGUMENTS IN C++**
DATE:

AIM:

To write a C++ program to find the sum of the given variables using function with default arguments.

ALGORITHM:

1. Start the program.
2. Declare the variables and functions.
3. Give the values for two arguments in the function declaration itself.
4. Call function sum() with three values such that it takes one default arguments.
5. Call function sum() with two values such that it takes two default arguments.
6. Call function sum() with one values such that it takes three default arguments
7. Inside the function sum(), calculate the total.
8. Return the value to the main() function.
9. Display the result.

SOURCE CODE:

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

void main()
{
        float sum(float a,int b=10,int c=15,int =20);

        int a=2,b=3,c=4,d=5;
        clrscr();

        cout<<"\nsum="<<sum(0);
        cout<<"\nsum="<<sum(a,b,c,d);
        cout<<"\nsum="<<sum(a,b,c);
        cout<<"\nsum="<<sum(a,b);
        cout<<"\nsum="<<sum(a);
        cout<<"\nsum="<<sum(b,c,d);
        getch();
}

float sum(float i, int j, int k, int l)
{
return(i+j+k+l);
}
```

OUTPUT:

```
sum=45
sum=14
sum=29
sum=40
sum=47
sum=32
```

RESULT:

Thus, the given program is verified and executed successfully.

EX. NO. 1B **DEFAULT ARGUMENTS IN C++**

DATE:

AIM:

To implement the concept of function with default arguments.

ALGORITHM:

1. Start the program.
2. Declare the default function.
3. Invoke the default function.
4. Display the result.
5. Stop the program.

SOURCE CODE:

```
#include<iostream.h>
void printLine(char ='_',int =70);
void main()
{
        printLine();
        printLine('/');
        printLine('*',40);
        printLine('R',55);
}
void printLine(char ch, int Repeatcount)
{
        int i;
        cout<<endl;
        for(i=0;i<Repeatcount;i++)
                cout<<ch;
}
```

OUTPUT:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R
```

RESULT:

Thus, the given program is verified and executed successfully.

EX NO . 1C **IMPLEMENTATION OF CALL BY VALUE**
DATE:

AIM:

To write a C++ program to find the value of a number raised to its power using call by value.

ALGORITHM:

1. Start the program.
2. Declare two integer type variables X and Y.
3. Get the values for the variables X and Y.
4. Call the function power() to which a copy of the two variables is passed.
5. The function power() is defined to calculate the value of x raised to power y and result is saved in p.
6. Return the value of p to the main function.
7. Display the result.
8. Stop the program.

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
void main()
{
        int x,y;
        double power(int,int);
        clrscr();
        cout<<"Enter ,y:"<<endl;
        cin>>x>>y;
        cout<<x<<" to the power "<<y <<" is "<< power(x,y);
        getch();
}

double power(int x,int y)
{
        double p;
        p=1.0;
        if(y>=0)
                while(y-)
                        p*=x;
        else
                while(y++)
                        p/=x;
        return(p);
}
```

OUTPUT:

Enter X, Y: 2 3
2 To the Power 3 is 8

RESULT:

Thus, the given program is verified and executed successfully.

EX. NO. 1D IMPLEMENTATION OF CALL BY ADDRESS

DATE:

AIM:

To write a C++ program to implement the concept of Call by Address.

ALGORITHM:

1. Start the program.
2. Include suitable header file.
3. Declare a function swap() with two pointer variables *X and *Y as arguments.
4. Declare and initialize the value for the two alias variables i and j in main().
5. Print the value of two variables i and j before swapping.
6. Call the swap() function by passing address of the two variables as arguments.
7. Print the value of two variables i and j after swapping.
8. Stop the program.

SOURCE CODE:

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

void swap(int *x,int *y);
int main()
{
        clrscr();
        int i,j;
        i=10;
        j=20;
        cout<<"\n the value of i before swapping is:"<<i;
        cout<<"\n the value of i before swapping is:"<<j;
        swap (&i,&j);
        cout<<"\n the value of i after swapping is:"<<i;
        cout<<"\n the value of i after swapping is:"<<j;
        getch();
        return(0);
}

void swap(int *x,int*y)
{
        int temp=*x;
        *x=*y;
        *y=temp;

}
```

OUTPUT:

The value of i before swapping is: 20
The value of j before swapping is: 10
The value of i after swapping is: 10
The value of j after swapping is: 20

RESULT:

Thus a program implementing the concept of Call by Address was verified and successfully implemented.

EX.NO:1E **FUNCTION: CALL BY REFERENCE**
DATE:

AIM:

To write a program in C++ to implement the concept of call by reference.

ALGORITHM:

1. Start the program.
2. Declare the variables x,y,m and n.
3. Define the function refer(int&a, int&b) to print the values of a and b.
4. Define the function refer(float &a, float &b) to print the values of a and b.
5. Call the function by reference ie, the variables a and b are alias variables for x,y,m and n.
6. Display the result.
7. Stop the program.

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
void refer(int &a, int &b)
{
        cout<<"\n\n The value of integer is:"<<a;
        cout<<"\n\n The value of integer is:"<<b;
}
void refer(float a, float b)
{
        cout<<"\n\n Value of floating is:"<<a;
        cout<<"\n\n Value of floating is:"<<b;
}
void main()
{
        int x,y;
        float n,m;
        clrscr();
        cout<<"\n Enter the Integer number of two number\n";
        cin>>x>>y;
        cout<<"\n Enter the Floating point of two number\n";
        cin>>n>>m;
        refer(x,y);
        refer(n,m);
        getch();
}
```

OUTPUT:

```
Enter the Integer number of two number   6  4
Enter the Floating point of two number 4.5  4.3
 The value of integer is: 6
 The value of integer is: 4
 Value of floating is: 4.5
 Value of floating is: 4.3
```

RESULT:

The Program to implement Call by Reference is successfully verified and executed using C++.

EX.NO:1F **INLINE FUNCTION**
DATE:

AIM:

To write C++ program to implement inline function.

ALGORITHM:

1. Start the program.
2. Declare and initialize the variables a and b.
3. Define the inline member function mul() using the keyword inline.
4. The parameterized member function mul () is used to get the values of two floating point variables x and y and their product is reurned.
5. The parameterized member function div() is used to get the values of two variables p and q of double data type.
6. Invoke the member function mul().
7. Display the result.
8. Stop the program.

SOURCE CODE:

```
//inline function
#include<iostream.h>
#include<conio.h>
inline float mul(float x,float y)
{
        return(x*y);
}
inline double div(double p,double q)
{
        return(p/q);
}
void main()
{
        clrscr();
        float a = 12.345;
        float b = 9.82;
        cout<<"\nINLINE FUNCTION\n";
        cout<<"Multiplication:"<< mul(a,b)<<"\n";
        getch();
}
```

OUTPUT:

```
INLINE FUNCTION
Multiplication:121.23
Division:1.25
```

RESULT:

The Program to implement the Inline Function has been successfully verified and executed using C++.

EX.NO.2A CLASSES WITH PRIMITIVE DATA MEMBERS

DATE:

AIM:

To write a program in C++ to prepare a student Record using classes with primitive data members.

ALGORITHM:

1. Start the program.
2. Create a class record.
3. Declare the variables name,regno,marks,m1,m2,m3 and avg.
4. Define the member function getdata() to read the values for name, Regno ,mark1,mark2,mark3 of the student.
5. Define the member function calculate() to calculate the average of mark as Avg=(mark1+mark2+mark3)/3.
6. Display the student record using the member function display().
7. Stop the program.

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
class record
{
	public:
		char name[20];
		int regno;
		int marks,m1,m2,m3;
		float avg;
	void getdata()
	{
		cout<<"\nenter the name: " ;
		cin>>name;
		cout<<"enter the regno: ";
		 cin>>regno;
		cout<<"enter the m1,m2,m3: \n";
		cin>>m1>>m2>>m3;
	}

	void calculate()
	{
		avg=(m1+m2+m3)/3;
	}

	void display()
	{
		cout<<"******************\n";
		cout<<"\nName: "<<name;
		cout<<"\nRegno: "<<regno;
		cout<<"\nMark1: "<<m1;
		cout<<"\nMark2: "<<m2;
		cout<<"\nMark3: "<<m3;
		cout<<"\nAvg: "<<avg;
		cout<<"******************\n";
	}
};
```

```
void main()
{
        record r;
       clrscr();
       r.getdata();
       r.calculate();
       r.display();
       getch();
}
```

OUTPUT:

```
Enter the name: Raja
Enter the reg no: 1

Enter the m1,m2,m3: 90,90,90
Name: Raja
Regno: 1
Mark1: 90
Mark2: 90
Mark3: 90
Average:90
```

RESULT:

Thus the program in C++ to prepare a student Record using classes with primitive data members was verified and executed successfully.

EX.NO.2B CLASSES WITH ARRAYS AS DATA MEMBERS

DATE:

AIM:

To write a program in C++ to display product detail using classes with array as data members.

ALGORITHM:

1. Start the program.
2. Create a class product.
3. Declare the variables pro_code, pro_price and count.
4. Define the member function getproduct() to read the values of product code and product cost.
5. Define the member function displaysum() to calculate the sum of product cost.
6. Define the member function displayproduct() to display product details.
7. Declare the object obj and invoke the member functions.
8. Stop the program.

SOURCE CODE:

```
#include<iostream.h>
class product
{
	int pro_code[50];
	float pro_price[50];
	int count;
	public:
	void cnt()
	{
		count=0;
	}
	void getproduct();
	void displaysum();
	void displayproduct();
};
void product::getproduct()
{
	cout<<"Enter product Code:";
	cin>>pro_code[count];
	cout<<"Enter product Cost:";
	cin>>pro_price[count];
	count++;
}
void product::displaysum()
{
	float sum=0;
	for(int i=0;i<count;i++)
		sum=sum+pro_price[i];
	cout<<"Total Value:"<<sum<<"\n";
}
void product::displayproduct()
{
	cout<<" \nCode	Price\n";
	for(int i=0;i<count;i++)
	{
		cout<<"\n"<<pro_code[i];
		cout<<" "<<pro_price[i];
```

```
        }
        cout<<"\n";
}
void main()
{
        product obj;
        obj.cnt();
        int x;
        do
        {
                cout<<"Enter choice\n";
                cout<<"\n----------";
                cout<<"\n1.Add a product";
                cout<<"\n2.Display a product total value";
                cout<<"\n3.Display all products";
                cout<<"\n4.Quit";
                cin>>x;
                switch(x)
                {
                case 1:
                        obj.getproduct();
                case 2:
                        obj.displaysum();
                case 3:
                        obj.displayproduct();
                case 4:break;
                default:
                        cout<<"\n Invalid choice";
                }
        }
        while(x!=4);
}
```

OUTPUT:

```
Code    Price

1234 3000
2345 4000
Enter choice

1.Add a product
2.Display a product total value
3.Display all products
4.Quit
2
Total Value:7000

Code    Price

1234 3000
2345 4000
Enter choice

1.Add a product
2.Display a product total value
3.Display all products
4.Quit
4
```

RESULT:

Thus the C++ program for implementing arrays as data members was created, executed and verified successfully.

EX.NO.2C CLASSES WITH POINTERS AS DATA MEMBERS

DATE:

AIM:

Write a program in C++ to implement the classes with pointers as data members.

ALGORITHM:

1. Start the program.
2. Create a class data.
3. Declare the variable a.
4. Define a function print() to print the value of the variable a.
5. Declare a pointer *dp to object d in the main function
6. To declare a pointer to data member dereference the pointer to what its point to.
7. display the result
8. stop the program

SOURCE CODE:

```
#include<iostream.h>
class data
{
        public:
        int a;
        void print()
        {
                cout<<"a:"<<a;
        }
};
void main()
{
        data d,*dp;
        dp=&d;
        int data::*ptr=&data::a;
        d.*ptr=10;
        d.print();
        dp->*ptr=20;
        dp->print();
}
```

OUTPUT:

a:10

a:20

RESULT:

Thus the C++ program for implementing classes with pointers as data members was verified and executed successfully.

EX.NO.2D CLASSES WITH CONSTANT DATA MEMBER
DATE:

AIM:

To write a program in C++ implements the concept of class with constant data member.

ALGORITHM:

1. Start the program.
2. Create the class test.
3. Declare the integer const variable t.
4. The parameterized constructor test(int t) was created and Initialized.
5. Create two objects t1 and t2 and the value for the variable t is initialized.(Since t is a constant data member every object has an independent copy of t)
6. Display the result.
7. Stop the program.

SOURCE CODE:

```
//constant data member
#include<iostream.h>
#include<conio.h>

class Test
{
   const int t;
public:
   Test(int t): t(t)
         { }         //Initializer list must be used
   int getT()
         {
          return t;
         }
};

void main()
{
   clrscr();
   cout<<"constant data member";
   Test t1(10);
   cout<<"\nDefault t1:"<<t1.getT();
   Test t2(20);
   cout<<"\nDefault t2:"<<t2.getT();
   getch();
}
```

OUTPUT:

```
Default t1:10
Default t2:20
```

RESULT:

Thus the C++ program for implementing classes with constant data members was verified executed successfully.

EX.NO.2E CLASSES WITH STATIC MEMBER FUNCTION
DATE:

AIM:

To write a program in C++ to implement the concept of class with static member functions.

ALGORITHM:

1. Start the program.
2. Create the class test.
3. Declare the variable code and static integer variable count.
4. Define the member function setcode() to autoincrement the value of count.
5. Define the member function showcode() and showcount() to display the values of code and count.
6. Create the objects t1,t2 and t3 and invoke the functions setcode() and showcount().
7. Display the output
8. Stop the program.

SOURCE CODE:

```
//STATIC MEMBER FUNCTION
#include<iostream.h>
#include<conio.h>
class test
	{
		int code;
		static int count;
	public:
		void setcode(void)
		{
			code=++count;
		}
		void showcode(void)
		{
			cout<<"object number:"<<code<<"\n";
		}
		static void showcount(void)
		{
			cout<<"count:"<<count<<"\n";
		}
	};

int test::count;
	void main()
	{
		clrscr();
		test t1,t2;
		t1.setcode();
		t2.setcode();
		test::showcount();

		test t3;
		t3.setcode();
		test :: showcount();
		t1.showcode();
		t2.showcode();
		t3.showcode();
		getch();
```

```
}
```

OUTPUT:

```
count:2
count:3
object number:1
object number:2
object number:3
```

RESULT:

Thus the C++ program for implementing classes with static member function was verified and executed successfully.

EX.NO:2F **FRIEND FUNCTION**
DATE:

AIM:

To write a C++ program to implement the friend function concept.

ALGORITHM:

1. Start the program.
2. Declare the class sample
3. Declare the variables a and b.
4. Define the member function setvalue() to assign the values for a and b.
5. Declare a friend function mean() using the keyword friend.
6. Define a parameterized function mean() with object s as a argument variable.
7. The object s is used to access the private data variables a and b.
8. Create the object x for the class sample.
9. Invoke the member function setvalue using x.
10. Display the value for mean.
11. Stop the program

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
class sample
{
        int a;
        int b;
public:
        void setvalue()
        {
                a=25;
                b=40;
        }
        friend float mean(sample s)
};
float mean(sample s)
{
        return float(s.a+s.b)/2.0;
}
void main()
{
        sample x;
        clrscr();
        x.setvalue();
        cout<<"FRIEND FUNCTION:\n";
        cout<<"Mean value ="<<mean(x)<<"\n";
        getch();
}
```

OUTPUT:

```
        FRIEND FUNCTION:
        Mean value =32.5
```

RESULT:

Thus the Program to implement the Friend Function has been successfully verified and executed using C++.

EX.NO.3A **UNARY OPERATOR OVERLOADING**
DATE:

AIM:

To implement the concept of unary operator overloading using c++.

ALGORITHM:

1. Start the Program.
2. Create a class space and declare necessary data members and member functions and operator function as member function.
3. The operator unary minus is overloaded to perform the operation of changing sign
4. Define member function getdata(), to get three values that is passed as arguments.
5. Define the operator overloading function to change the sign of the values.
6. Define the function display() to display the values before and after sign change.
7. Stop the program.

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
class space
{
        int x,y,z;
public:
        void getdata(int a, int b, int c);
        void display(void);
        void operator-();
};
void space::getdata(int a,int b,int c)
{
        x=a;
        y=b;
        z=c;
}
void space::display(void)
{
        cout<<x<<" ";
        cout<<y<<" ";
        cout<<z<<"\n";
}
void space::operator-()
{
        x=-x;
        y=-y;
        z=-z;
}
void main()
{       clrscr();
        space s;
        s.getdata(10,-20,30);
        cout<<"s:";
        s.display();
        -s;
        cout<<"s:";
        s.display();
        getch();
}
```

OUTPUT:

```
S = 10 -20 30
S = -1020 -30
```

RESULT:

Thus the unary operator overloading concept was successfully executed and verified.

EX.No. 3B BINARY OPERATOR OVERLAODING
DATE:

AIM :

To write a C++ program to implement the concept of Binary operator overloading.

ALGORITHM:

1. Start the program.
2. Declare the class complex.
3. Declare the variables and its member function.
4. Declare the default constructor complex() and a parameterized constructor complex(float real,float imag).
5. Define a binary operator overloading function complex operator+(complex c) which has one object in the argument list.
6. Create the objects c1,c2 and c3 and invoke the constructors and the binary operator overloading function.
7. Invoke the display() function to display the result.
8. Stop the program

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
class complex
{
        float x;
        float y;
 public:
        complex() {}
        complex(float real, float imag)
        {

                x=real;
                y=imag;
        }
        complex operator+(complex c);
        void display(void);
};
complex complex::operator+(complex c)
{
        complex temp;
        temp.x=x+c.x;
        temp.y=y+c.y;
         return(temp);
}
void complex::display(void)
{
        cout<<x<<"+j"<<y<<"\n";
}
void main()
{
        clrscr();
        complex c1,c2,c3;
        c1=complex(2.5,3.5);
        c2=complex(1.6,2.7);
         c3=c1+c2;

        cout<<"c1=";
        c1.display();
```

```
        cout<<"c2=";
        c2.display();

        cout<<"c3=";
        c3.display();
        getch();
}
```

OUTPUT:

C1 = 2.5+j3.5
C2 = 1.6+j2.7
C3 = 4.1+j6.2

RESULT:

Thus the implementation concept of binary operator overloading was successfully completed.

EX.NO.3C FUNCTION OVERLOADING

DATE:

AIM:

To write a C++ program to implement the concept of Function Overloading.

ALGORITHM:

1. Start the program.
2. Create the class
3. Declare the necessary variables.
4. Define three member functions with the same name volume with different number of variables in the argument list.
5. Invoke the member functions and pass the value in the argument list.
6. According to the number of values passed in the argument list the appropriate member function is invoked.
7. Stop the program.

SOURCE CODE:

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

int volume(int s)
{
        return(s*s*s);
}
double volume(double r,int h)
{
        return(3.14*r*r*h);
}
long volume(long l,int b,int h)
{
        return(l*b*h);

}

void main()
{
        clrscr();
        cout<<"!!!VOLUME!!!\n";
        cout<<volume(10)<<endl;
        cout<<volume(10,20)<<endl;
        cout<<volume(10,20,30)<<endl;
        getch();
}
```

OUTPUT:

```
1000
157.26
112500
```

RESULT:

Thus the implementation of function overloading was successfully implemented and verified.

EX.NO:4A **SINGLE INHERITANCE**
DATE:

AIM:

To implement single inheritance using c++.

ALGORITHM:

1. Start the program.
2. Declare the base class emp.
3. Define and declare the function get() to get the employee details.
4. Declare the derived class salary.
5. Declare and define the function get1() to get the salary details.
6. Define the function calculate() as the member of class salary to find the net pay.
7. Define the function display() as the member of class salary to display the details of the employee.
8. Create the object s for the derived class.
9. Read the number of employees.
10. Invoke the member functions get(),get1() and calculate().
11. Invoke the member function display().
12. Stop the program.

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
class emp
{
public:
        int eno;
        char name[20],des[20];
        void get()
        {
                cout<<"Enter the Employee Id : ";
                cin>>eno;
                cout<<"Enter the employee name:";
                cin>>name;
                cout<<"Enter the designation:";
                cin>>des;
        }
};
class salary:public emp
{
        float bp,hra,da,pf,np;
public:
        void get1()
        {
        cout<<"Enter the basic pay:";
        cin>>bp;
        cout<<"Enter the House Rent Allowance:";
        cin>>hra;
        cout<<"Enter the Dearness Allowance :";
        cin>>da;
        cout<<"Enter the Provident Fund:";
        cin>>pf;
}
void calculate()
{
        np=bp+hra+da-pf;
 }
```

```
void display()
{
        cout<<eno<<"\t"<<name<<"\t"<<des<<"\t  "<<bp<<"\t  "<<hra<<"\t "
        <<da<<"\t"<<pf<<"\t"<<np<<"\n";
}
};
void main()
{
        int i,n;
        char ch;
        salary s[10];
        clrscr();
        cout<<"\t EMPLOYEE DETAILS\n";
        cout<<"\t------------------\n";
        cout<<"Enter the number of records :";
        cin>>n;
        for(i=0;i<n;i++)
        {
                s[i].get();
                s[i].get1();
                s[i].calculate();
        }
        cout<<"EmpId   EmpName Designation BasicPay  HRA\tDA\tPF\tNetPay\n";
        for(i=0;i<n;i++)
        {
                s[i].display();
        }
        getch();
}
```

```
                EMPLOYEE DETAILS
          ------------------------------------
Enter the number of records :2
Enter the Employee Id : 1234
Enter the employee name:Rani
Enter the designation:AP
Enter the basic pay:25000
Enter the House Rent Allowance:4000
Enter the Dearness Allowance :2000
Enter the Provident Fund:1000

Enter the Employee Id : 2345
Enter the employee name:Raja
Enter the designation:AP
Enter the basic pay:30000
Enter the House Rent Allowance:6000
Enter the Dearness Allowance :2000
Enter the Provident Fund:2000

EmpId   EmpName Designation BasicPay  HRA     DA     PF     NetPay
1234    Rani            AP      25000          4000   2000   1000    30000
2345    Raja            AP      30000          6000   2000   2000    36000
```

RESULT:

The Program to implement Single Inheritance successfully verified and executed.

EX.NO:4B **MULTIPLE INHERITANCE**
DATE:

AIM:

To write a C++ program to implement multiple inheritance.

ALGORITHM:

1. Start the program.
2. Declare the base class student.
3. Declare and define the function get() to get the student details.
4. Declare the base class sports.
5. Declare and define the function getsm() to read the sports mark.
6. Declare the class statement derived from class student and class sports.
7. Declare and define the function display() to find out the total and average.
8. Declare the derived class object, call the functions get(),getsm() and display().
9. Stop the program.

SOURCE CODE:

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

class student
{
        protected:
        int rollno,m1,m2;
public:
        void get()
        {
                cout<<"Enter the Roll Number :";
                cin>>rollno;
                cout<<"Enter the mark1   :";
                cin>>m1;
                cout<<"Enter the mark2   :";
                cin>>m2;
        }
};
class sports
{
        protected:
        int sm;
public:
        void getsm()
        {
                cout<<"Enter the sports mark :";
                cin>>sm;
        }
};
class report: public student, public sports
{
        int tot,avg;
public:
        void display()
        {
                tot=(m1+m2+sm);
                avg=tot/3;
```

```
                cout<<"\n\tTotal      : "<<tot;
                cout<<"\n\tAverage    : "<<avg;
        }
};


void main()
{
        clrscr();
        report obj;
        cout <<"\t STUDENT DETAILS\n";
        cout <<"\t----------------------------\n";
        obj.get();
        obj.getsm();
        obj.display();
        getch();
}
```

OUTPUT:

```
STUDENT DETAILS
---------------------------
Enter the Roll Number :123456
Enter the mark1   :80
Enter the mark2   :90
Enter the sports mark :78

      Total      : 248
      Average    : 82
```

RESULT:

The Program to implement the Multiple Inheritance has been successfully verified and executed using C++.

EX.NO:4C MULTILEVEL INHERITANCE

DATE:

AIM:

To write a C++ program to implement multilevel inheritance.

ALGORITHM:

1. Start the program.
2. Declare the base class student.
3. Declare and define the function getdata() to get the student details.
4. Declare and define the function putdata() to display the student details.
5. Declare the class test derived from student.
6. Declare and define the function gettest() to read the marks.
7. Declare and define the function puttest() to display the marks.
8. Declare the class result derived from test.
9. Declare and define the function displayresult() to find out the total marks.
10. Declare the derived class object, call the functions getdata(),gettest() and displayresult().
11. Stop the program.

SOURCE CODE:

```
#include<iostream.h>
#include <conio.h>
class student
{
protected:
	int roll_number;
public:
	void get_number(int a)
	{
		roll_number=a;
	}
	void put_number(void)
	{
		cout<<"Roll Number :"<<roll_number<<"\n";
	}
};
class test :public student
{
protected:
	float sub1,sub2;
public:
	void get_marks(float x,float y)
	{
		sub1=x;
		sub2=y;
	}
	void put_marks(void)
	{
		cout<<"\nMarks in Subject 1 ="<<sub1;
		cout<<"\nMarks in Subject 2 ="<<sub2<<"\n";
	}

};
class result: public test
{
protected:
```

```
        float total;
public:
        void display(void)
        {
                total=sub1+sub2;
                put_number();
                put_marks();
                cout<<"Total :"<<total<<"\n";
        }
};
void main()
{
        result s;
        clrscr();
        cout<<"\t MULTILEVEL INHERITANCE";
        cout<<"\n\t-----------------------\n";
        s.get_number(111);
        s.get_marks(80.5,75.5);
        s.display();
        getch();
}
```

OUTPUT:

```
MULTILEVEL INHERITANCE
---------------------------------------
Roll Number :111
Marks in Subject 1 =80.5
Marks in Subject 2 =75.5
Total :156
```

RESULT:

The Program to implement the Multilevel Inheritance has been successfully verified and executed using C++.

EX. NO. 4D VIRTUAL FUNCTION

DATE:
AIM:

To write a C++ program to implement the concept of Virtual functions

ALGORITHM:

1. Start the program.
2. Define the base class base.
3. Define the base class virtual function display() using the keyword virtual.
4. Derive the classes sub1, sub2 from base class base and define the function display() in the respective classes.
5. Declare a base class pointer in main function.
6. Declare objects for d1 and d2 for the classes sub1 and sub2.
7. Assign the objects to the base pointer *bptr.
8. Invoke the display() function using -> pointer.
9. Depending upon the object in the bptr the appropriate display() function is displayed.

SOURCE CODE:

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

```
class base
{
public:
	virtual void display()
	{
		cout<<"Base class display is called\n";
	}
};

class sub1 : public base
{
public:
	void display()
	{
		cout<<"\nDerived class1 display is called\n";
	}
};
class sub2:public base
{
public:
	void display()
	{
		cout<<"\nDerived class2 display is called\n";
	}
};
void main()
{
	clrscr();
	base *bptr,b;
	sub1 d1;
	sub2 d2;
	cout <<"\t VIRTUAL FUNCTION\n";
	cout <<"\t------------------\n";
	bptr=&b;
	bptr->display();
	bptr=&d1;
	bptr->display();
	bptr=&d2;
	bptr->display();
	getch();
```

```
}
```

OUTPUT:

```
 VIRTUAL FUNCTION
----------------------------------
Base class display is called

Derived class1 display is called

Derived class2 display is called
```

RESULT:

Thus the C++ program for virtual function is verified and executed successfully.

EX. NO:4E **VIRTUAL BASE CLASS**
DATE:

AIM:

To write a C++ program to implement the concept of virtual base class.

ALGORITHM:

1. Start the program
2. Include suitable header files
3. Create a base class student.
4. In the base class student define the function get_number() and put_number().
5. In the class sports define the function get_score() and put_score().
6. Derive a class test form base student and define the function get_mark() and put_mark().
7. Derive a class result from test and sports class and define function display().
8. Create the object s for the class result using the object invoke the functions get_number(), get_score(), get_mark() and display().
9. Stop the program

SOURCE CODE:

```
#include<iostream.h>
#include <conio.h>
class student
{
protected:
        int roll_number;
public:
        void get_number(int a)
        {
        roll_number=a;
}
        void put_number(void)
        {
                cout<<"ROLL NO:"<<roll_number<<"\n";
        }
};
class test :public virtual student
{
protected:
        float part1,part2;
public:
        void get_marks(float x,float y)
        {
                part1=x;
                part2=y;
        }
        void put_marks(void)
        {
                cout<<"Marks Obtained:\n";
                cout<<"Part 1 = "<<part1;
                cout<<"\nPart 2 = "<<part2;
        }
};
class sports: public virtual student
{
protected:
        float score;
public:
        void get_score(float s)
        {
                score=s;
        }
```

```
        void put_score(void)
        {
                cout<<"\nSports wt : "<<score;
        }
};

class result: public test, public sports
{

        float total;
 public:
        void display(void);
};
void result::display(void)
{
        total=part1+part2+score;
        put_number();
        put_marks();
        put_score();
        cout<<"\nTotal Score : "<<total<<"\n";
}
void main()
{
        result s;
        clrscr();
        cout<<"\t VIRTUAL BASECLASS";
        cout<<"\n\t-------------------\n";
        s.get_number(678);
        s.get_marks(30.5,25.5);
        s.get_score(7.0);
        s.display();
        getch();
}
```

OUTPUT:

```
VIRTUAL BASECLASS
---------------------------------
ROLL NO:678
Marks Obtained:
Part 1 = 30.5
Part 2 = 25.5
Sports wt : 7
Total Score : 63
```

RESULT:

Thus the C++ program for virtual base class is verified and executed successfully.

EX.NO:4F **CLASS TEMPLATES**

AIM:

To write a C++ program to implement the concept of class template.

ALGORITHM:

1. Start the program.
2. Create the class template test using the keyword template.
3. Declare the two member variables a and b of type T.
4. Define the parameterized constructor with two variables x and y in the argument list of type T1 and T2.
5. The variables a and b acts as alias variable for x and y.
6. Objects test1,test2,test3 and test4 are created and the different type of data values are passed to the constructor test.
7. Display the vales of a and b
8. Stop the process.

SOURCE CODE:

```
#include <iostream.h>
#include <conio.h>
template <class T1, class T2>
class test
{
        T1 a;
         T2 b;
  public :
        test (T1 x, T2 y)
        {
         a= x;
        b= y;
        cout << "\n\nThe value of a is : "<<a;
        cout << "\nThe value of b is : "<<b;
        cout <<" \n Sum is :" << a+b;
        }
};
void main()
{
     clrscr();
     cout<< " CLASS TEMPLATES";
     cout<<"\n------------------";
     test <int,int> test1(10,20);
     test <float,float> test2(15.5,25.5);
     test <int,float> test3(20,25.5);
     test <float,int> test4(15.5,30);
     getch();
 }
```

OUTPUT:

```
CLASS TEMPLATES
---------------------------

The value of a is : 10
The value of b is : 20
 Sum is :30

The value of a is : 15.5
The value of b is : 25.5
 Sum is :41

The value of a is : 20
The value of b is : 25.5
 Sum is :45.5

The value of a is : 15.5
The value of b is : 30
 Sum is :45.5
```

RESULT:

Thus the program for implementing the concept of class template was successfully verifies and executed successfully.

EX. NO:4G **FUNCTION TEMPLATE**
DATE:
AIM:

To write a C++ program for swapping two values using function templates

ALGORITHM:

1. Start the program.
2. Create the template class X using the keyword template.
3. Create the parameterized member function bubble() with two variables in the argument list to perform bubble sort.
4. Define the member function swap() to swap the values of the variables a and b.
5. Display the values in the array before and after sorting.
6. Stop the program.

SOURCE CODE:

```
#include <iostream.h>
#include <conio.h>
template <class T>
void bubble(T a[],int n)
{
        for(int i=0;i<n-1;i++)
                for(int j=n-1;i<j;j--)
                        if ( a[j]<a[j-1])
                                swap(a[j],a[j-1]);
}
template <class X>
void swap(X &a, X &b)
{
         X temp=a;
          a=b;
          b=temp;
 }
 void main()
 {
          int x[5]={6,4,8,2,1};
         float y[5]={1.1,3.5,4.5,2.2,3.3};
     clrscr();
     cout<< " FUNCTION TEMPLATES";
     cout<<"\n------------------";
     cout <<"\nGiven X Array";
     cout <<"\n--------------\n";
     for (int i=0;i<5;i++)
             cout <<x[i]<<"\t";
         bubble(x,5);
          cout <<"\nSorted X Array";
     cout <<"\n--------------\n";
     for ( i=0;i<5;i++)
     cout <<x[i]<<"\t";

         cout <<"\nGiven Y Array";
     cout <<"\n--------------\n";
     for (int j=0;j<5;j++)
     cout <<y[j]<<"\t";
```

```
bubble(y,5);

    cout <<"\nSorted Y Array";
cout <<"\n--------------\n";
for ( j=0;j<5;j++)
cout <<y[j]<<"\t";

getch();
}
```

OUTPUT:

```
FUNCTION TEMPLATES
---------------------------------
Given X Array
-----------------
6      4      8      2      1
Sorted X Array
------------------
1      2      4      6      8
Given Y Array
------------------
1.1    3.5    4.5    2.2    3.3
Sorted Y Array
------------------
1.1    2.2    3.3    3.5    4.5
```

RESULT:

Thus the C++ program for function template was verified successfully executed.