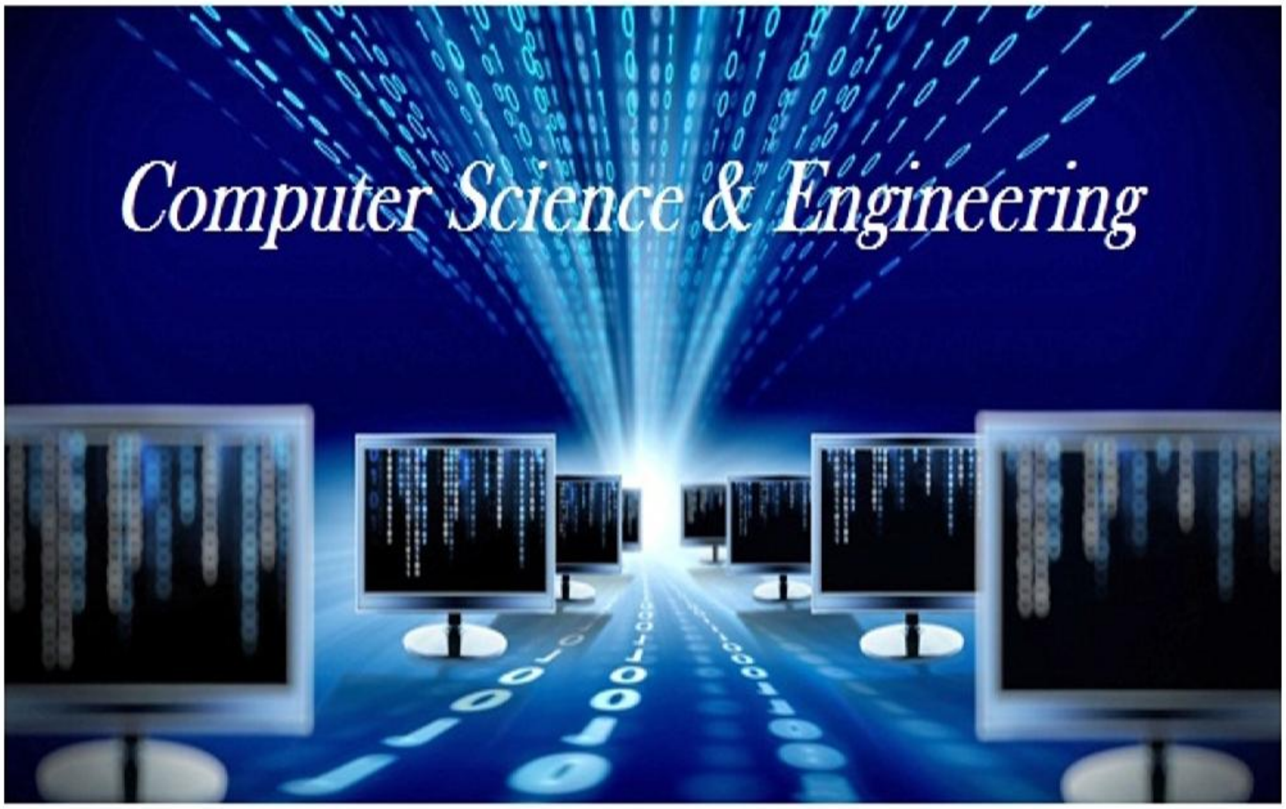
Varuvan Vadivelan Institute of Technology

LAB MANUAL

Regulation	: *2013*
Branch	: ***B.E. – EEE***
Year & Semester	: ***II Year / IV Semester***

CS6461 – OBJECT ORIENTED PROGRAMMING LAB



ANNA UNIVERSITY: CHENNAI

REGULATION - 2013

CS6461 OBJECT ORIENTED PROGRAMMING LABORATORY

LIST OF EXPERIMENTS:

C++ PROGRAMS:

1. Program using functions

. Functions with default arguments

. Implementation of call by value, address, reference

2. Simple classes for understanding objects, member functions & constructors

. 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

. Inheritance

. Virtual functions

. Virtual base classes

. Templates

5. File handling

. Sequential access

. Random access

JAVA PROGRAMS:

6. Simple java applications

. For understanding references to an instant of a class

. Handling strings in JAVA

7. Simple package creation

. Developing user defined packages in java

8. Interfaces

. Developing user defined interfaces

. Use predefined interfaces

9. Threading

. Creation of threading in java applications

. Multi threading

10. Exception handling mechanism in java

. Handling predefined exceptions

. Handling user defined exceptions

TOTAL PERIODS: 45

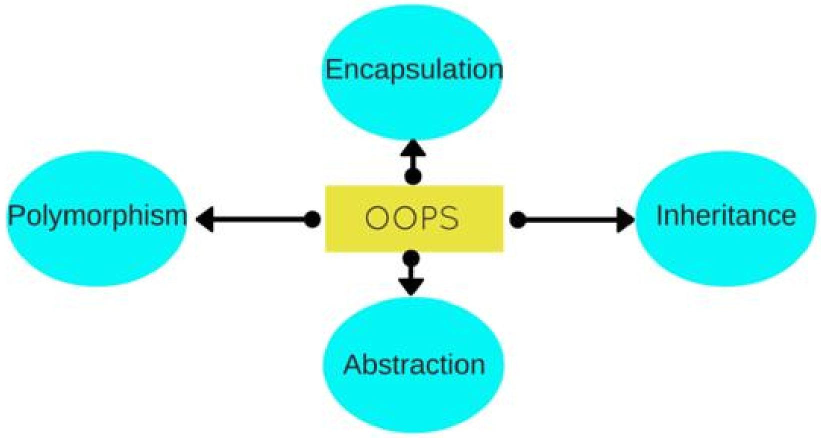
INDEX

INTRODUCTION

Basic Concepts of C++ :

C++ was developed by **Bjarne stroustrup** at bell labs. C++ is an intermediate level language, as it comprises of both high level and low level language features. C++ is an Object Oriented Programming language but is not purely Object Oriented.

Object Oriented programming is a programming style that is associated with the concept of Class, Objects and various other concepts revolving around these two, like Inheritance, Polymorphism, Abstraction, Encapsulation.



Basic Built in types

char	for character storage (1 byte)
int	for integral number (2 bytes)
float	single precision floating point (4 bytes)
double	double precision floating point numbers (8 bytes)

Features of C++:

1. Objects
2. Classes
3. Abstraction
4. Encapsulation
5. Inheritance

6. Overloading
7. Exception Handling

Classes and Objects:

A class is a blueprint for any functional entity which defines its properties and its functions. Like Human Being, having body parts, and performing various actions.

Objects are instances of class, which holds the data variables declared in class and the member functions work on these class objects.

Static Keyword:

Static is a keyword in C++ used to give special characteristics to an element. Static elements are allocated storage only once in a program lifetime in static storage area. And they have a scope till the program lifetime.

Functions:

Functions are used to provide modularity to a program. Creating an application using function makes it easier to understand, edit, check errors

Inheritance:

Inheritance is the capability of one class to acquire properties and characteristics from another class. The class whose properties are inherited by other class is called the **Parent** or **Base** or **Super** class. And, the class which inherits properties of other class is called **Child** or **Derived** or **Sub** class. Inheritance makes the code reusable. When we inherit an existing class, all its methods and fields become available in the new class, hence code is reused.

Function Overloading

If any class has multiple functions with same names but different parameters then they are said to be overloaded. Function overloading allows you to use the same name for different functions, to perform, either same or different functions in the same class. Function overloading is usually used to enhance the readability of the program.

Operator Overloading:

Operator overloading is an important concept in C++. It is a type of polymorphism in which an operator is overloaded to give user defined meaning to it. Overloaded operator is used to perform operation on user-defined data type. For example '+' operator can be overloaded to perform addition on various data types, like for Integer, String(concatenation) etc.

Virtual Functions:

Virtual Function is a function in base class, which is overrided in the derived class, and which tells the compiler to perform Late Binding on this function. Virtual keyword is used to make a member function of the base class Virtual.

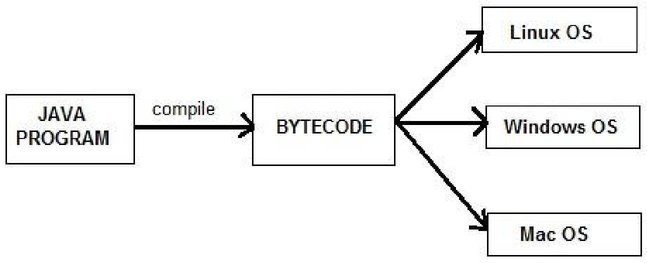
Basic Concepts of JAVA

Java was developed by **James Ghosling, Patrick Naughton, Mike Sheridan** at Sun Microsystems Inc. in 1991. It took 18 months to develop the first working version.

The initial name was **Oak** but it was renamed to **Java** in 1995 as OAK.

Java Features:

1. Simple
2. Object Oriented
3. Robust
4. Platform Independent
5. Secure
6. Multithreading
7. Portable
8. High Performance



Class and Object

A class is declared using **class** keyword. A class contain both data and code that operate on that data. The data or variables defined within a **class** are called **instance variables** and the code that operates on this data is known as **methods**. Object is an instance of class

String:

String is the most commonly used class in java library. String class is encapsulated under java.lang package. String objects are immutable that means once a string object is created it cannot be altered.

Package:

A package can be defined as a group of similar types of classes, interface, enumeration and sub-package. Using package it becomes easier to locate the related classes.

Interface:

Interface is a pure abstract class. They are syntactically similar to classes, but we cannot create instance of an Interface and their methods are declared without any body. Interface is used to achieve complete abstraction in Java

Multithreading:

A program can be divided into a number of small processes. Each small process can be addressed as a single thread (a lightweight process). Multithreaded programs contain two or more threads that can run concurrently. This means that a single program can perform two or more tasks simultaneously. For example, one thread is writing content on a file at the same time another thread is performing spelling check.

Exception Handling:

Exception Handling is the mechanism to handle runtime malfunctions. We need to handle such exceptions to prevent abrupt termination of program. The term exception means exceptional condition, it is a problem that may arise during the execution of program

Ex. No : 1

Date :

FUNCTIONS WITH DEFAULT ARGUMENTS

AIM:

To write a C++ program to implement functions with default arguments

ALGORITHM:

Step 1: Start the program.

Step 2: Declare the simple interest function with default argument.

Step 3: From main function call the required data.

Step 4: Define simple interest function.

Step 5: Calculating simple interest.

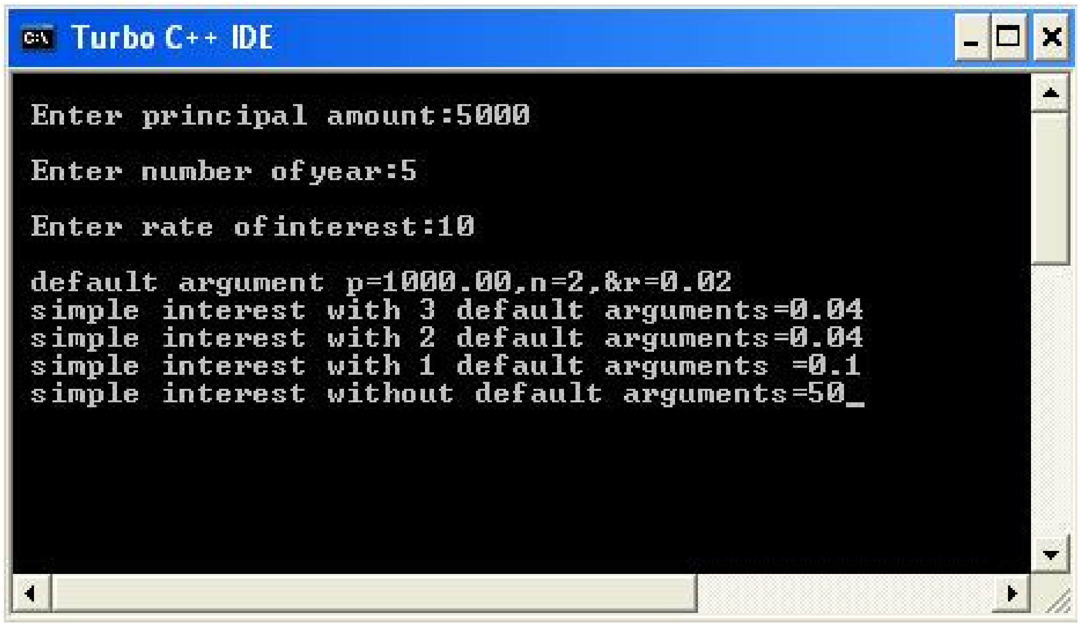
Step 6: Display the details given.

Step 7: Stop the program.

PROGRAM: (FUNCTIONS WITH DEFAULT ARGUMENTS)

```
#include<iostream.h>

#include<conio.h>

float si(float p=1000.00,int n=2,float =0.02); void main()

{

clrscr(); float p,r;

int n;

cout<<"\n Enter principal amount:"; cin>>p;

cout<<"\n Enter number ofyear:";

cin>>n;

cout<<"\n Enter rate ofinterest:";

cin>>r;

cout<<"\n default argument p=1000.00,n=2,&r=0.02";

cout<<"\n simple interest with 3 default arguments="<<si();

cout<<"\n simple interest with 2 default arguments="<<si(p);

 cout<<"\n simple interest with 1 default arguments ="<<si(p,n);

cout<<"\n simple interest without default arguments="<<si(p,n,r);

getch();

}

float si(float pr,int no,float ra)

{

return((pr*no*ra)/pr);

}
```

INPUT AND OUTPUT:



```
Enter principal amount:5000

Enter number ofyear:5

Enter rate ofinterest:10

default argument p=1000.00,n=2,&r=0.02
simple interest with 3 default arguments=0.04
simple interest with 2 default arguments=0.04
simple interest with 1 default arguments =0.1
simple interest without default arguments=50_
```

RESULT:

Thus the implementation of c ++ program for default argument is executed and the output has been verified.

Ex. No: 2

Date:

CALL BY VALUE, CALL BY REFERENCE AND CALL BY ADDRESS

AIM:

To write a C++ program using call by value, call by reference and call by address.

ALGORITHM:

Step 1: Start the program

Step 2: Declare and define a function swapval using call by value

Step 3: Declare and define a function swapref using call by reference

Step 4: Declare and define a function swapadr using call by address

Step 5: Pass necessary arguments to these functions

Step 6: Display the output

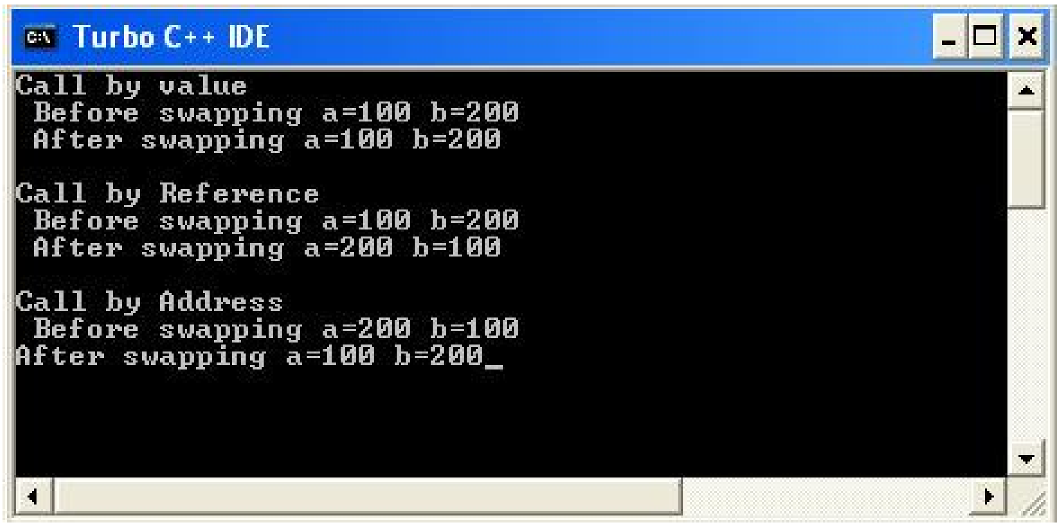
Step 7: Stop the program

PROGRAM:(CALL BY VALUE, CALL BY REFERENCE AND CALL BY ADDRESS)

```
#include<iostream.h>

#include<conio.h>

void swapval(int,int);

void swapref(int &x,int &y);

void swapadr(int *,int*);

void main()

{

int a=100,b=200; clrscr();

cout<<"Call by value";

cout<<"\n Before swapping a="<<a<<" b="<<b;

swapval(a,b);

cout<<"\n After swapping a="<<a<<" b="<<b;

cout<<"\n\nCall by Reference";

cout<<"\n Before swapping a="<<a<<" b="<<b;

swapref(a,b);

cout<<"\n After swapping a="<<a<<" b="<<b;

cout<<"\n\nCall by Address";

cout<<"\n Before swapping a="<<a<<" b="<<b;

swapadr(&a,&b);

cout<<"\nAfter swapping a="<<a<<" b="<<b;

getch();

}
```

```
void  swapval(int x, int y)
{
     int z=x;
     x=y;
     y=z;
}
 void swapref(int &x, int &y)
 {
     int z=x;
     x=y;
     y=z;
 }
 void swapadr( int *x, int *y)
 {
     int z=*x;
     *x=*y;
     *y=z;
 }
```

OUTPUT:



```
Call by value
 Before swapping a=100 b=200
 After swapping a=100 b=200

Call by Reference
 Before swapping a=100 b=200
 After swapping a=200 b=100

Call by Address
 Before swapping a=200 b=100
After swapping a=100 b=200_
```

RESULT:

Thus the implementation of C++ program using call by value, call by reference and call by address is executed and verified.

Ex. No: 3

Date:

CLASSES AND OBJECTS

AIM:

To write a C++ program using Classes and Objects.

ALGORITHM:

Step 1. Start the program

Step 2. Create a class student which contains primitive data members and constant data members

Step 3: Define a constructor to initialize the data members

Step 4: Define the methods get and display to get and display the data members

Step 5: Create object for the class to access the member function

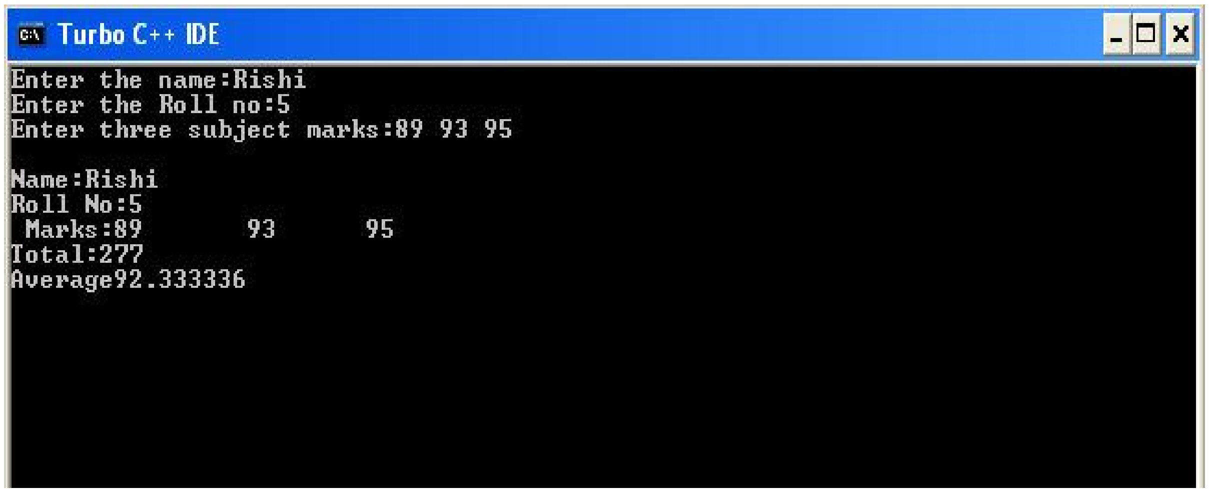
Step 6: Display the result

Step 7: Stop the program.

PROGRAM: (CLASSES AND OBJECTS)

```
#include<iostream.h>

#include<conio.h>

class student

{

int rno;

char name[10]; int marks[3];

int total;

float avg;

const float no; public: student():no(3.0)

{

rno=0; total=0;

avg=0;

}

void get()

{

cout<<"Enter the name:";

cin>>name;

cout<<"Enter the Roll no:";

cin>>rno;

cout<<"Enter three subject marks:";

for(int i=0;i<3;i++)

cin>>marks[i];

}
void display()

{

cout<<"\nName:"<<name;

 cout<<"\nRoll No:"<<rno;

cout<<"\n Marks:";

for(int i=0;i<3;i++)

{ cout<<marks[i]<<"\t";

total+=marks[i];

}

cout<<"\nTotal:"<<total;

cout<<"\nAverage"<<total/no;

}

};

void main()

{

student s;

clrscr();

s.get();

s.display();

getch();

}
```

INPUT AND OUTPUT:



```
Enter the name:Rishi
Enter the Roll no:5
Enter three subject marks:89 93 95

Name:Rishi
Roll No:5
 Marks:89       93      95
Total:277
Average92.333336
```

RESULT:

Thus the above program is executed and output is verified successfully.

Ex. No: 4

Date :

STATIC MEMBER FUNCTION

AIM:

To write a C++ program to implement static member function

ALGORITHM:

Step 1: Start the program.

Step 2: Declare the class name as Stat with data member s and member functions.

Step 3: The constructor stat() which is used to increment the value of count as 1 to to assign the variable code.

Step 4: The function showcode() to display the code value.

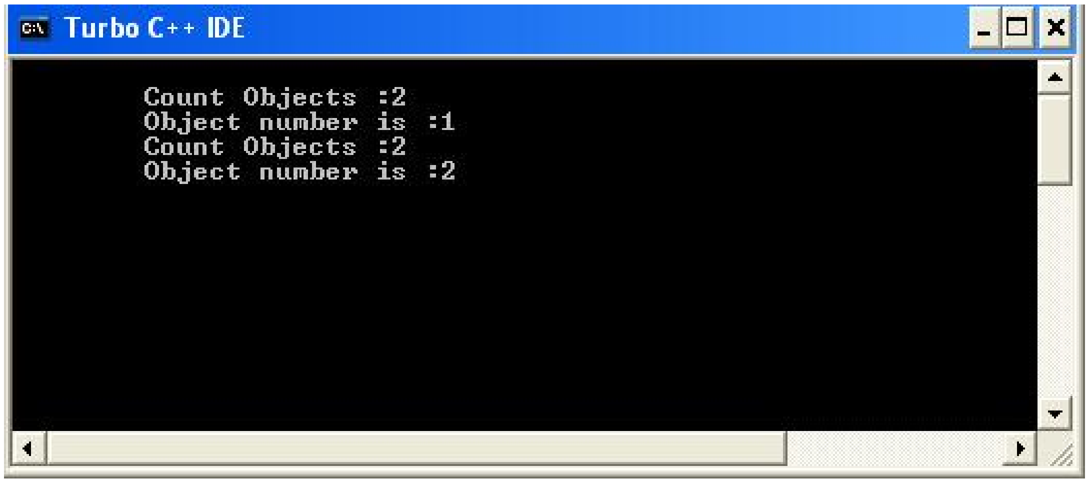
Step 5: The function showcount() to display the count value.

Step 6: Stop the program.

PROGRAM: **(STATIC MEMBER FUNCTION)**

```
#include<iostream.h>
#include<conio.h>
class stat
{
int code;
static int count;
public: stat()
{
code=++count;
}
void showcode()
{
cout<<"\n\tObject number is :"<<code;
}
static void showcount() {
cout<<"\n\tCount Objects :"<<count;
}
};
int stat::count=0;
void main()
{
clrscr();
stat obj1,obj2;

obj1.showcount();
obj1.showcode();
obj2.showcount();
obj2.showcode();
getch();
}
```

OUTPUT:



```
Count Objects :2
Object number is :1
Count Objects :2
Object number is :2
```

RESULT

Thus the above program is executed and output is verified successfully.

Ex. No: 5

Date :

OPERATOR OVERLOADING

AIM:

To write a C++ program to perform complex no addition using operator overloading.

ALGORITHM:

Step 1. Start the program

Step 2. Declare a class as complex with real and imaginary part as data member s

Step 3. Define constructor overloading to assign different value for complex data member

Step 4. Define member function getdata() to get the value of complex no

Step 5. Define operator function +() to perform complex addition

Step 6. Define member function Display() to display complex number.

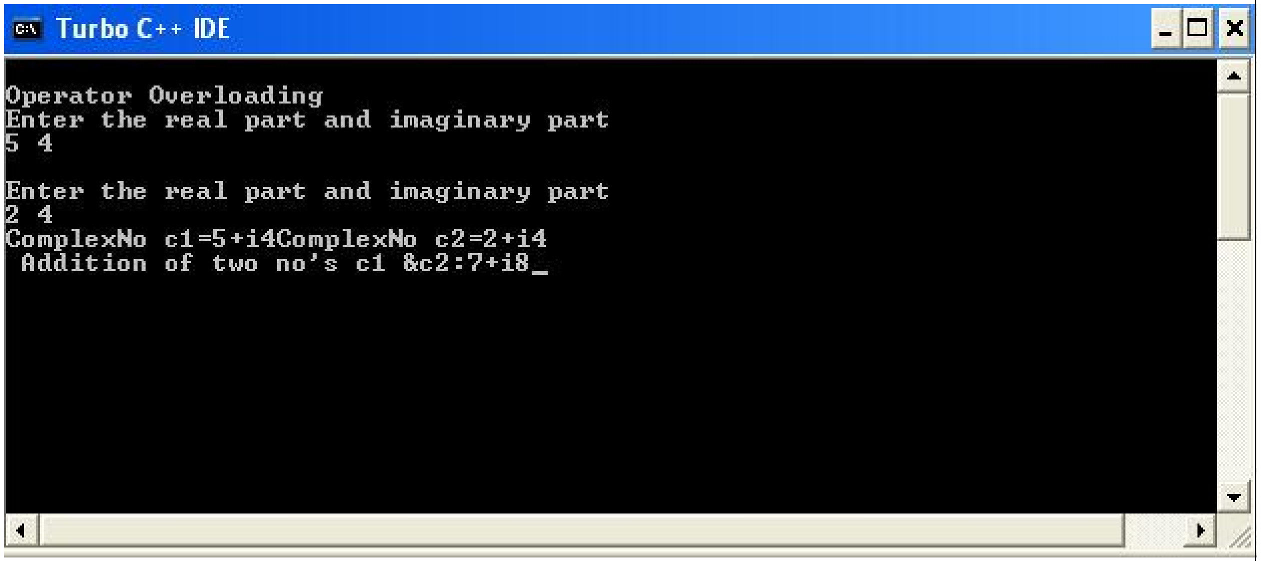
Step 7. In main function create object, invoke constructor, member function and operator function through object

Step 8. Stop the program

PROGRAM: (OPERATOR OVERLOADING)

```
#include<iostream.h>
#include<conio.h>
class complex
{
double real,imag;
public : complex()
{
real=0;imag=0;
}
complex (double r,double i)
{
real=r; imag=i;
}
void getdata()
{
cout<<"Enter the real part and imaginary part\n";
cin >>real>>imag;
}
complex operator +(complex c2)
{
complex temp;
temp.real=real+c2.real;
temp.imag=imag+c2.imag;

return(temp);
}
void display()
{
cout<<real<<"+i"<<imag;
}
};
void main()
{
clrscr();
complex c1,c2,c3;
cout<<"Operator Overloading"; c1.getdata();
c2.getdata();
cout<<"ComplexNo c1=";
c1.display();
cout<<"ComplexNo c2=";
c2.display();
c3=c1+c2;
cout <<"\n Addition of two no's c1 &c2:"; c3.display();
getch();
}
```

INPUT AND OUTPUT:



```
Operator Overloading
Enter the real part and imaginary part
5 4

Enter the real part and imaginary part
2 4
ComplexNo c1=5+i4ComplexNo c2=2+i4
 Addition of two no's c1 &c2:7+i8_
```

RESULT:

Thus the implementation of C++ program for operator overloading is executed and verified.

Ex. No: 6

Date :

FUNCTION OVERLOADING

AIM:

To write a C++ program to calculate volume of cube, cylinder and rectangle using function overloading.

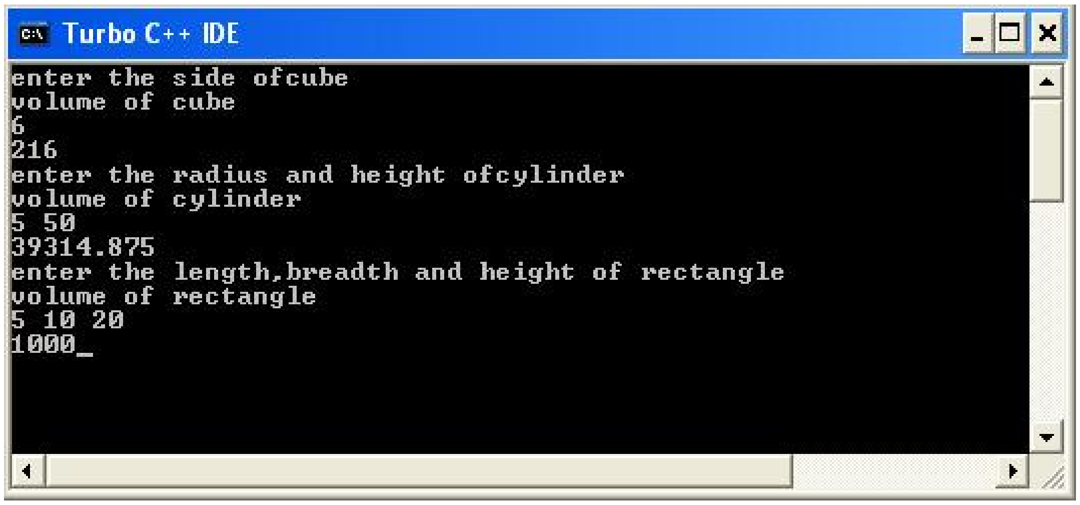
ALGORITHM:

Step 1. Start the program.
Step 2. Declare the prototypes for volume function to find volume of cube, cylinder, rectangle.
Step 3. Get the input values such as side, length, breadth, height, and radius.
Step 4. Invoke volume function of cylinder by passing radius and height to find volume of cylinder.
Step 5. Invoke volume function of cube by passing side of cube to find volume of cube.
Step 6. Invoke volume function of Rectangle by passing length, breadth and height to find volume of rectangle.
Step 7. Print the volume of cube, cylinder and rectangle
Step 8. Stop the program.

PROGRAM: (FUNCTION OVERLOADING)

```
#include<iostream.h>
 #include<conio.h>
int volume(int);
double volume(double,int);
long volume(long,int,int);
int main()
{
int n,r,b,h1;
double h;
long l;
clrscr();
cout<<"enter the side ofcube\n";
cout<<"volume of cube"<<"\n"; cin >>n;
cout<<volume(n);
cout<<"enter the radius and height ofcylinder\n";
cout<<"volume of cylinder"<<"\n";
cin >>r>>h; cout<<volume(h,r);
cout<<"enter the length,breadth and height of rectangle\n";
cout<<"volume of rectangle"<<"\n";
cin >>l>>b>>h1;
 cout<<volume(l,b,h1);
getch();
return 0;
}
int volume(int s)
{
return(s*s*s);
}

double volume(double r,int h)
{
return (3.14519*r*r*h);
}
long volume (long l,int b,int h)
{
return (l*b*h);
}
```

INPUT AND OUTPUT:



```
enter the side ofcube
volume of cube
6
216
enter the radius and height ofcylinder
volume of cylinder
5 50
39314.875
enter the length,breadth and height of rectangle
volume of rectangle
5 10 20
1000_
```

RESULT:

Thus the implementation of C++ program for function overloading is executed and verified.

Ex. No: 7

Date :

INHERITANCE AND VIRTUAL BASE CLASS

AIM:

To calculate the total mark of a student using the concept of inheritance and virtual base class.

ALGORITHM:

Step 1: Start the program.

Step 2: Declare the base class student.

Step 3: Declare and define the functions getnumber() and putnumber().

Step 4: Create the derived class test virtually derived from the base class student.

Step 5: Declare and define the function getmarks() and putmarks().

Step 6: Create the derived class sports virtually derived from the base class student.

Step 7: Declare and define the function getscore() and putscore().

Step 8: Create the derived class result derived from the class test and sports.

Step 9: Declare and define the function display() to calculate the total.

Step 10: Create the derived class object obj.

Step 11: Call the function get number(),getmarks(),getscore() and display().

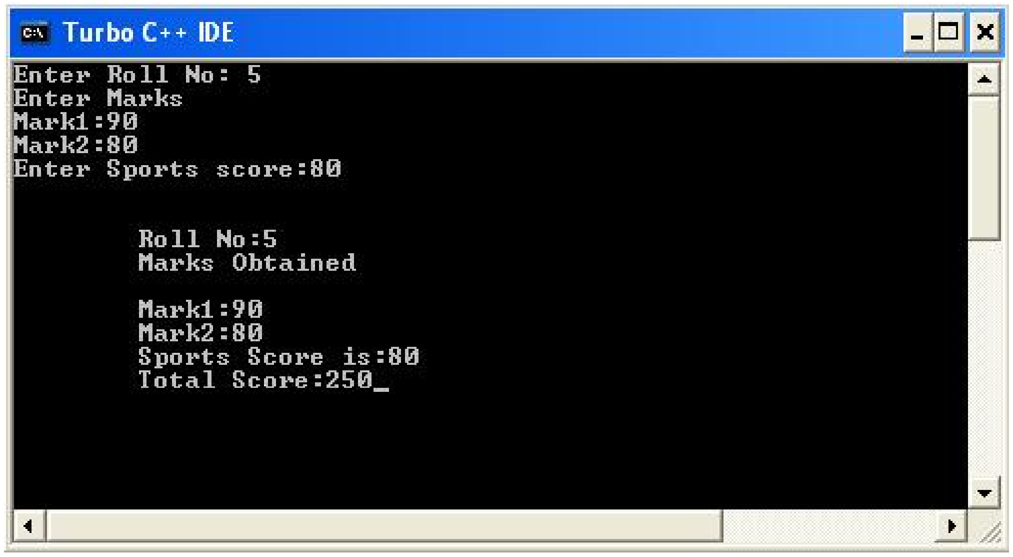
Step 12: Stop the program.

PROGRAM:(INHERITANCE AND VIRTUAL BASE CLASS)

```
#include<iostream.h>
#include<conio.h>
class student
{
int rno;
public:
void getnumber()
{
cout<<"Enter Roll No:";
cin>>rno;
}
void putnumber()
{
cout<<"\n\n\tRoll No:"<<rno<<"\n";
}
};
class test:virtual public student
{
public:
int Mark1,Mark2;
void getmarks()
{
cout<<"Enter Marks\n";
cout<<"Mark1:";
cin>>Mark1;
cout<<"Mark2:";
cin>>Mark2;
}
void putmarks()
{
cout<<"\tMarks Obtained\n";
cout<<"\n\tMark1:"<<Mark1;
cout<<"\n\tMark2:"<<Mark2;
}
};
```

```
class sports:public virtual student

{
public:
int score;
void getscore()
{
cout<<"Enter Sports score:";
cin>>score;
}
void putscore()
{
cout<<"\n\tSports Score is:"<<score;
}
};
class result:public test,public sports
{
int total;
public:
void display()
{
total=Mark1+Mark2+score;
putnumber();
putmarks();
putscore();
cout<<"\n\tTotal Score:"<<total;
}
};
void main()
{
result obj;
clrscr();
obj.getnumber();
obj.getmarks();
obj.getscore();
obj.display();
getch();}
```

INPUT AND OUTPUT:



```
Enter Roll No: 5
Enter Marks
Mark1:90
Mark2:80
Enter Sports score:80


        Roll No:5
        Marks Obtained

        Mark1:90
        Mark2:80
        Sports Score is:80
        Total Score:250_
```

RESULT:

Thus the implementation of inheritance and virtual base class is executed and verified successfully.

Ex. No: 8

Date :

VIRTUAL FUNCTIONS

AIM:

To write a C++ program to implement run time polymorphism through virtual function.

ALGORITHM:

Step 1. Start the program

Step 2. Declare a base and define display() member function to display base class content. Define show() member function to show base class content.

Step 3. Declare a derived class inherit from base and define display() member function to display derived class content. Define show() member function to derived class content.

Step 4. In main function create object for base and derived class.

Step 5. Assign base pointer to base class object.

Step 6. Invoke display function of base class using base pointer variable

Step 7. Invoke show function of base class using base pointer variable

Step 8. Assign base pointer to derived class object.

Step 9. Invoke display function of base class using base pointer variable

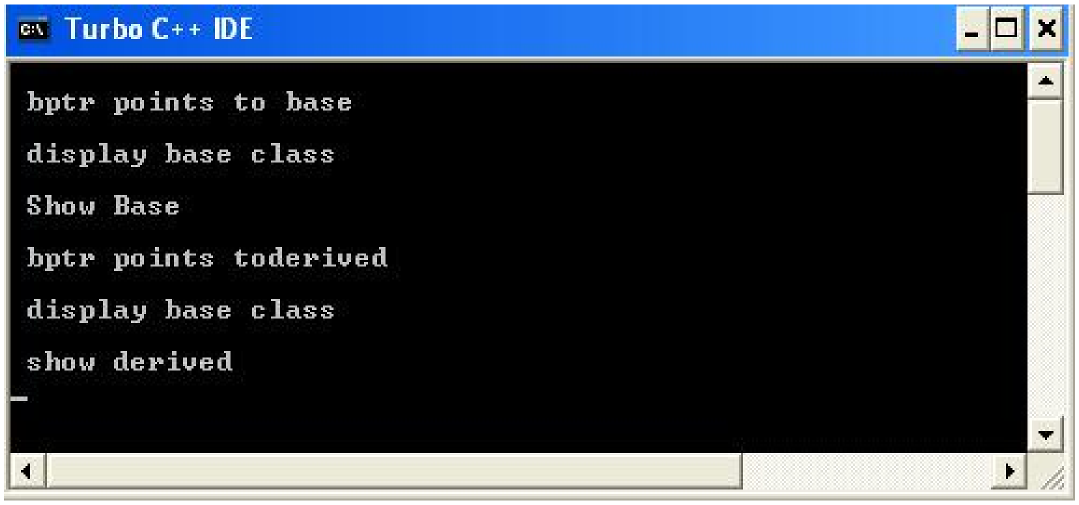
Step 10. Invoke show function of derived class using base pointer variable

Step 11. Stop the program

PROGRAM : (VIRTUAL FUNCTIONS)

```
/*Program using Run time polymorphism */
#include<iostream.h>
#include <conio.h>
class base
{
public:
void display()
{
cout<<"\n display base class\n";
}
virtual void show()
{
cout<<"\n Show Base";
}
};
class derived :public base
{
public:
void display()
{
cout<<"\n display derived class\n";
}
void show()
{
cout<<"\n show derived\n";
}
};
int main()
{
base b; derived d;
base *bptr;
clrscr();
cout<<"\n bptr points to base\n";
bptr=&b;
bptr->display();
bptr->show();
cout<<"\n bptr points toderived";
bptr=&d;
bptr->display();
bptr->show();
getch();
return 0;
}
```

OUTPUT



```
bptr points to base
display base class
Show Base
bptr points toderived
display base class
show derived
_
```

RESULT:

Thus the implementation of virtual functions is executed and verified successfully.

Ex. No : 9

Date :

FUNCTION TEMPLATE

AIM:

To write a C++ program to implement function template.

ALGORITHM:

Step 1: Start the program.

Step 2: Create the function template, to find the maximum of two numbers

Step 3: Get different data type values like integer, float and character as argument to the function

Step 4: Find maximum of those data using template function.

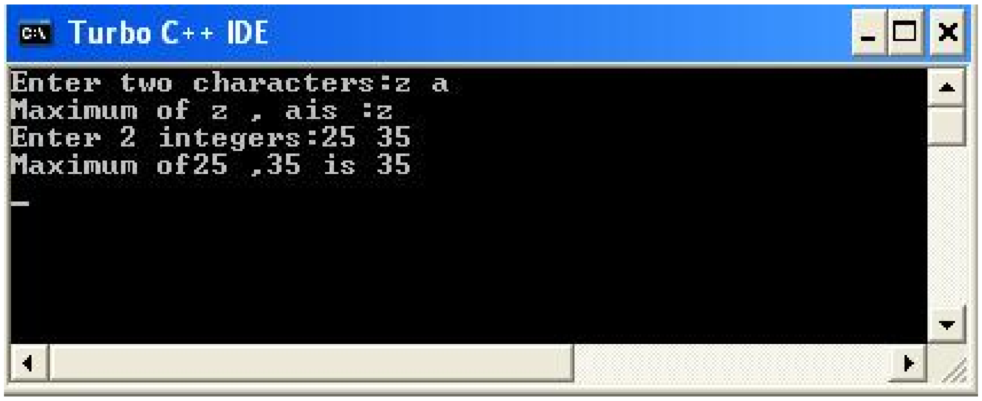
Step 5: Display the result

Step 6: Stop the program

PROGRAM: (FUNCTION TEMPLATE)

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
template<class t>
t max(t a,t b)
{
if (a>b) return a; else return b;
}
void main()
{
clrscr();
char ch1,ch2;
int a,b;
cout<<"Enter two characters:"; cin>>ch1>>ch2;
cout<<"Maximum of "<<ch1 <<” , “<<ch2 <<”is
:”<<max(ch1,ch2)<<endl;
cout<<"Enter 2 integers:";
cin>>a>>b;
cout<<"Maximum of"<<a<<” ,”<<b<<” is ”<<max(a,b)<<endl;
getch();
}
```

INPUT AND OUTPUT:



```
Enter two characters:z a
Maximum of z , ais :z
Enter 2 integers:25 35
Maximum of25 ,35 is 35
_
```

RESULT:

Thus the implementation of function template is executed and verified successfully.

Ex. No : 10

Date :

FILE HANDLING: SEQUENTIAL FILE ACCESS

AIM:

To write a C++ program for creating student data using sequential file access.

ALGORITHM:

Step 1: Start the program

Step 2: Define student class and create the function get and show.

Step 3: Use ofstream and ifstream, get and display the file information.

Step 4: Invoke function using sequential file access.

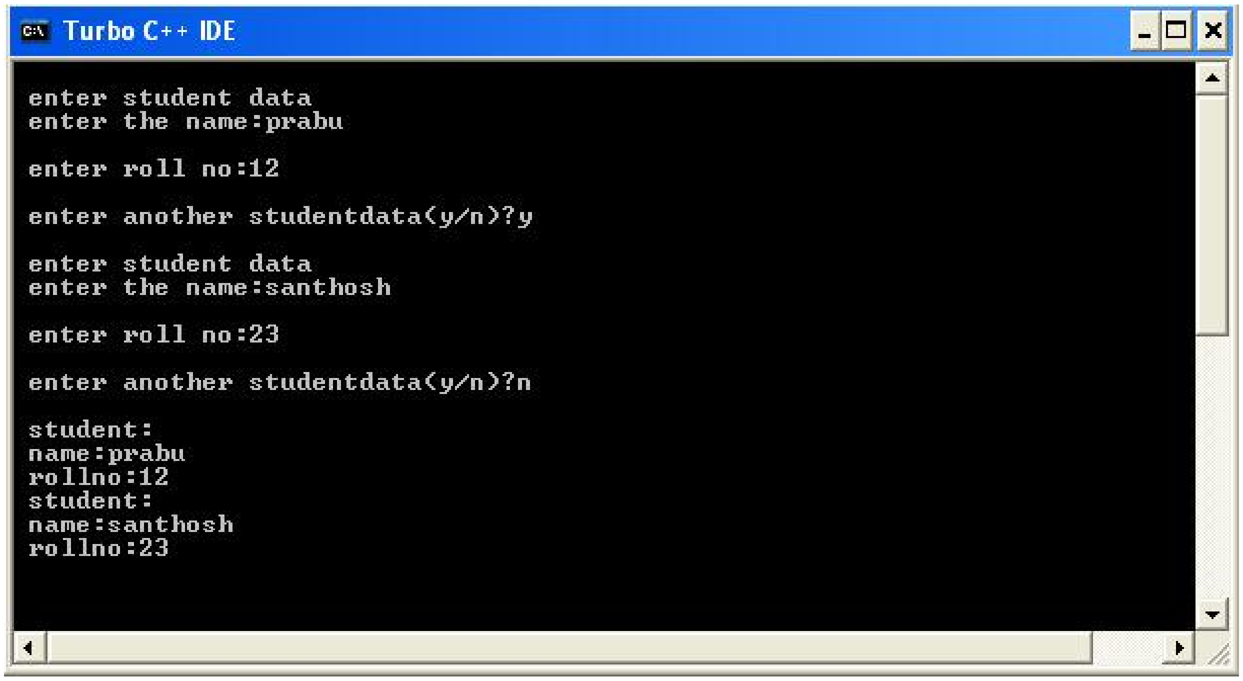
Step 5: Display the output

Step 6: Stop the program

PROGRAM: **(FILE HANDLING: SEQUENTIAL FILE ACCESS)**

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
class student
{
protected: char name[10];
int rollno; public:
void get()
{
cout<<"\n enter the name:";
cin>>name;
cout<<"\n enter roll no:";
cin>>rollno;
}
void show()
{
cout<<"\n name:"<<name;
cout<<"\n rollno:"<<rollno; }  };
void main()
{
char ch; student s; ofstream out;
clrscr();
out.open("file1.txt");
do
 {
cout<<"\n enter student data";
 s.get();
out.write((char*)&s,sizeof(s));
cout<<"\n enter another studentdata(y/n)?";
cin>>ch;
}while(ch=='y');
out.close();
ifstream in;
in.open("file1.txt");
in.seekg(0);

in.read((char*)&s,sizeof(s));
while(!in.eof())
{
cout<<"\n student:"; s.show(); in.read((char*)&s,sizeof(s));
}
getch();
}
```

INPUT AND OUTPUT:



Turbo C++ IDE

```
enter student data
enter the name:prabu

enter roll no:12

enter another studentdata(y/n)?y

enter student data
enter the name:santhosh

enter roll no:23

enter another studentdata(y/n)?n

student:
name:prabu
rollno:12
student:
name:santhosh
rollno:23
```

RESULT:

Thus the C++ program to create student data using sequential file access was executed and output verified.

Ex. No : 11

Date :

FILE HANDLING: RANDOM FILE ACCESS

AIM:

To write a C++ program for creating student data using random file access.

ALGORITHM:

Step 1: Start the program.

Step 2: Define student class and create the function get and show.

Step 3: Overload stream operation function for displaying the file information.

Step 4: Invoke function using random file access.

Step5: Use stream operator function to read and the write the contents

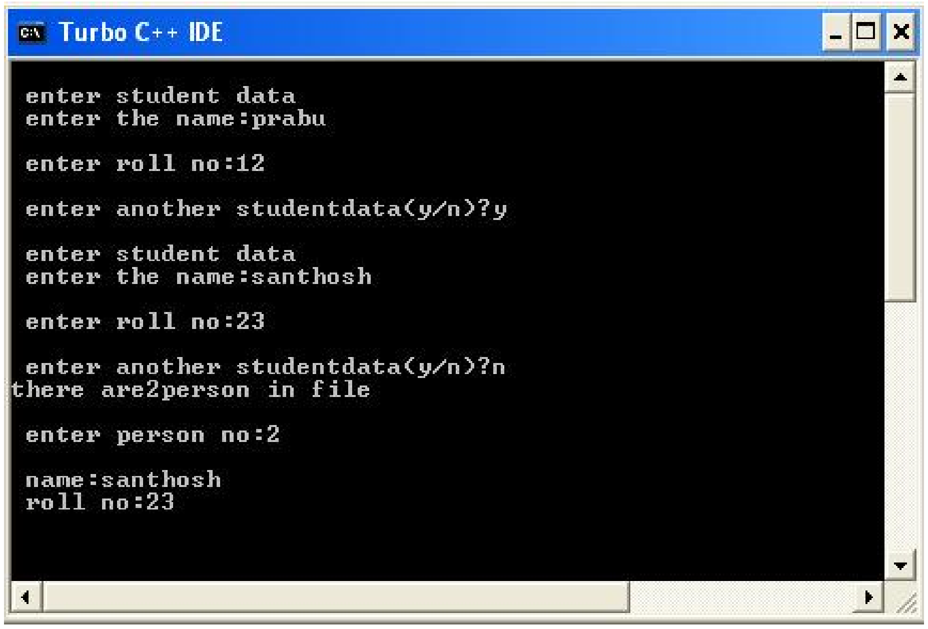
Step 6: Display the result

Step7: Stop the program

PROGRAM: (FILE HANDLING: RANDOM FILE ACCESS)

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
 class student
{
protected: char name[10];
int rollno; public:
void get()
{
cout<<"\n enter the name:"; cin>>name;
cout<<"\n enter roll no:"; cin>>rollno;
}
void show()
{
cout<<"\n name:"<<name;
cout<<"\n roll no:"<<rollno;
}
};
void main()
{
char ch;
student s;
ofstream out;
clrscr();
out.open("file2.txt");
do
{
cout<<"\n enter student data";
s.get();
out.write((char*)&s,sizeof(s));
cout<<"\n enter another studentdata(y/n)?";
cin>>ch;
}while(ch=='y');
out.close();
ifstream in;
in.open("file2.txt");
 in.seekg(0,ios::end);
int endpos=in.tellg();
int n=endpos/sizeof(student);
cout<<"there are"<<n<<"person in file"<<endl; cout<<"\n enter
person no:";
cin>>n;
int pos=(n-1)*sizeof(student); in.seekg(pos);
in.read((char*)&s,sizeof(s)); s.show();
getch();
}
```

INPUT AND OUTPUT:



```
 enter student data
 enter the name:prabu

 enter roll no:12

 enter another studentdata(y/n)?y

 enter student data
 enter the name:santhosh

 enter roll no:23

 enter another studentdata(y/n)?n
there are2person in file

 enter person no:2

 name:santhosh
 roll no:23
```

RESULT:

Thus the C++ program to create student data using Random file access was executed and output verified.

Ex. No: 12

Date :

CLASS AND OBJECT IN JAVA

AIM:

To write a java program to implement class and object

ALGORITHM:

Step 1: Start the program

Step2: Create a class Rectangle

Step 3: Define methods to get the values and to calculate the area of rectangle

Step 4: Create object for the class Rectangle

Step 5: Call the methods using the created object

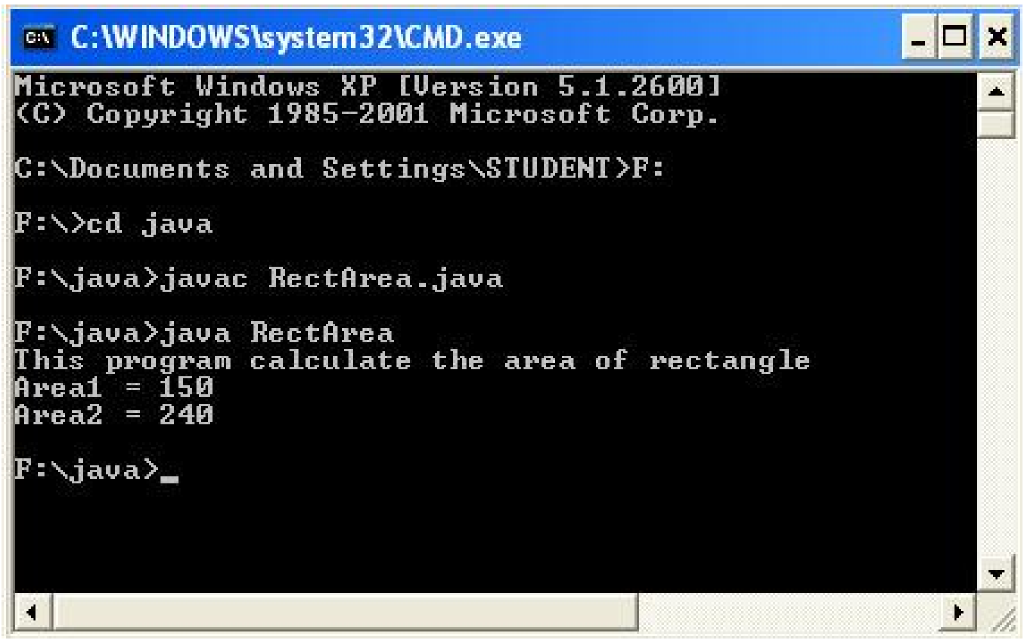
Step6: Display the result

Step 7: Stop the program

PROGRAM: (CLASS AND OBJECT IN JAVA)

```
class Rectangle
{
int length,width;
void getData (int x, int y)
{
length=x;
width=y;
}
int rectArea()
{
int area=length*width;
return (area);
}
}
class RectArea
{
public static void main (String args[])
{
int area1,area2;
System.out.println("This program calculate the area of rectangle");
Rectangle Rect1=new Rectangle();
Rectangle Rect2= new Rectangle();
Rect1.length=15;
Rect1.width=10;
area1=Rect1.length*Rect1.width;
Rect2.getData(20,12);
area2=Rect2.rectArea();
System.out.println("Area1 = " +area1);
System.out.println("Area2 = " +area2);
}
}
```

OUTPUT:



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\STUDENT>F:

F:\>cd java

F:\java>javac RectArea.java

F:\java>java RectArea
This program calculate the area of rectangle
Area1 = 150
Area2 = 240

F:\java>_
```

RESULT

Thus the above program is executed and output is verified

Ex. No: 13

Date :

STRINGS IN JAVA

AIM:

To write a java program to perform string operations.

ALGORITHM:

Step 1: Start the program.

Step2: Create a class to handle string functions.

Step 3: Declare a string and initialize it.

Step 4: Perform the string operations on the string.

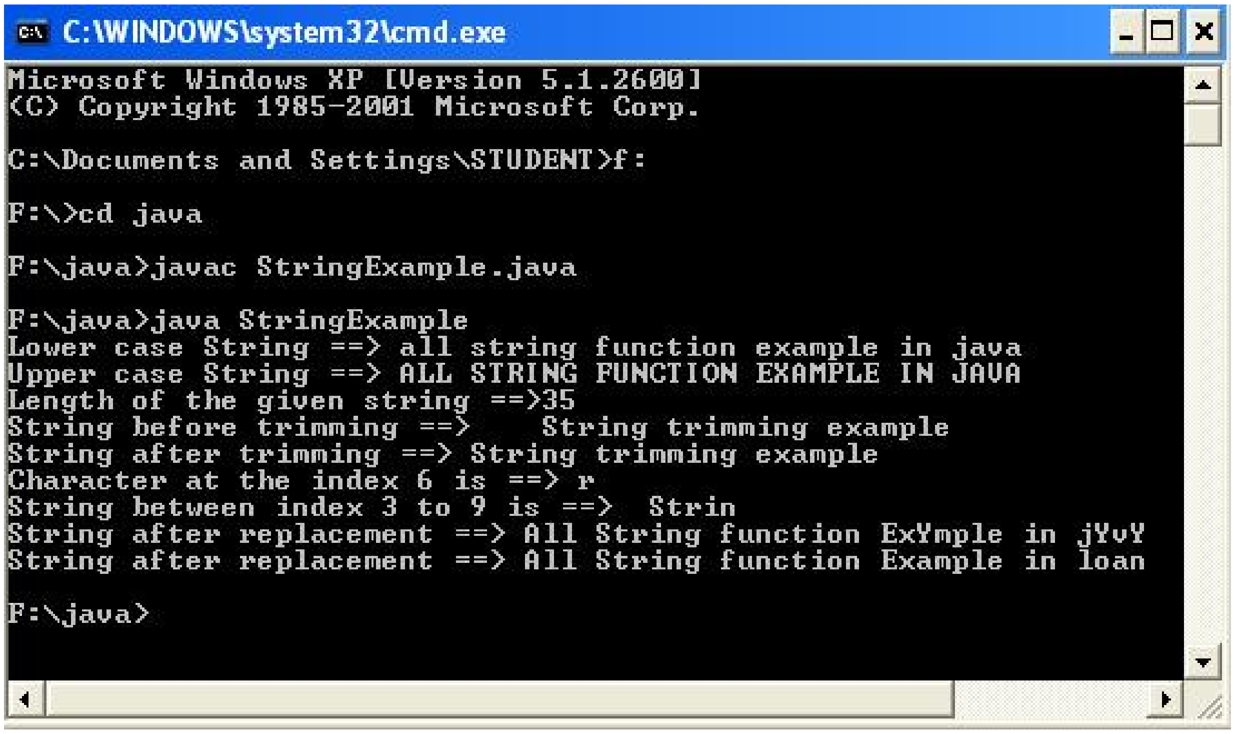
Step 5: Display the output.

Step 6: Stop the program.

PROGRAM: **(STRINGS IN JAVA)**

```
public class StringExample
{
public static void main(String[] args)
{
String str = "All String function Example in java";
//   convert string into Lower case
String Lowercase = str.toLowerCase();
System.out.println("Lower case String ==> " + Lowercase);
//   convert string into upper case
String Uppercase = str.toUpperCase();
System.out.println("Upper case String ==> " + Uppercase);
// Find length of the given string
System.out.println("Length of the given string ==>" +
str.length());
//   Trim the given string i.e. remove all first and last the
spaces from  the string
String tempstr = "   String trimming example  ";
System.out.println("String before trimming ==> " + tempstr);
System.out.println("String after trimming ==> " +
tempstr.trim());
//   Find the character at the given index from the given
string
System.out.println("Character at the index 6 is ==> " +
str.charAt(6));
//   find the substring between two index range
System.out.println("String between index 3 to 9 is ==> "+
str.substring(3, 9));
//   replace the character with another character
System.out.println("String after replacement ==> "
+str.replace("a","Y"));
//   replace the substring with another substring
System.out.println("String after replacement ==> " +
str.replace("java", "loan"));
}
}
```

OUTPUT:



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\STUDENT>f:

F:\>cd java

F:\java>javac StringExample.java

F:\java>java StringExample
Lower case String ==> all string function example in java
Upper case String ==> ALL STRING FUNCTION EXAMPLE IN JAVA
Length of the given string ==>35
String before trimming ==>    String trimming example
String after trimming ==> String trimming example
Character at the index 6 is ==> r
String between index 3 to 9 is ==>  Strin
String after replacement ==> All String function ExYmple in jYvY
String after replacement ==> All String function Example in loan

F:\java>
```

RESULT:

Thus the above program is executed and output is verified.

Ex. No: 14

Date :

PACKAGES IN JAVA

AIM:

To write a java program to implement packages.

ALGORITHM:

Step 1. Start the program

Step 2. Create the package as mypackage and define a class called balance which consist of two data members account holder name and balance

Step 3. Define member function show() to display balance of the account holder.

Step 4. Import the package in new class called as test balance which consist of main function Step 5. Create object for balance and call the display function.

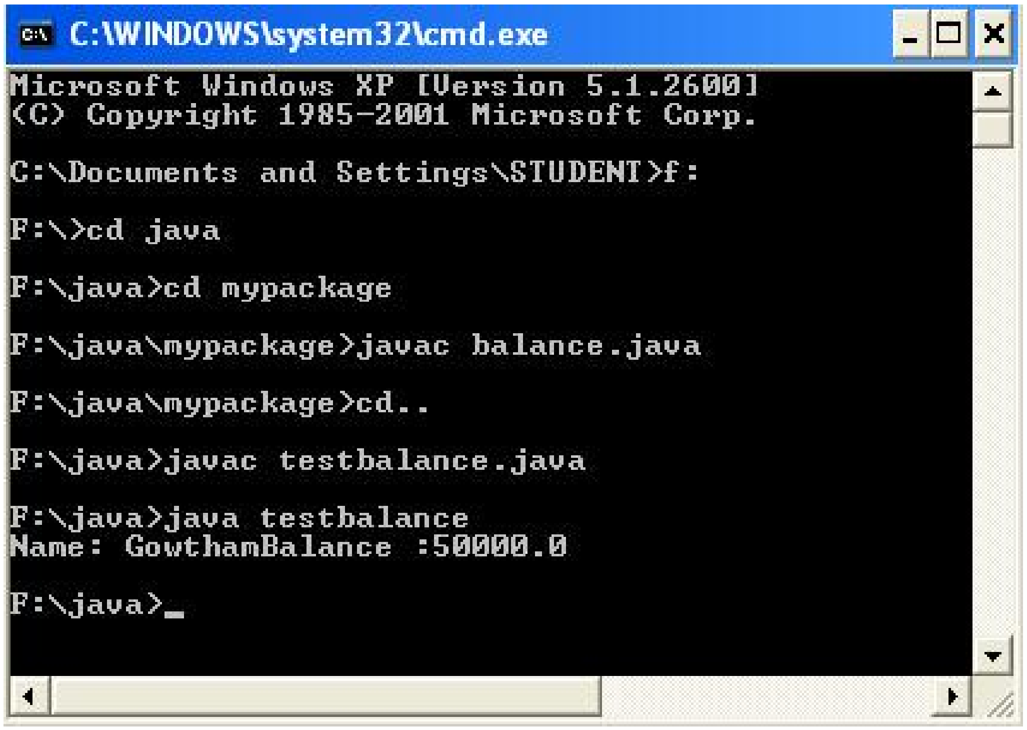
Step 6. Display the result.

Step 7. Stop the program

PROGRAM : (PACKAGES IN JAVA)

```
/* package creation*/
package mypackage;
public class balance
{
String name; double bal;
public balance(String n,double b)
{
name=n;
bal=b;
}
public void show()
{
if(bal>0)
System.out.println("Name:"+name+"Balance :"+bal);
}
}

/* importing package*/
import  mypackage .*;
import  java.io.*;
class testbalance
{
public static void main(String args[])
{
balance test =new balance("Gowtham",50000);
 test.show();
}
}
```

OUTPUT



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\STUDENT>f:

F:\>cd java

F:\java>cd mypackage

F:\java\mypackage>javac balance.java

F:\java\mypackage>cd..

F:\java>javac testbalance.java

F:\java>java testbalance
Name: GowthamBalance :50000.0

F:\java>_
```

RESULT:

Thus the java program to implement packages was executed and output verified.

Ex. No: 15

Date :

INTERFACES IN JAVA

AIM:

To write a java program to find area of circle and rectangle using interface

ALGORITHM:

Step 1. Start the program

Step 2. Create the interface name as area and declare variables and methods in interface

Step 3. Define class rectangle and the class circle which implements the interface area and define member function compute() for each class to compute area of rectangle and circle

Step 4. Create a class called interfacetest define main function and Create object for rectangle and circle

Step 5. Create object for interface

Step 6. Assign rectangle object to interface object and call compute function of rectangle to find area of rectangle

Step 7. Assign circle object to interface object and call compute function of circle to find area of circle.

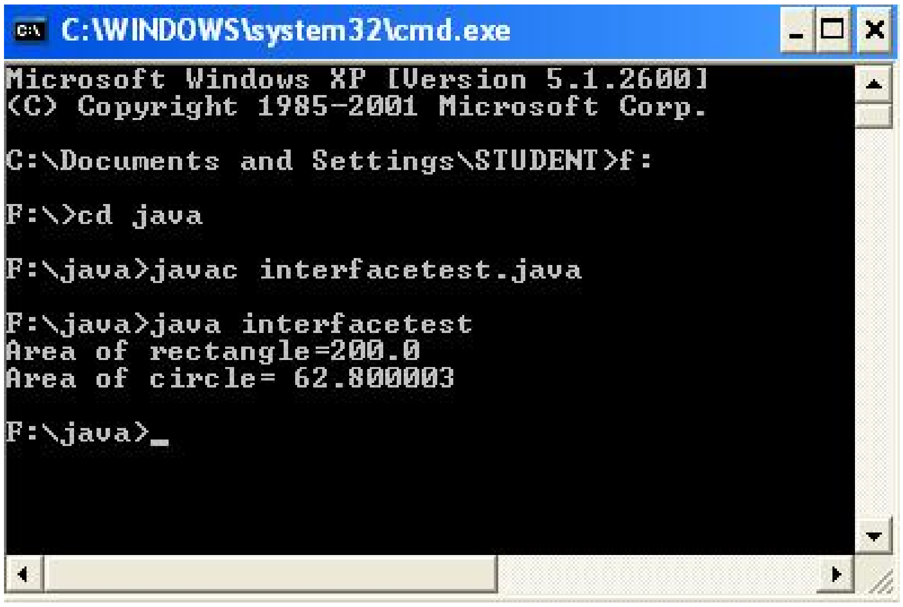
Step 8. Display the result.

Step 9. Stop the program.

PROGRAM : (INTERFACES IN JAVA)

```
import java.io.*;
interface area
{
final static float pi=3.14f;
float compute(float x,float y);
}
class rectangle implements area
{
public float compute (float x,float y)
{   return(x*y);
}
}
class circle implements area
{
public float compute(float x,float y)
{
return (pi*x*y);
}
}
class interfacetest
{
public static void main(String args[])
{
rectangle rect =new rectangle();
circle cir =new circle();
area area1;
area1 =rect;
System.out.println("Area of rectangle="+area1.compute(10,20));
area1=cir;
System.out.println("Area of circle= "+area1.compute(10,2));
}
}
```

OUTPUT



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\STUDENT>f:

F:\>cd java

F:\java>javac interfacetest.java

F:\java>java interfacetest
Area of rectangle=200.0
Area of circle= 62.800003

F:\java>_
```

RESULT

Thus the java program to implement interfaces was executed and output verified

Ex. No: 16

Date :

THREADS IN JAVA

AIM:

To write a java program to implement threads.

ALGORITHM:

Step 1: Start the program

Step 2: Create a class worker that implements the runnable interface

Step 3: Define the run method

Step 4: Create a new thread for the worker class

Step 5: Execute the thread by calling the start method

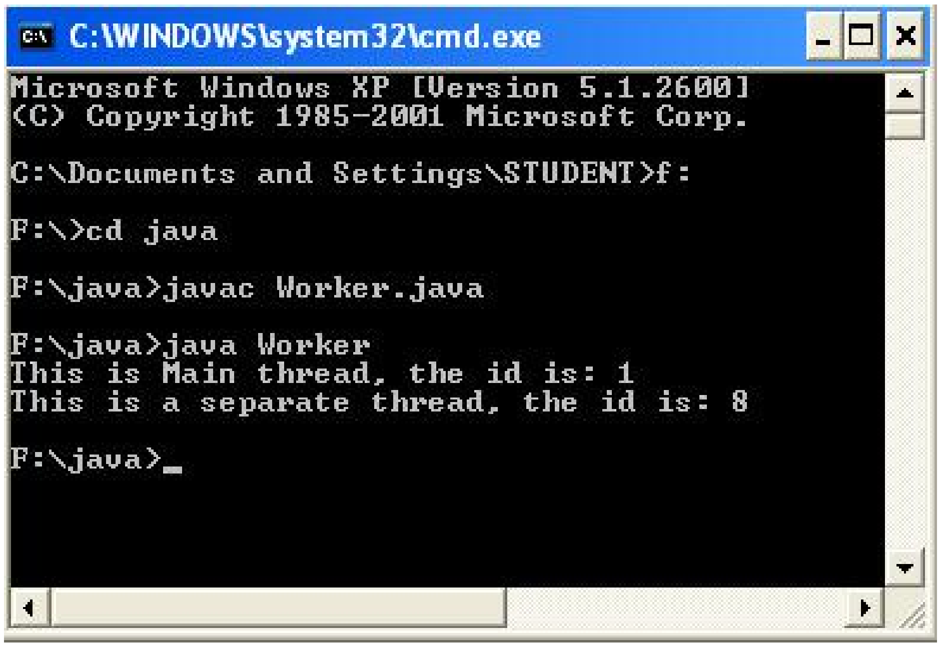
Step 6: Display the result

Step 7: Stop the program

PROGRAM: (THREADS IN JAVA)

```
import java.io.*;
public class Worker implements Runnable
{
public static void main (String[] args)
{
System.out.println("This is Main thread, " +"the id is: " +
Thread.currentThread().getId());
Worker  worker = new Worker();
Thread  thread = new Thread(worker); thread.start();
}
public void run() {
System.out.println("This is a separate thread, " +"the id is:
"  + Thread.currentThread().getId());
}
}
```

OUTPUT:



C:\WINDOWS\system32\cmd.exe

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\STUDENT>f:

F:\>cd java

F:\java>javac Worker.java

F:\java>java Worker
This is Main thread, the id is: 1
This is a separate thread, the id is: 8

F:\java>_
```

RESULT

Thus the java program to implement threads was executed and output verified.

Ex. No: 17

Date :

MULTITHREADING

AIM:

To write a java program to implement multithreading concept

ALGORITHM:

Step 1: Start the program

Step 2: Create a class ThreadDemo that extends the Thread class

Step 3: Override the run method of the Thread class

Step 4: Create a new thread by creating object for the ThreadDemo class

Step 5: Execute the thread by calling the start method

Step 6: Display the result

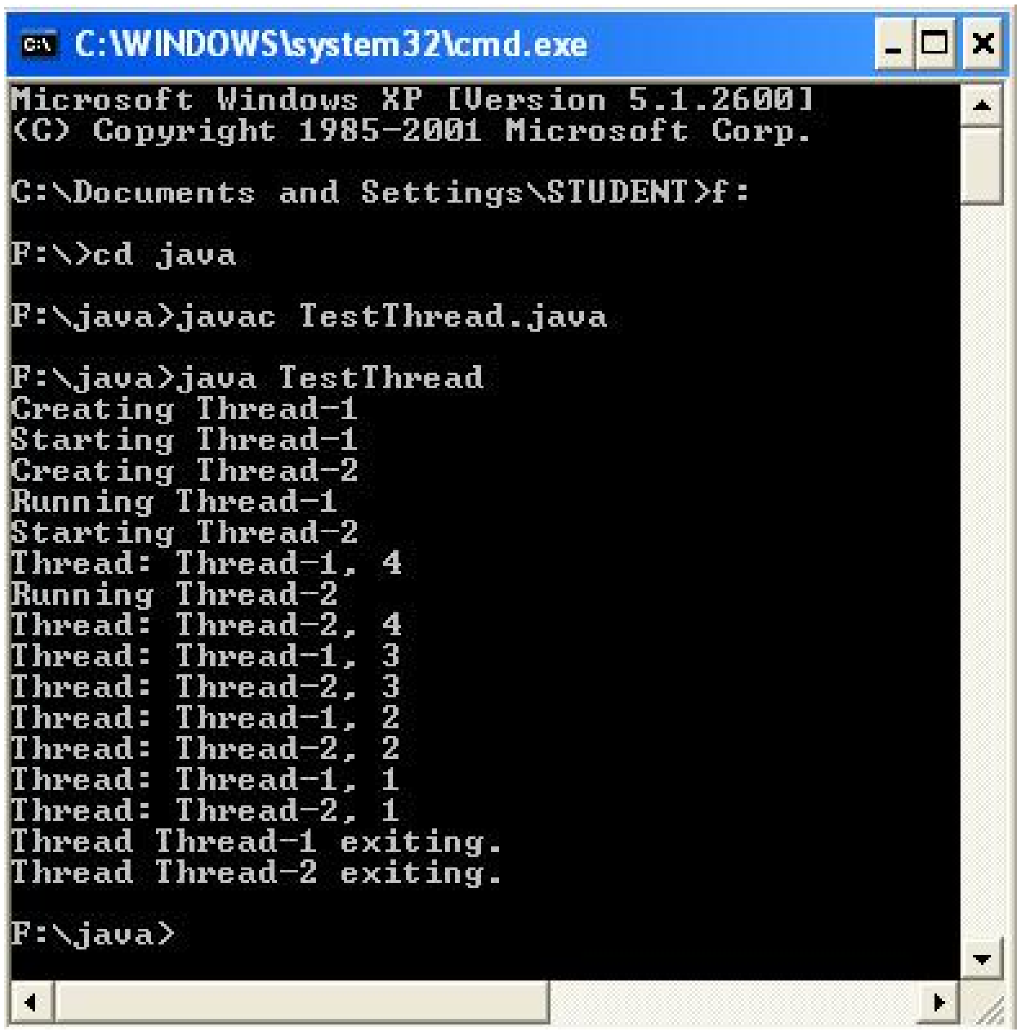
Step 7: Stop the program

PROGRAM: (MULTITHREADING)

```
class ThreadDemo extends Thread
{
private Thread t;
private String threadName;
ThreadDemo( String name)
{
threadName = name;
System.out.println("Creating " + threadName );
}
public void run() {
System.out.println("Running " + threadName );
try {
for(int i = 4; i > 0; i--)
 {
System.out.println("Thread: " + threadName + ", " +i );
Thread.sleep(50); // Let the thread sleep for a while.
}
}
catch (InterruptedException e) {
System.out.println("Thread " + threadName + " interrupted.");
}
System.out.println("Thread " + threadName + " exiting.");
}
public void start ()
{
System.out.println("Starting " + threadName );
if (t == null)
{
t = new Thread (this, threadName);
t.start ();
}
}
}
```

```
public class TestThread {
public static void main(String args[]) {
ThreadDemo T1 = new ThreadDemo( "Thread-1");
T1.start();
ThreadDemo T2 = new ThreadDemo( "Thread-2");
T2.start();
}
}
```

OUTPUT:



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\STUDENT>f:

F:\>cd java

F:\java>javac TestThread.java

F:\java>java TestThread
Creating Thread-1
Starting Thread-1
Creating Thread-2
Running Thread-1
Starting Thread-2
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.

F:\java>
```

RESULT

Thus the java program to implement multithreading was executed and verified

Ex. No: 18

Date :

EXCEPTION HANDLING

AIM:

To write a java program to handle exceptions

ALGORITHM:

Step 1. Start the program

Step 2. Define function divide () which perform division operation.

Step 3. Try to find with if there is any exception occurs within divide function.

Step 4. If exception occurs catch the exception and throw it to exception handler to take necessary action.

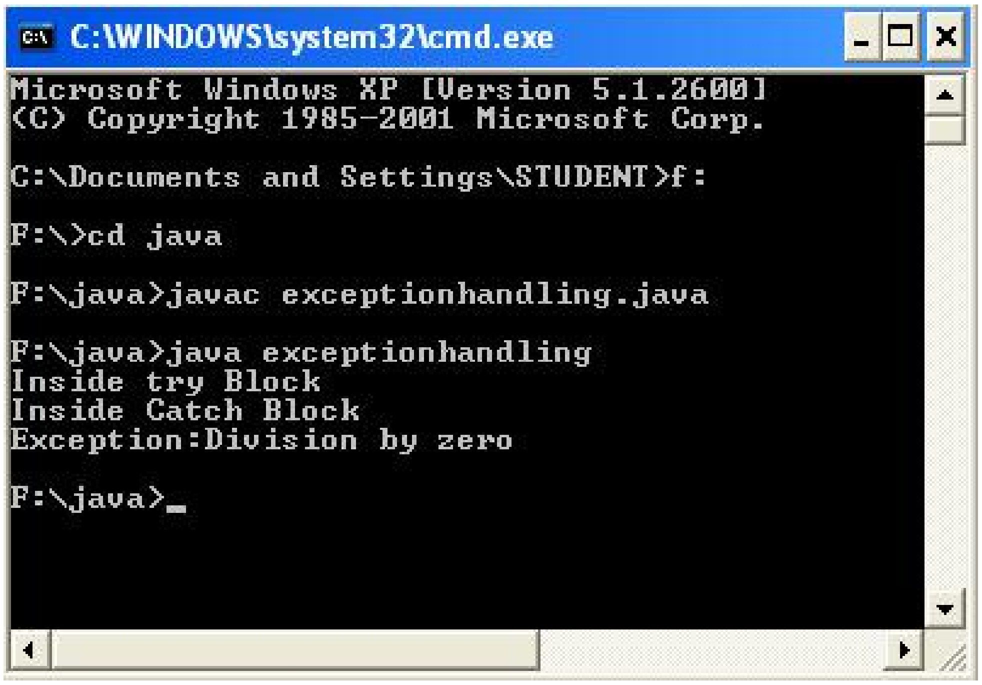
Step 5: Display the output

Step 6. Stop the program

PROGRAM: (EXCEPTION HANDLING)

```
class exceptionhandling
{
public static void main(String args[])
{
int d,a;
try
{
System.out.println("Inside try Block");
d=0;
a=42/d;
}
catch(ArithmeticException  e)
{
System.out.println("Inside Catch Block");
System.out.println("Exception:Division by zero");
}
}
}
```

OUTPUT:



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\STUDENT>f:

F:\>cd java

F:\java>javac exceptionhandling.java

F:\java>java exceptionhandling
Inside try Block
Inside Catch Block
Exception:Division by zero

F:\java>_
```

RESULT

Thus above program is executed and output is verified.

Ex. No: 19

Date :

USER DEFINED EXCEPTION

AIM:

To write a java program to handle user defined exceptions

ALGORITHM:

Step 1: Start the program

Step 2: Create a class My Exception that extends the class Exception

Step 3: Initialize the base class constructor using super

Step 4: Handle the exception by creating an object for the class My Exception

Step5: Use throw keyword to throw the exception

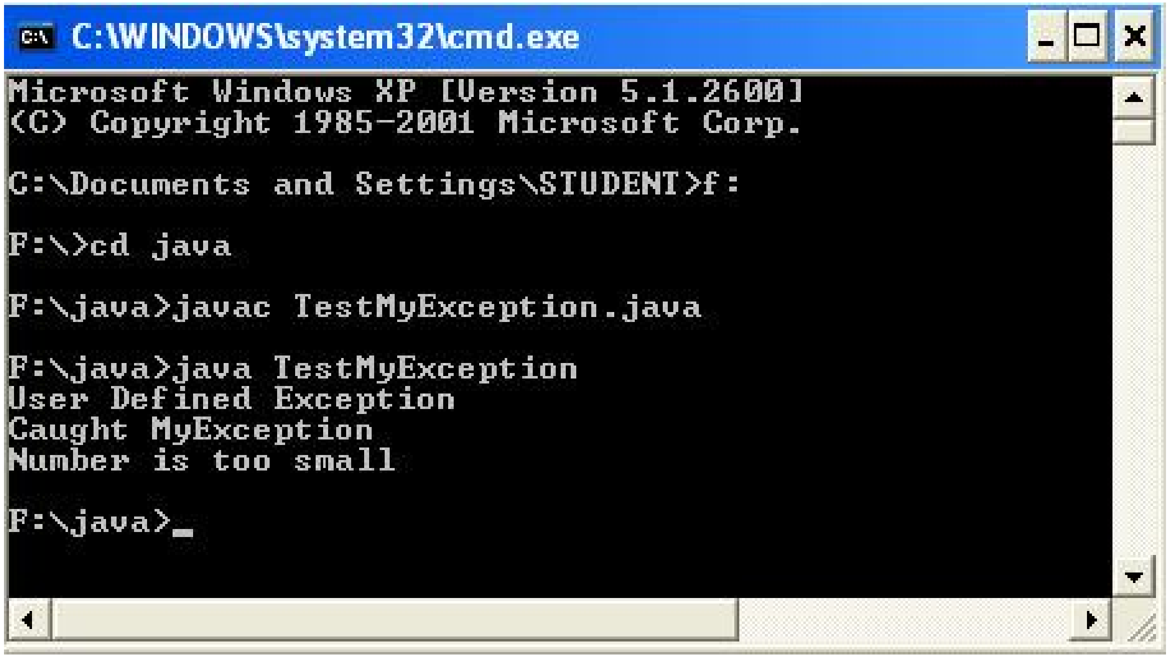
Step 6: Display the result

Step 7: Stop the program

PROGRAM: **(USER DEFINED EXCEPTION)**

```
import java.lang.Exception;
class MyException extends Exception
{
MyException (String message)
{
super (message);
}
}
class TestMyException
{
public static void main(String args[])
{
int x=5, y=1000;
System.out.println("User Defined Exception");
try
{
float z=(float)x/(float)y;
if(z<0.1)
{
throw new MyException ("Number is too small");
}
}
catch (MyException e)
{
System.out.println("Caught MyException");
System.out.println(e.getMessage());
}
}
}
```

OUTPUT:



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\STUDENT>f:

F:\>cd java

F:\java>javac TestMyException.java

F:\java>java TestMyException
User Defined Exception
Caught MyException
Number is too small

F:\java>_
```

RESULT

Thus above java program to handle user defined exception is executed and output is verified.