SONA COLLEGE OF ARTS AND SCIENCE
Affiliated to Periyar University, Salem -11

DEPARTMENT OF COMPUTER APPLICATIONS

Practical Record

Name : _____

Register Number : _____

Subject Code : 21UCAP02

Subject Title : PRACTICAL II : C++ PROGRAMMING LAB

Year / Sem : I / II

ACADEMIC YEAR: 2021 – 2022

SONA COLLEGE OF ARTS AND SCIENCE
Affiliated to Periyar University, Salem

Certificate

This is to Certify that the Practical Record **"Practical – II: C++ Programming Lab"** is a bonafide work done by _____

Reg. No_____submitted to the Department of Computer Applications, during the academic year 2021 – 2022.

SUBJECT IN-CHARGE **HEAD OF THE DEPARTMENT**

Submitted for University Practical Examination held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

INDEX

Ex.No: 1	FUNCTION OVERLOADING, DEFAULT ARGUMENTS AND INLINE FUNCTION
Date:	

AIM:

To write a C++ program to demonstrate function overloading, default arguments and Inline Functions to perform addition.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the data members and member functions with different data types.

STEP 3: Get the input in different data types as declared.

STEP 4: Define the function prototypes to perform function overloading.

STEP 5: Declare and define the inline function with default arguments.

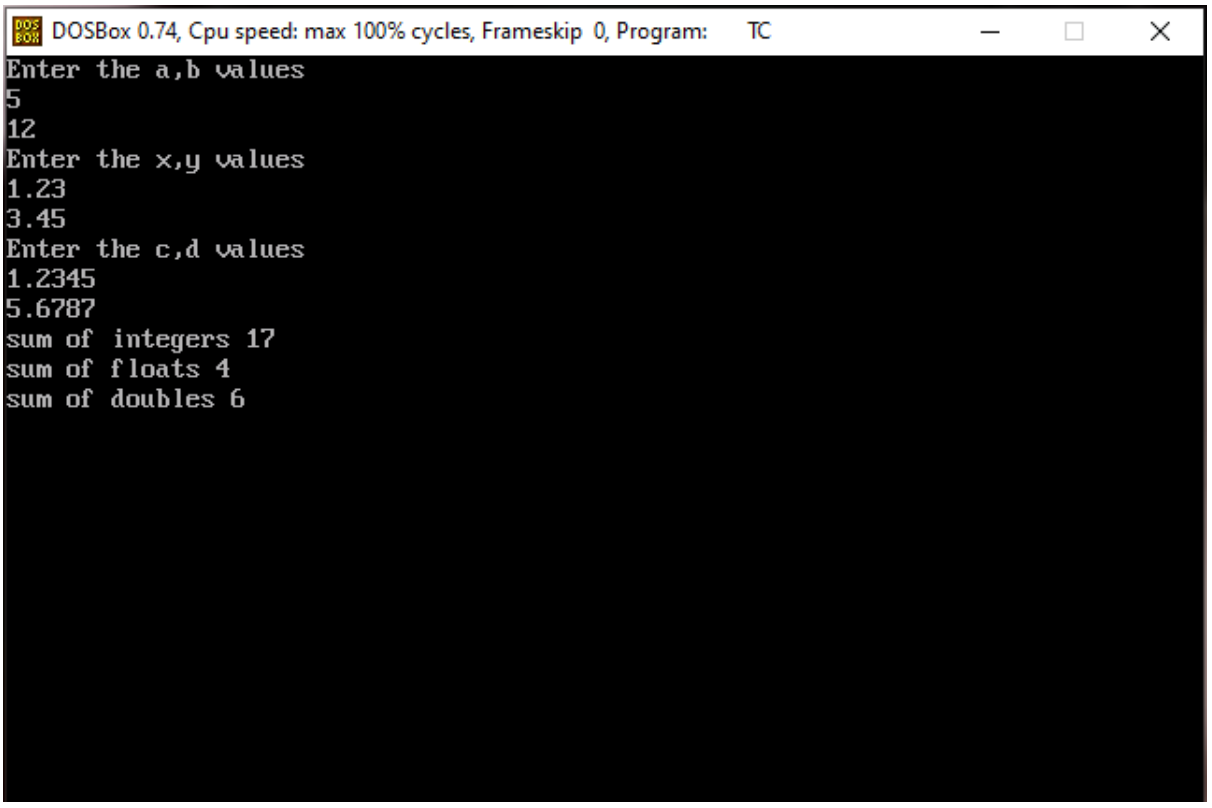
STEP 6: Display the result for the respective inputs

STEP 7: Stop the program.

CODING:

```
# include<iostream.h>
# include<conio.h>
void main()
{
int add(int,int);//Function prototype
float add(float,float);
double add(double,double);
int a,b;
float x,y;
double c,d;
cout<<"Enter the a,b values "<<endl;
cin>>a>>b;
cout<<"Enter the x,y values "<<endl;
cin>>x>>y;
cout<<"Enter the c,d values "<<endl;
cin>>c>>d;
cout<<"sum of integers "<<add(a,b)<<endl;
cout<<"sum of floats "<<add(x,y)<<endl;
cout<<"sum of doubles "<<add(c,d)<<endl;
getch();
}
int add(inta,int b)
{    return(a+b);
}
float add(float x,float y)
{    return(x+y);
}
inline double add(double c,double d)
{   return(c+d);
}
```

OUTPUT:



RESULT:

Thus the above program has been compiled and executed successfully.

Ex.No: 2	CLASS AND OBJECTS WITH THE CONCEPT OF PASSING OBJECTS IN FUNCTIONS
Date:	

AIM:

To write a C++ program to demonstrate class and objects with the concept of passing objects in functions.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the class name with data members and member functions.

STEP 3: Define the functions with corresponding data members.

STEP 4: The function get time() receives the input.

STEP 5: The function put time () displays the output.

STEP 6: The function sum () will perform the needed operations to calculate time, hours, minutes by passing the objects as arguments.

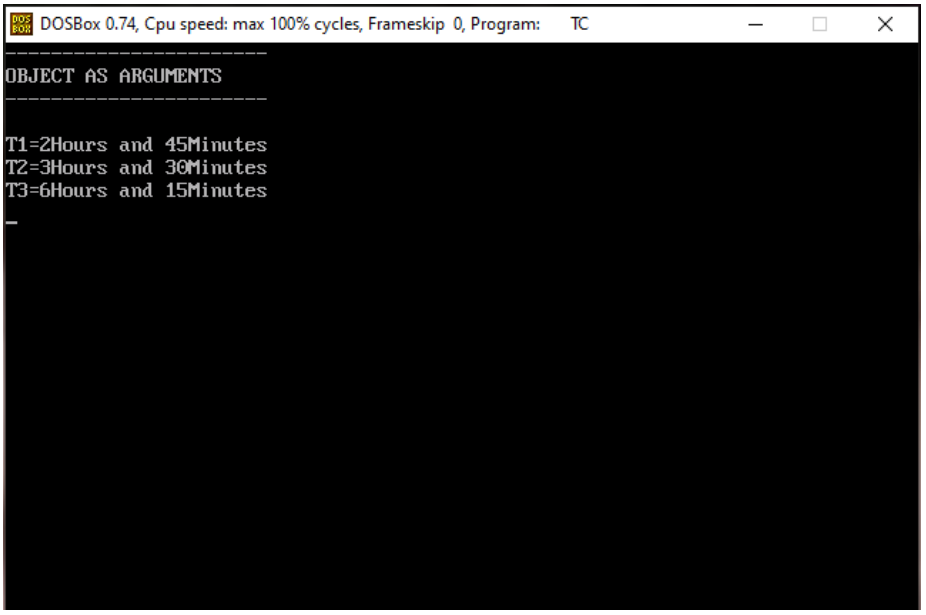
STEP 7: Stop the program.

CODING:

```
# include<iostream.h>
# include<conio.h>
class time
{
int hours;
int minutes;
public:
void gettime(int h, int m)
{
hours=h;
minutes=m;
}
void puttime(void)
{
cout<<hours<<"Hours and"<<endl;
cout<<minutes<<"Minutes"<<endl;
}
void sum(time,time);
};
void time::sum(time t1,time t2)
{
minutes = t1.minutes+t2.minutes;
hours = minutes/60;
minutes = minutes = minutes%60;
hours = hours+t1.hours+t2.hours;
}
const int size=2;
void main()
{
clrscr();
time t1[size],t2;
t1[1].gettime(2,45);
t1[2].gettime(3,30);
t2.sum(t1[1],t1[2]);
cout<<"-------"<<endl;
cout<<"OBJECT AS ARGUMENTS"<<endl;
```

```
cout<<"------\n"<<endl;
cout<<"T1=";
t1[1].puttime();
cout<<"T2=";
t1[2].puttime();
cout<<"T3=";
t2.puttime();
getch();
}
```

OUTPUT:



RESULT:

Thus the above program has been compiled and executed successfully.

Ex.No: 3	CONSTRUCTOR AND DESTRUCTOR
Date:	

AIM:

To write a C++ program to demonstrate constructor and destructor.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the class name as book with data members and member functions.

STEP 3: Create a constructor as book() and define them using string functions.

STEP 4: Create a destructor as ~book() and define then to display the destructor as invoked.

STEP 5: The function disp () will display the book details.

STEP 6: Create the object to corresponding class and display the output by calling concern function.

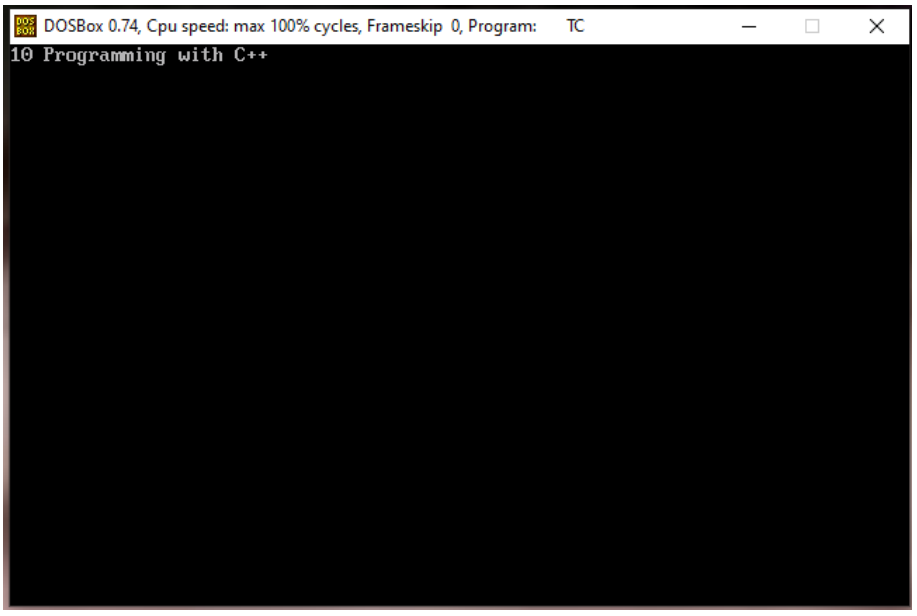
STEP 7: The function area () to find area of rectangle with two integer argument.

STEP 8: Stop the progr

CODING:

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
#include<string.h>
class book
{
private:
  int book_no;
 char book_name[10];
public:
  book()      //constructor
  {
   book_no=10;
   strcpy(book_name,"Programming with C++");
   }
    ~book()    //destructor
   {    book_no=0;
        strcpy(book_name,"");
        cout<<"destructors invoked"<<endl;
        disp();
   }
void disp()
{
cout<<book_no<<setw(5)<<book_name<<endl;
 }   };
       void main()
       {
book s1;
clrscr();
s1.disp();
getch();
}
```

OUTPUT:



RESULT:

Thus the above program has been compiled and executed successfully.

Ex.No: 4	UNARY AND BINARY OPERATOR OVERLOADING
Date:	

AIM:

To write a C++ program to demonstrate unary and binary operator overloading.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the class name as ool with data members and member functions.

STEP 3: Define the function ool () using string copy functions.

STEP 4: Define the function to perform binary operator overloading.

STEP 5: The function show () will display the result for binary operator overloading.

STEP 6: Define the function to perform unary operator overloading.

STEP 5: The function show () will display the result for unary operator overloading to find fact value.

STEP 8: Create an object for the class and call the corresponding functions to display the result.

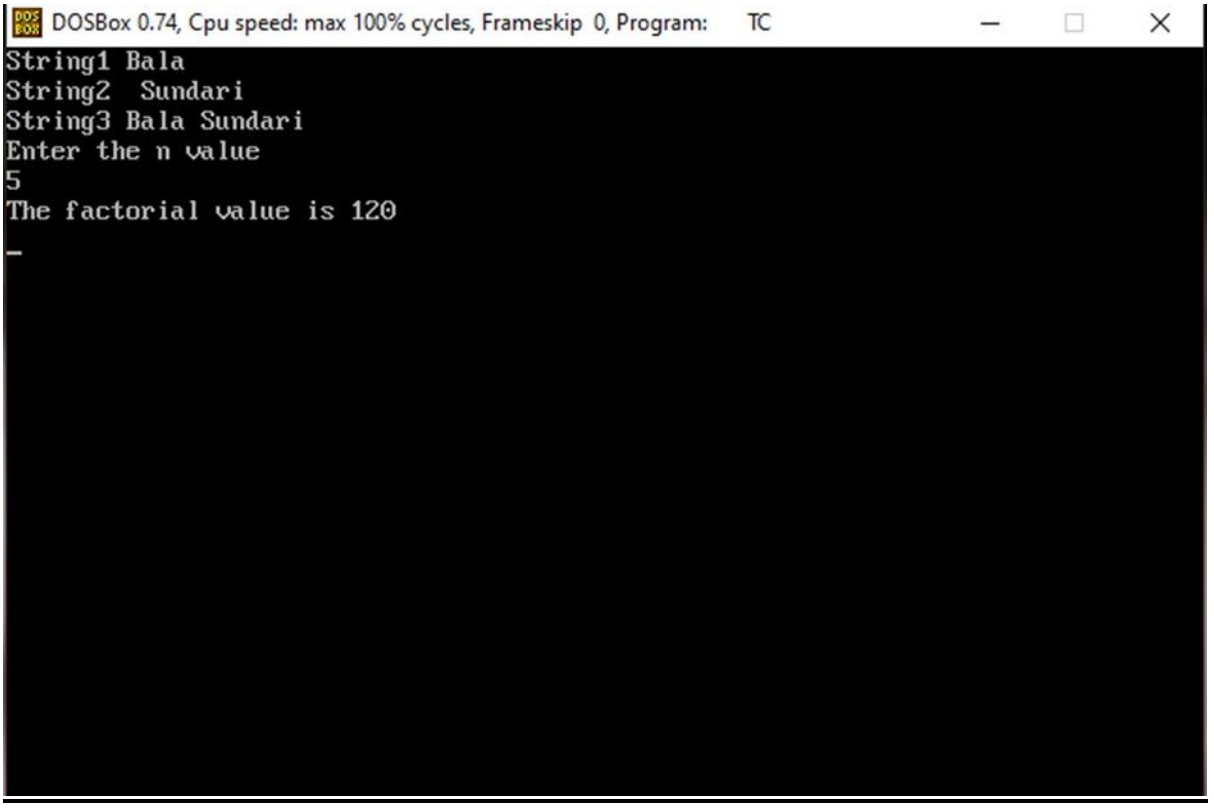
STEP 9: Stop the program

CODING:

```
#include<iostream.h>
#include<string.h>
#include<conio.h>
class ool
{
private:
     char str[30];
     int n, fac;
public:
ool()
   {
strcpy(str,"");
fac=1;
   }
void operator = (char*cptr)
   {
strcpy(str,cptr);
   }
void operator--();
void display();
ool operator+(ool t1) // Binary operator overloading
   {
ool t2;
strcpy(t2.str,str);
strcat(t2.str,t1.str);
return t2;
   }
void show_string(char*obj)
   {
cout<<obj<<str<<endl;
   }
};
void ool::operator--() // Unary operator overloading
    {
cout<<"Enter the n value "<<endl;
cin>>n;
```

```
for(int i=1;i<=n;i++)
fac*=i;
    }
void ool::display()
{
cout<<"The factorial value is "<<fac<<endl;
}
void main()
{
clrscr();
ool s1,s2,s3,obj;
s1="Bala";
s2=" Sundari";
s3=s1+s2;
s1.show_string("String1 ");
s2.show_string("String2 ");
s3.show_string("String3 ");
--obj;
obj.display();
getch();
}
```

OUTPUT:



RESULT:

Thus the above program has been compiled and executed successfully.

Ex.No: 5(i)	SINGLE INHERITANCE
Date:	

AIM:

To write a C++ Program to demonstrate single inheritance.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the base class.

STEP 3: Declare and define the function.

STEP 4: Declare the other class derive.

STEP 5: Declare and define the function.

STEP 6: Declare the derived class with functions.

STEP 7: Defined the functions to perform operations.

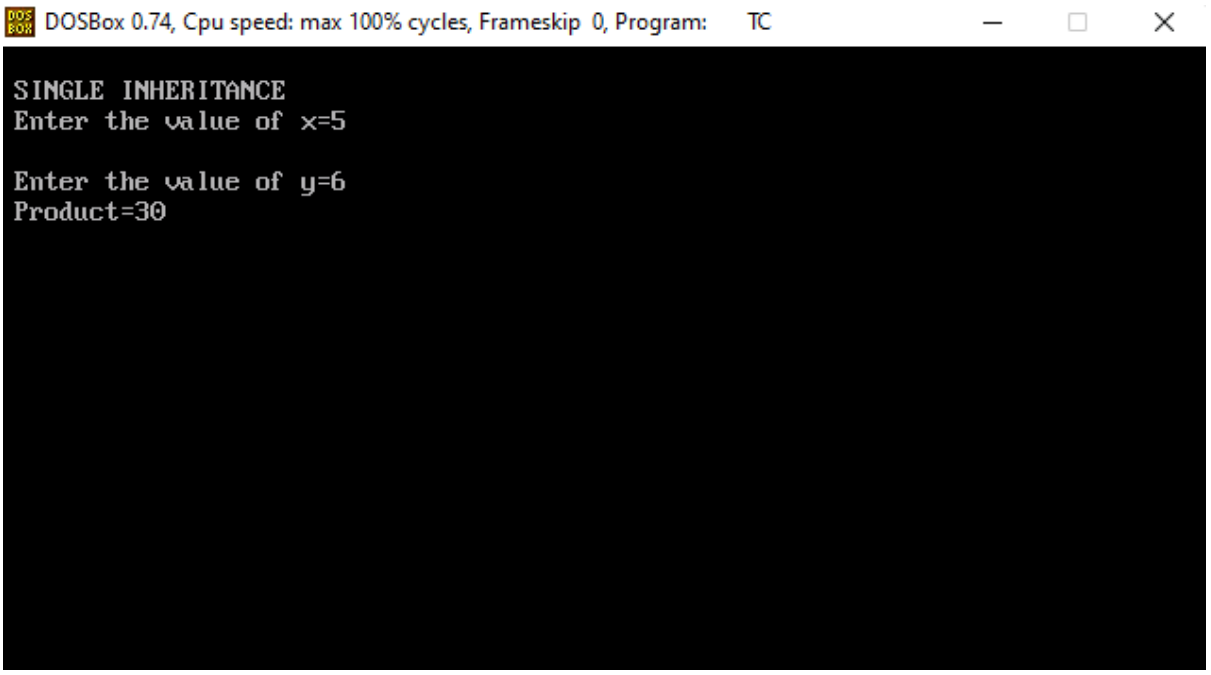
STEP 8: Create the objects and called the functions getdata(),read data(),product().

STEP 9: Stop the program.

CODING:

```
#include <iostream>
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>> y;
  }
void product()
  {
cout<< "Product = " << x * y;
  }
};
int main()
 {
derive a;     //object of derived class
a.getdata();
a.readdata();
a.product();
return 0;
 }
```

OUTPUT:



RESULT:

Thus the above program has been compiled and executed successfully.

Ex.No: 5(ii)	MULTILEVEL INHERITANCE
Date:	

AIM:

To find out the product of given values using multilevel inheritance.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the base class as base.

STEP 3: Declare the variables and define the function.

STEP 4: Declare the other class derive1.

STEP 5: Declare and define the function.

STEP 6: Declare the derived class with functions.

STEP 7: Declare the other class derive2.

STEP 8: Declare the derived class with functions.

STEP 8: Create the objects and called the functions get data(),read data(),indata() and product() to display the result.

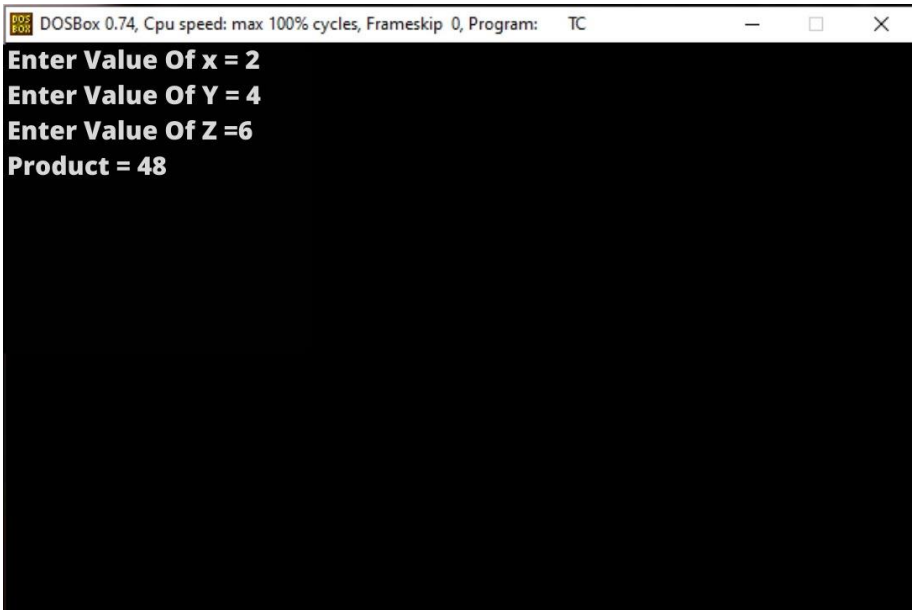
STEP 9: Stop the program.

CODING:

```
#include <iostream>
using namespace std;
class base //single base class
{
  public:
  int x;
  voidgetdata()
  {
  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= ";
   cin>> y;
  }
};
class derive2 : public derive1   // derived from class derive1
{
  private:
  int z;
  public:
  void indata()
  {
  cout<< "\n Enter value of z= "; cin>> z;
  }
  void product()
  {
      cout<< "\n Product= " << x * y * z;
  }
};
```

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

OUTPUT:



RESULT:

Thus the above program has been compiled and executed successfully.

Ex.No: 5(iii)	MULTIPLE INHERITANCE
Date:	

AIM:

To find out the student details using multiple inheritance.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the base class student.

STEP 3: Declare and define the function get () to get the student details.

STEP 4: Declare the other class sports.

STEP 5: Declare and define the function getsm () to read the sports mark.

STEP 6: Create the class statement derived from student and sports.

STEP 7: Declare and define the function display () to find out the total and average.

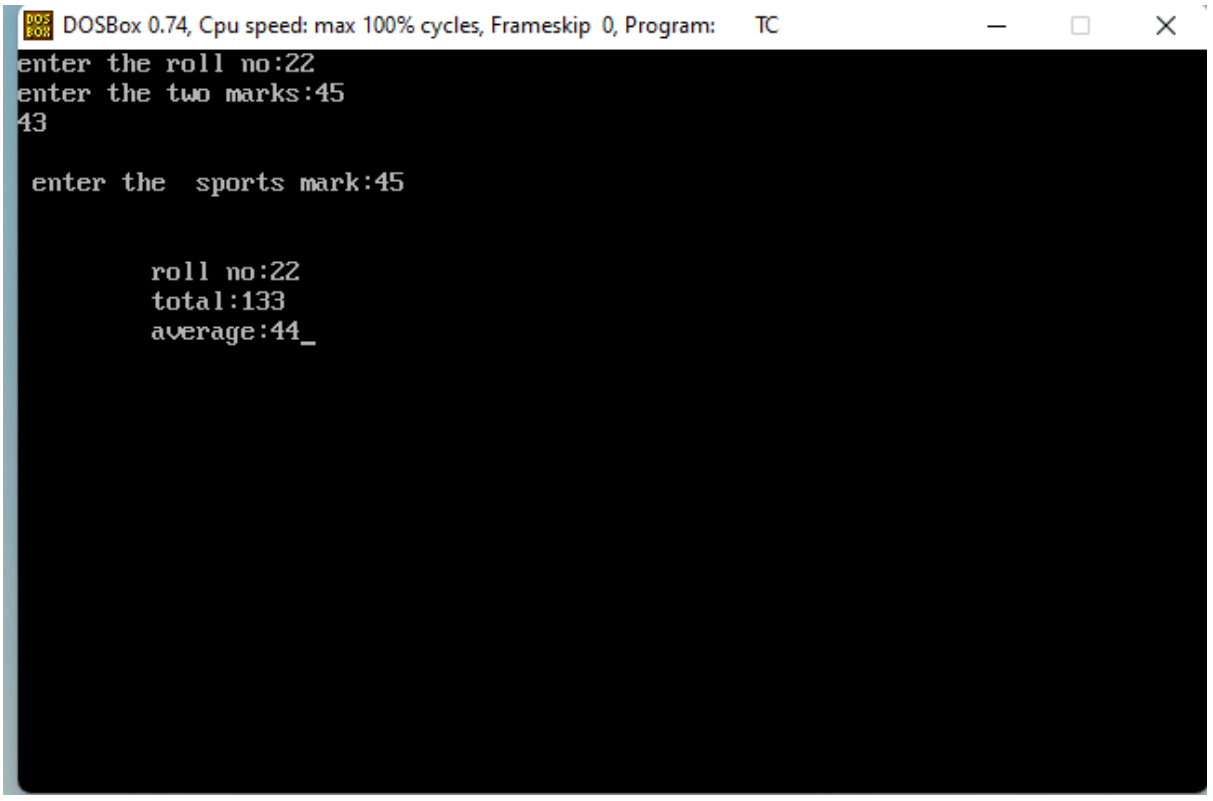
STEP 8: Declare the derived class object,call the functions get (),getsm () and display ().

STEP 9: Stop the program.

CODING:

```
# include<iostream.h>
# include<conio.h>
class student
{  protected:
int rno, m1, m2;
public:
void get() {
cout<<"Enter the Roll no :";
cin>>rno;
cout<<"Enter the two marks   :";
cin>> m1>>m2;
   }   };
class sports
{ protected:
int sm; // sm = Sports mark
public:
void getsm() {
cout<<"\n Enter the sports mark :";
cin>>sm;
   }  };
class statement : public student, public sports
{ int tot, avg;
public:
void display()
{ tot = (m1 + m2 + sm);
avg = tot / 3;
cout<<"\n\n\tRoll No    : "<<rno<<"\n \t Total      : "<< tot;
cout<<"\n\tAverage    : "<<avg;
   }    };
void main()
{ clrscr();
statement obj;
obj.get();
obj.getsm();
obj.display();
getch();
}
```

OUTPUT:



RESULT:

Thus the above program has been compiled and executed successfully.

Ex.No: 5(iv)	HIERARCHICAL INHERITANCE
Date:	

AIM:

To find out the sum and product values of given numbers using hierarchical inheritance.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the base class A.

STEP 3: Declare and define the function getdata () to get the values of x and y.

STEP 4: Declare the other class B.

STEP 5: Declare and define the function product () to display the product values.

STEP 6: Create the class C and define the function Sum() to display the sum values.

STEP 7: Create the object for derived classes as B and C

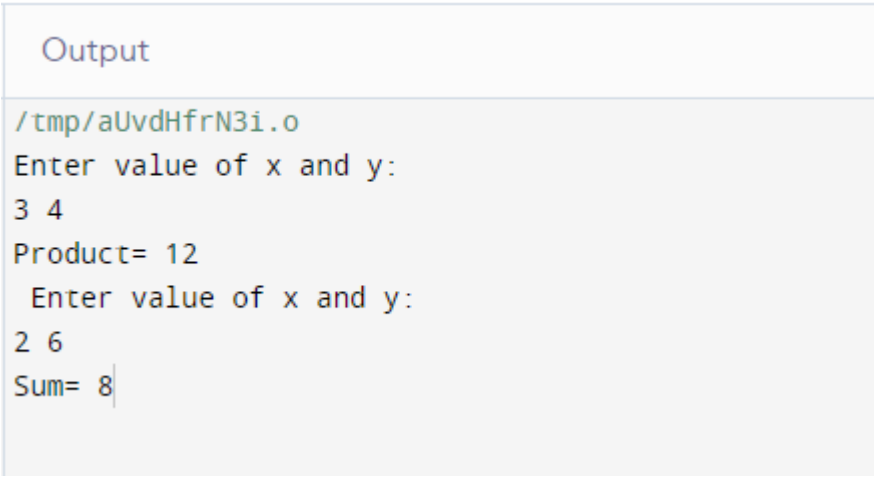
STEP 8: Call the functions getdata(), product() and sum().

STEP 9: Stop the program.

CODING:

```
# include <iostream>
using namespace std;
class A //single base class
 {
 public:
 int x, y;
 void getdata()
 {
        cout<< "\n Enter value of x and y:\n";
        cin>> x >> y;
 } };
class B : public A //B is derived from class base
{
public:
  void product()
  {
  cout<< "\n Product= " << x * y;
  } };
 class C : public A //C is also derived from class base
 {   public:
      void sum()
  {  cout<< "\n Sum= " << 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
```

OUTPUT:



RESULT:

Thus the above program has been compiled and executed successfully.

Ex.No: 5(v)	HYBRID INHERITANCE
Date:	

AIM:

To find out the sum values using hybrid inheritance.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the base class as class A.

STEP 3: Declare the variables.

STEP 4: Declare the other class derive1.

STEP 5: Declare the other class B and inherited from base class A.

STEP 6: Declare the derived class with functions and initialized the constructor.

STEP 7: Declare the other class C.

STEP 8: Declare the derived class with functions, and initialized the constructor.

STEP 9: Declare the other class as D and inherited from class B and C.

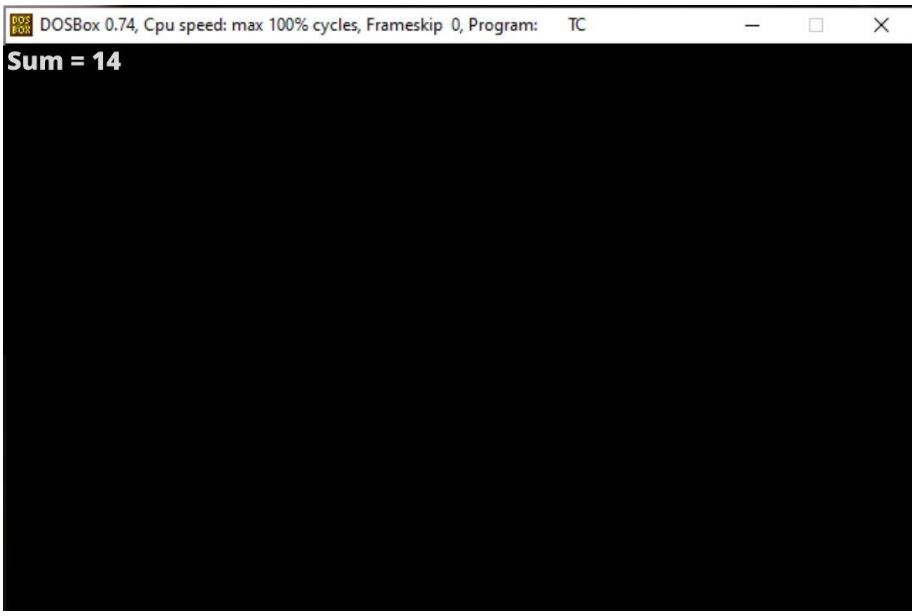
STEP 8: Create the objects and called the function sum() to display the result.

STEP 9: Stop the program.

CODING:

```
# include <iostream>
using namespace std;
class A
{
  public:
  int x;
};
class B : public A
{
  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
```

OUTPUT:



RESULT:

Thus the above program has been compiled and executed successfully.

Ex.No: 6	VIRTUAL FUNCTIONS
Date:	

AIM:

To write a C++ program to demonstrate virtual function.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Create a class as sample.

STEP 3: Declare the variables.

STEP 4: Declare the functions as getdata(), calc(), display().

STEP 5: Define the functions to perform concern operations.

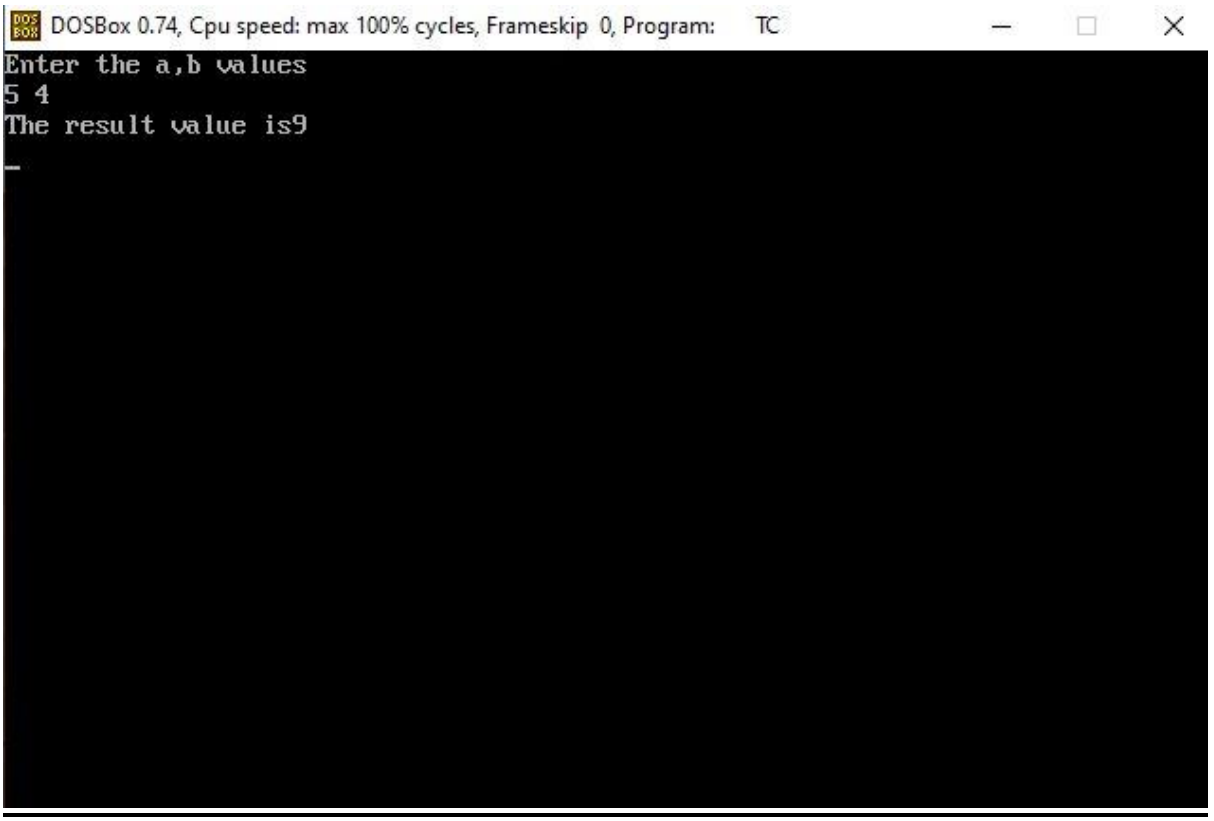
STEP 6: Create the objects and perform virtual functions to display the result.

STEP 7: Stop the program.

CODING:

```
#include<iostream.h>
#include<conio.h>
class sample
{
private:
inta,b,res;
public:
virtual void getdata();
virtual void calc();
virtual void display();
};
void sample::getdata()
{
cout<<"Enter the a,b values "<<endl;
cin>>a>>b;
}
void sample::calc()
{
res=a+b;
}
void sample::display()
{
cout<<"The result value is "<<res<<endl;
}
void main()
{
clrscr();
sampleobj;
sample *obj1;
    obj1=&obj;
    obj1->getdata();
    obj1->calc();
    obj1->display();
getch();
}
```

OUTPUT:



RESULT:

Thus the above program has been compiled and executed successfully.

Ex.No: 7	MANIPULATE A TEXT FILE
Date:	

AIM:

To write a C++ program to manipulate a text file.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the header files to manipulate a text file.

STEP 3: Declare the source file and target file.

STEP 4: Create a source file to perform open operation.

STEP 5: Check the condition where the file is exists or not.

STEP 6: Give the target file to manipulate it.

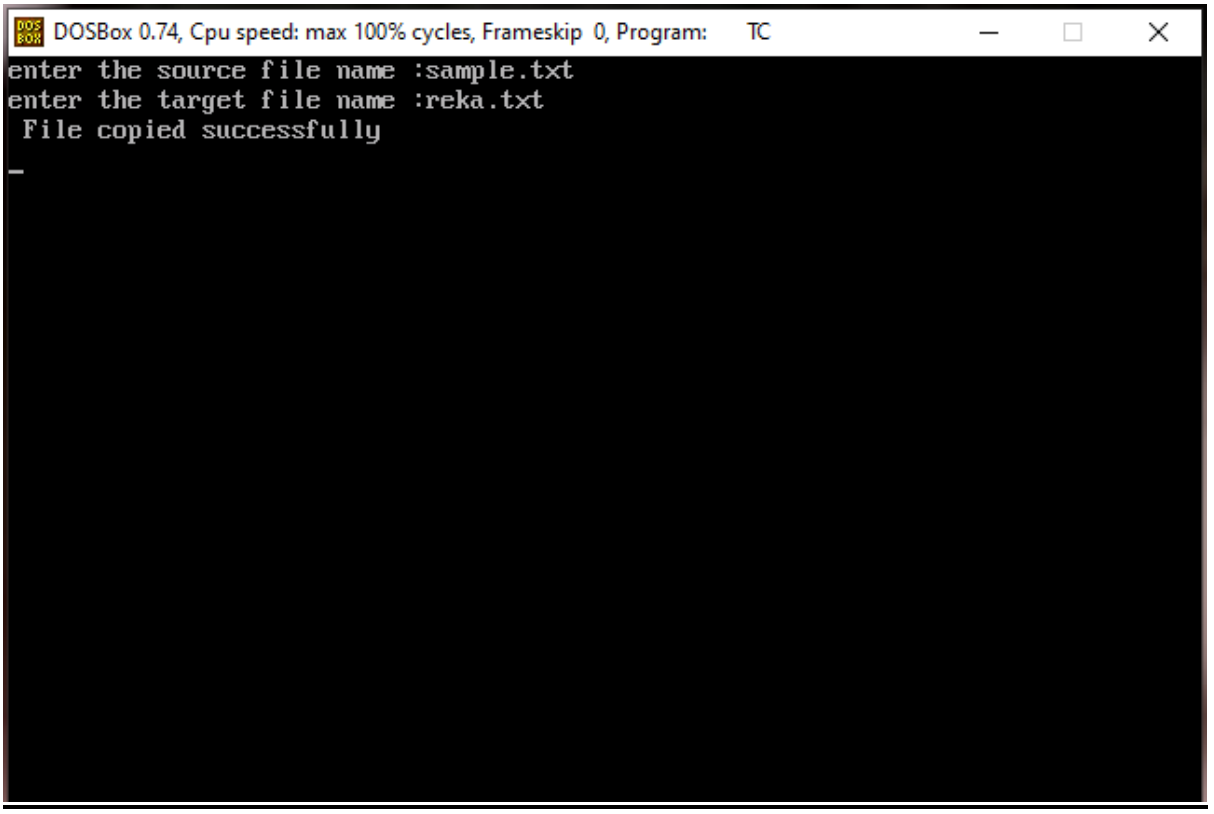
STEP 7: Check the file is copied or not.

STEP 8: Stop the program.

CODING:

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
#include<stdlib.h>
void main()
{
ofstream outfile;
ifstream infile;
char source [10],target[10];
char ch;
clrscr();
cout<<"enter the source file name :";
cin>>source;
cout<<"enter the target file name :";
cin>>target;
infile.open(source);
if(infile.fail())
{
  cout<<"source file does not exist ";
  exit(1);
}
  outfile.open(target);
  if(outfile.fail())
 {
    cout<<"unable to create file";
    exit(1);
 }
    while(!infile.eof())
 {
     ch=infile.get();
    outfile.put(ch);
  }
    infile.close();
    outfile.close();
   getch();
}
```

OUTPUT:



RESULT:

Thus the above program has been compiled and executed successfully.

Ex.No:8	SEQUENTIAL I/O OPERATIONS ON A FILE
Date:	

AIM:

To write a C++ program to demonstrate sequential I/O operations on a text file.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Include the header files to perform file operations.

STEP 3: Include the header files to perform file input / output operations.

STEP 4: Declare object for input and output file.

STEP 5: Create a file using open() and copy the content.

STEP 6: Open the file to read the data and copy the data.

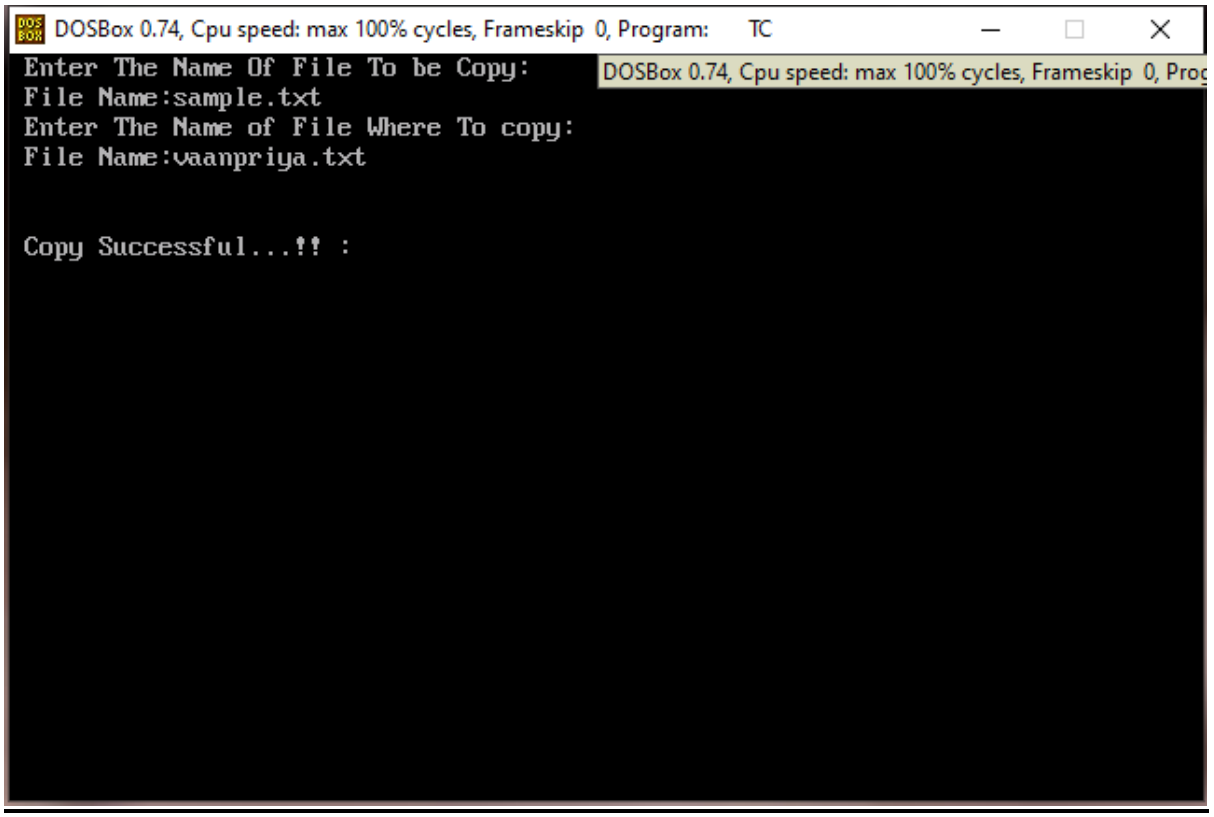
STEP 7: Check the file is copied or not and close the file using close ().

STEP 8: Stop the program.

CODING:

```
#include<fstream.h>              //required for file operation:
#include<iostream.h>             //required for input/output:
#include<conio.h>
void main()
{
char File1[20],File2[20],ch;       //variable to store file-Names
ifstream in_file; //Object for Input file
ofstream out_file; //Object for Output file
 cout<<" Enter The Name Of File To be Copy: \n";
cout<<" File Name:";
cin>>File1;
cout<<" Enter The Name of File Where To copy: \n";
cout<<" File Name:";
cin>>File2;
in_file.open(File1,ios::in);                    //file Open for reading purpose: (copy)
if(!in_file) //Error Checker: if file not open:
   {
      cout<<"An Error Occur While Opening File: \n";
   }
out_file.open(File2,ios::app);                  //file Open for writing purpose: (paste)
if(!out_file) //Error Checker: if file not open:
   {
      cout<<"An Error Occur While Opening File: \n";
   }
while (in_file.get(ch))                         //read data from file: (copy data)
   {
      out_file.put(ch);                         //paste it here:
   }
in_file.close();                                //closing files:
out_file.close();
cout<<" \n\n Copy Successful...!! : \n";
getch();
}
```

OUTPUT:



RESULT:

Thus the above program has been compiled and executed successfully.

Ex.No:9	BIGGEST NUMBER USING COMMAND LINE ARGUMENTS
Date:	

AIM:

To write a C++ program to perform biggest elements using command line arguments.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the variables.

STEP 3: Check the loop to store the largest number.

STEP 4: Check the condition to find the largest number.

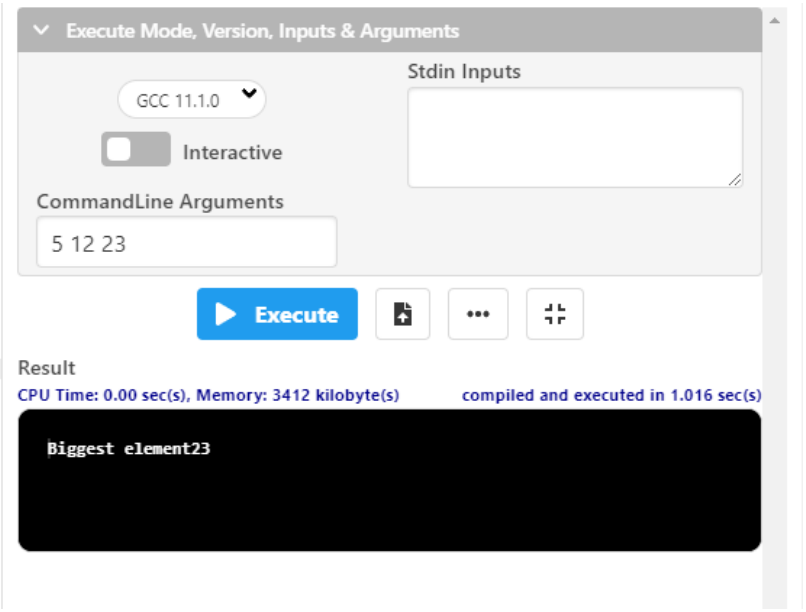
STEP 5: Display the biggest number using command line arguments.

STEP 6: Stop the program.

CODING:

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
void main(int argc, char * argv[])
{
int i, n, big;
clrscr();
   n = argc;
if(argc == 1)
        exit(0);
big = atoi(argv[1]);
   // Loop to store largest number to large
for(i = 2; i< n; i++)
   {
if (big <atoi(argv[i]))
        big = atoi(argv[i]);
   }
cout<<"Biggest element" <<big;
getch();
}
```

OUTPUT :



RESULT:

Thus the above program has been compiled and executed successfully.

Ex.No:10	CLASS TEMPLATES
Date:	

AIM:

To write a C++ program to demonstrate the concept of Class Templates

ALGORITHM:

STEP 1: Start the program.

STEP 2: Create a class as sample..

STEP 3: Declare the variables in private mode.

STEP 4: Declare the functions getdata(), sum() in public mode.

STEP 5: Declare the template class as class T.

STEP 6: Create the object for the class sample.

STEP 7: Call the function using objects.

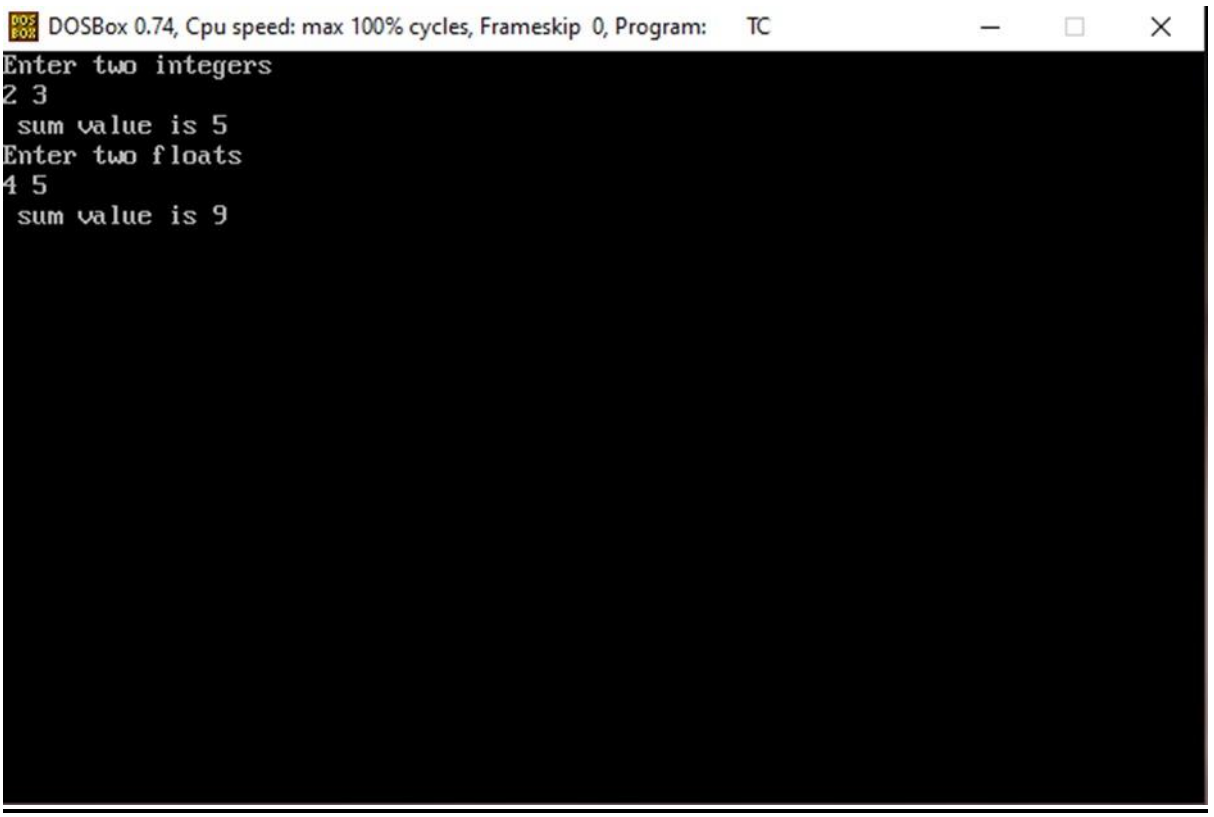
STEP 8: Display the result.

STEP 9: Stop the program.

CODING:

```
#include<iostream.h>
#include<conio.h>
template<class T>
class sample
{
private:
        T value,value1,value2;
 public:
voidgetdata();
void sum();
  };
template<class T>
void sample <T>::getdata()
{
cin>>value1>>value2;
}
template<class T>
void sample <T>::sum()
{
value=value1+value2;
cout<<" sum value is "<<value<<endl;
}
void main()
{
clrscr();
sample<int> obj1;
sample<float> obj2;
cout<<"Enter two integers "<<endl;
obj1.getdata();
obj1.sum();
cout<<"Enter two floats "<<endl;
obj2.getdata();
obj2.sum();
getch();
}
```

OUTPUT:



RESULT:

Thus the above program has been compiled and executed successfully.

Ex.No:11	EXCEPTION HANDLING
Date:	

AIM:

To write a C++ program to demonstrate the concept of Exception handling.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the variables.

STEP 3: Check the conditions in try block.

STEP 4: If there is any exception use throw and catch block to display it.

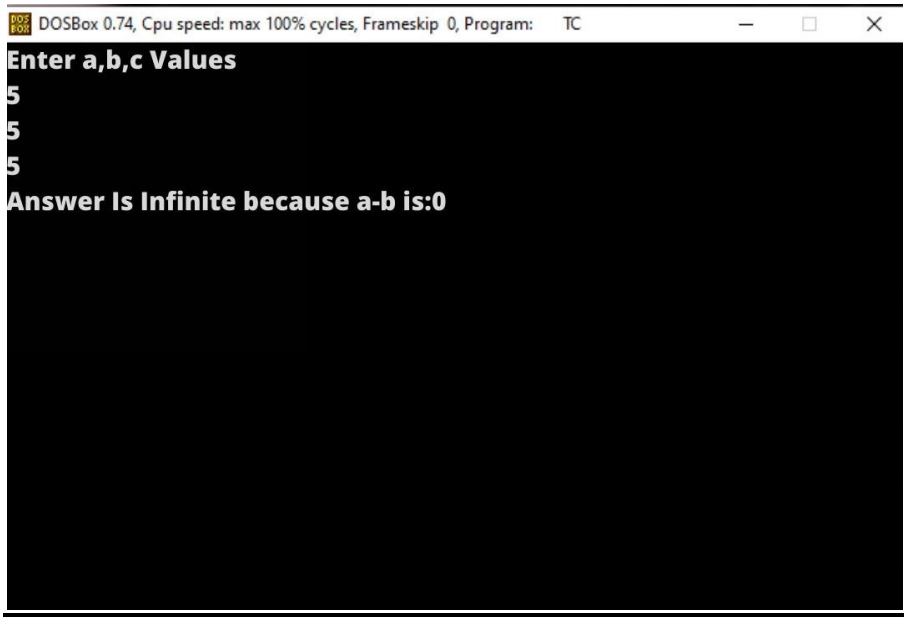
STEP 5: Display the result.

STEP 6: Stop the program.

CODING:

```
# include<iostream>
using namespace std;
int main()
{
int a,b,c;
float d;
cout<<"Enter a,b,c values";
cin>>a>>b>>c;
try
{
if((a-b)!=0)
{
d=c/(a-b);
cout<<"Result is"<<d;
}
else
throw(a-b);
}
catch(int i)
{
cout<<"Answer is infinite because a-b is:"<<i;
}
return 0;
}
```

OUTPUT :



RESULT:

Thus the above program has been compiled and executed successfully.