Object Oriented Programming Lab

BCS-DS-352

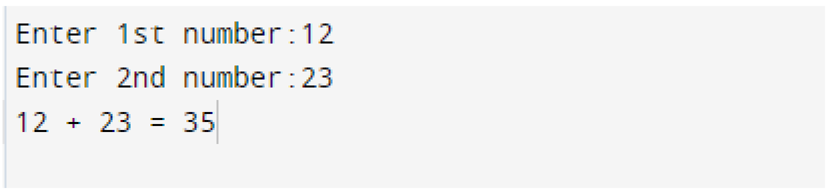
Program 1

Aim: C++ program to add two numbers.

Code:

```cpp
#include <iostream>
using namespace std;
int main(){
    int a,b;
    cout<<"Enter 1st number:";
    cin>>a;
    cout<<"Enter 2nd number:";
    cin>>b;
    cout<<a<<" + "<<b<<" = "<<a+b;
    return 0;
}
```

Output:



```
Enter 1st number:12
Enter 2nd number:23
12 + 23 = 35
```

Program 2

Aim: Write a function called power that takes double values for n and an int value for p. It returns a double value. Use a default argument of 2 for p so that if this argument is omitted the number will be squared. Write a main function that gets value from the user to test this function.

Code:

```
#include <iostream>
using namespace std;
double power(double base,int pow=2){
    double a=1;
    for (int i=0;i<pow;i++){
        a*=base;
    }
    return a;
}

int main(){
    double base;
    int p;
    cout<<"Enter base:";
    cin>>base;
    cout<<"Enter power:";
    cin>>p;
    cout<<base<<" ^ "<<p<<" = "<<power(base,p)<<endl;
    return 0;
}
```

Output:

```
Enter base:12
Enter power:3
12 ^ 3 = 1728

Enter base:2.1
Enter power:2
2.1 ^ 2 = 4.41
```

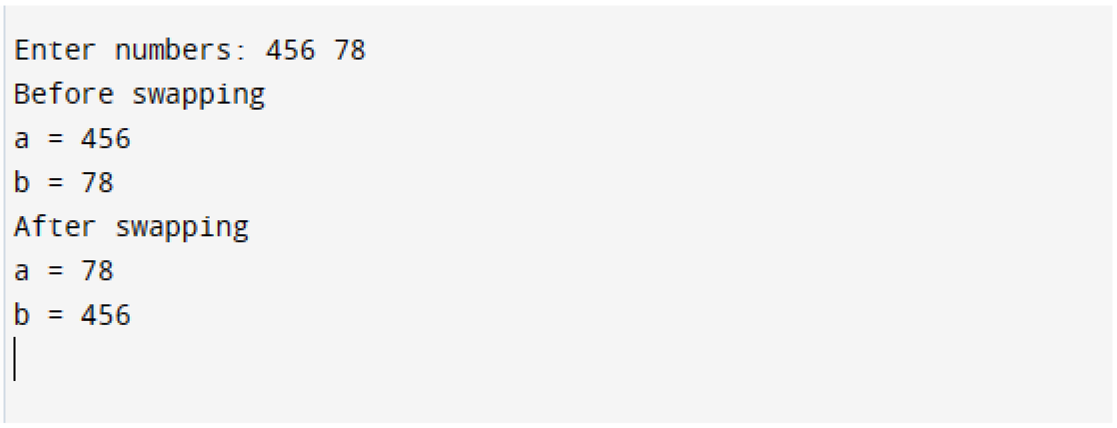
Program 3

Aim: C++ program to Swap two values using call by value.

Code:

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

void swap(int a, int b){
    int temp=a;
    a=b;
    b=temp;
    cout<<"a = "<<a<<endl;
    cout<<"b = "<<b<<endl;
}

int main(){
    int a,b;
    cout<<"Enter numbers: ";
    cin>>a>>b;
    cout<<"Before swapping"<<endl;
    cout<<"a = "<<a<<endl;
    cout<<"b = "<<b<<endl;
    cout<<"After swapping"<<endl;
    swap(a,b);
    return 0;
}
```

Output:



```
Enter numbers: 456 78
Before swapping
a = 456
b = 78
After swapping
a = 78
b = 456
```

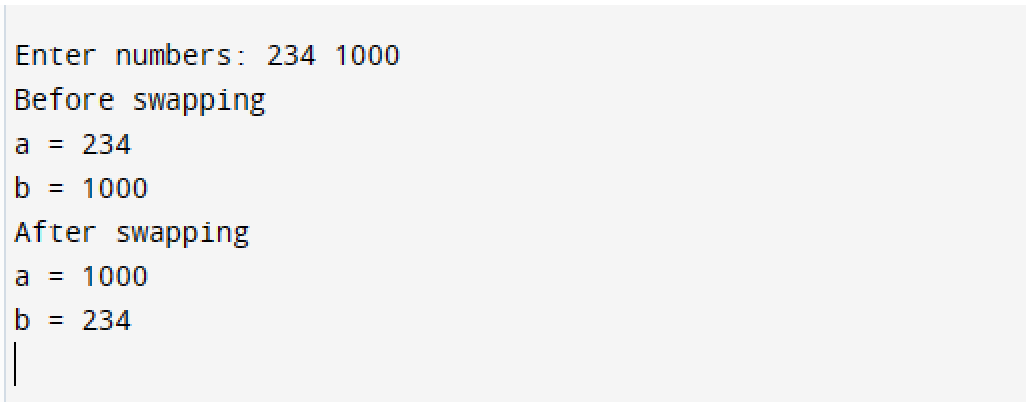
Program 4

Aim: C++ program to Swap two values using call by reference.

Code:

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

void swap(int *a, int *b){
    int temp=*a;
    *a=*b;
    *b=temp;
}

int main(){
    int a,b;
    cout<<"Enter numbers: ";
    cin>>a>>b;
    cout<<"Before swapping"<<endl;
    cout<<"a = "<<a<<endl;
    cout<<"b = "<<b<<endl;
    cout<<"After swapping"<<endl;
    swap(&a,&b);
    cout<<"a = "<<a<<endl;
    cout<<"b = "<<b<<endl;
    return 0;
}
```

Output:



```
Enter numbers: 234 1000
Before swapping
a = 234
b = 1000
After swapping
a = 1000
b = 234
```

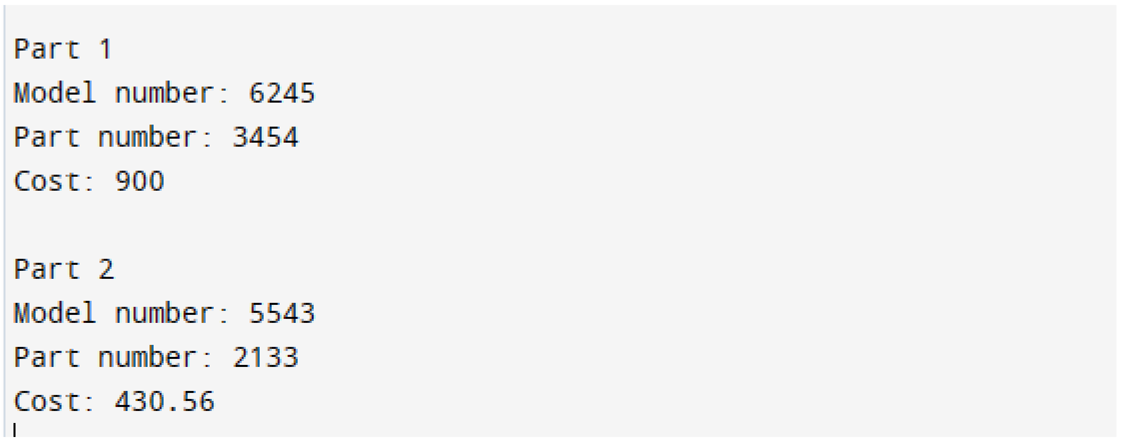
Program 5

Aim: Program to create a class 'Part' and create two objects of it.

Code:

```cpp
#include <iostream>
using namespace std;
class Part{
    private:
      int Modelnumber;
      int Partnumber;
      float Cost;
    public:
      void setpart(int m,int p,float c){
          Modelnumber=m;
          Partnumber=p;
          Cost=c;
      }
      void printpart(){
          cout<<"Model number: "<<Modelnumber<<endl;
          cout<<"Part number: "<<Partnumber<<endl;
          cout<<"Cost: "<<Cost<<endl;
      }
};

int main(){
    Part p1,p2;

    cout<<"Part 1"<<endl;
    p1.setpart(6245,3454,900);
    p1.printpart();

    cout<<endl<<"Part 2"<<endl;
    p2.setpart(5543,2133,430.56);
    p2.printpart();
    return 0;
}
```

Output:



```
Part 1
Model number: 6245
Part number: 3454
Cost: 900

Part 2
Model number: 5543
Part number: 2133
Cost: 430.56
```

Program 6

Aim: Program to create a class 'Part' and demonstrate the concept of array of objects.

Code:

```
#include <iostream>
using namespace std;
class Part{
    private:
      int Modelnumber;
      int Partnumber;
      float Cost;
    public:
      void setpart(int,int,float);
      void printpart();
};

void Part::setpart(int m,int p,float c){
    Modelnumber=m;
    Partnumber=p;
    Cost=c;
}
void Part::printpart(){
    cout<<"Model number: "<<Modelnumber<<endl;
    cout<<"Part number: "<<Partnumber<<endl;
    cout<<"Cost: "<<Cost<<endl;
}
int main(){
    Part parts[3];

    for (int i=0;i<3;i++){
        int m,p;
        float c;
        cout<<"For Part "<<i+1<<endl;
        cout<<"Enter Model no: ";
        cin>>m;
        cout<<"Enter Part no:";
        cin>>p;
        cout<<"Enter cost: ";
        cin>>c;
        parts[i].setpart(m,p,c);
        cout<<endl;
    }

    cout<<endl;
    for (int i=0; i<3; i++){
        parts[i].printpart();
        cout<<endl;
    }
    return 0;
}
```

Output:

```
For Part 1
Enter Model no: 2003
Enter Part no:232
Enter cost: 400
For Part 2
Enter Model no: 1001
Enter Part no:56
Enter cost: 500
For Part 3
Enter Model no: 6768
Enter Part no:454
Enter cost: 800
```

```
Model number: 2003
Part number: 232
Cost: 400

Model number: 1001
Part number: 56
Cost: 500

Model number: 6768
Part number: 454
Cost: 800
```

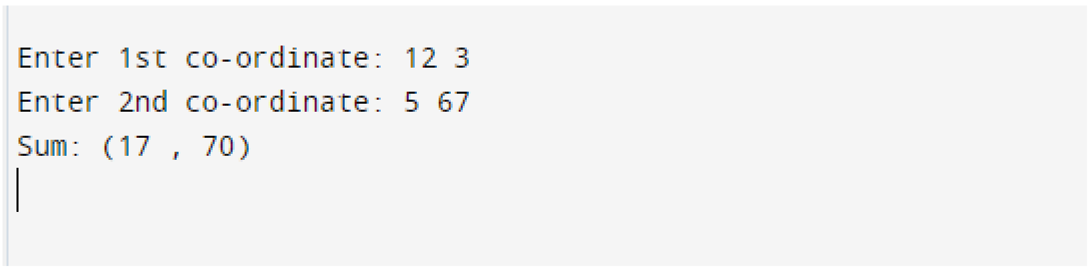
Program 7

Aim: A point on the two-dimensional plane can be represented by two numbers: an X coordinate and a Y coordinate. For example, (4,5) represents a point 4 units to the right of the origin along the X axis and 5 units up the Y axis. The sum of two points can be defined as a new point that X coordinate is the sum of the X coordinates of the points and whose Y coordinate is the sum of their Y coordinates. Write a program that uses a structure called point to model a point.

Code:

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

struct point{
    int x,y;
};

int main(){
    struct point p1,p2,p3;
    cout<<"Enter 1st co-ordinate: ";
    cin>>p1.x>>p1.y;
    cout<<"Enter 2nd co-ordinate: ";
    cin>>p2.x>>p2.y;

    p3.x=p2.x+p1.x;
    p3.y=p2.y+p1.y;

    cout<<"Sum: ("<<p3.x<<" , "<<p3.y<<")"<<endl;
    return 0;
}
```

Output:



```
Enter 1st co-ordinate: 12 3
Enter 2nd co-ordinate: 5 67
Sum: (17 , 70)
```

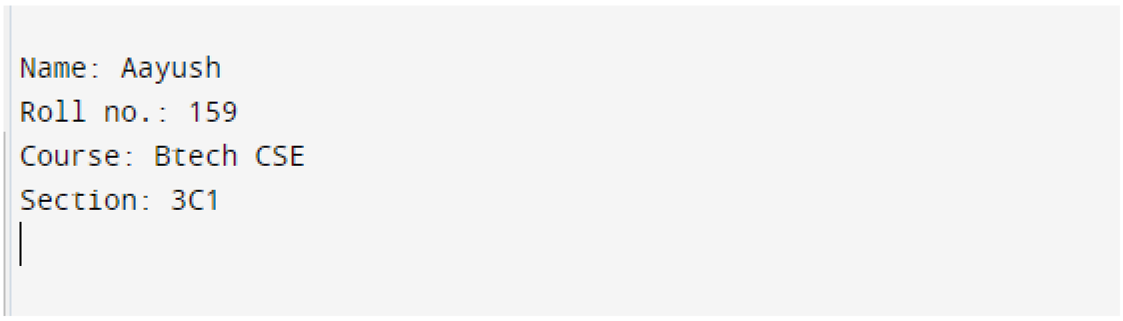
Program 8

Aim: Create student class and display student data of a single student.

Code:

```cpp
#include <iostream>
using namespace std;
class Student{
    private:
       string Name;
       int roll;
       string course;
       string section;
    public:
       Student(string name,int roll,string course, string section){
           this->Name=name;
           this->roll=roll;
           this->course=course;
           this->section=section;
       }
       void display(){
           cout<<"Name: "<<Name<<endl;
           cout<<"Roll no.: "<<roll<<endl;
           cout<<"Course: "<<course<<endl;
           cout<<"Section: "<<section<<endl;
       }
};

int main(){
    Student st("Aayush",159,"Btech CSE","3C1");
    st.display();
    return 0;
}
```

Output:



```
Name: Aayush
Roll no.: 159
Course: Btech CSE
Section: 3C1
```

Program 9

Aim: Write a C++ program to take input and display data of 5 employees using array of objects and constructors.

Code:

```
#include <iostream>
using namespace std;
class Employee{
  private:
     string Name;
     float salary;
     string DOB;
  public:
     static int i;
     Employee(){
         cout<<"For "<<i<<" employee"<<endl;
         cout<<"Enter Name: ";
         cin>>Name;
         cout<<"Enter DOB: ";
         cin>>DOB;
         cout<<"Enter salary: ";
         cin>>salary;
         i++;
     }
     void display(){
         cout<<"Name: "<<Name<<endl;
         cout<<"DOB: "<<DOB<<endl;
         cout<<"Salary: "<<salary<<endl;
     }
};

int Employee::i=1;
int main(){
    Employee arr[5];
    cout<<endl;
    for (int i=0; i<5; i++){
        cout<<"Employee "<<i+1<<endl;;
        arr[i].display();
        cout<<endl;
    }
    return 0;
}
```

Output:

```
For 1 employee
Enter Name: Vinayak
Enter DOB: 08-12-1976
Enter salary: 600000
For 2 employee
Enter Name: Arpit
Enter DOB: 17-08-1999
Enter salary: 455000.50
For 3 employee
Enter Name: Kirti
Enter DOB: 27-01-1987
Enter salary: 3400000
For 4 employee
Enter Name: Anmol
Enter DOB: 31-07-1990
Enter salary: 500000
For 5 employee
Enter Name: Himani
Enter DOB: 22-09-1986
Enter salary: 4560000
```

```
Employee 1
Name: Vinayak
DOB: 08-12-1976
Salary: 600000

Employee 2
Name: Arpit
DOB: 17-08-1999
Salary: 455000

Employee 3
Name: Kirti
DOB: 27-01-1987
Salary: 3.4e+06

Employee 4
Name: Anmol
DOB: 31-07-1990
Salary: 500000

Employee 5
Name: Himani
DOB: 22-09-1986
Salary: 4.56e+06
```

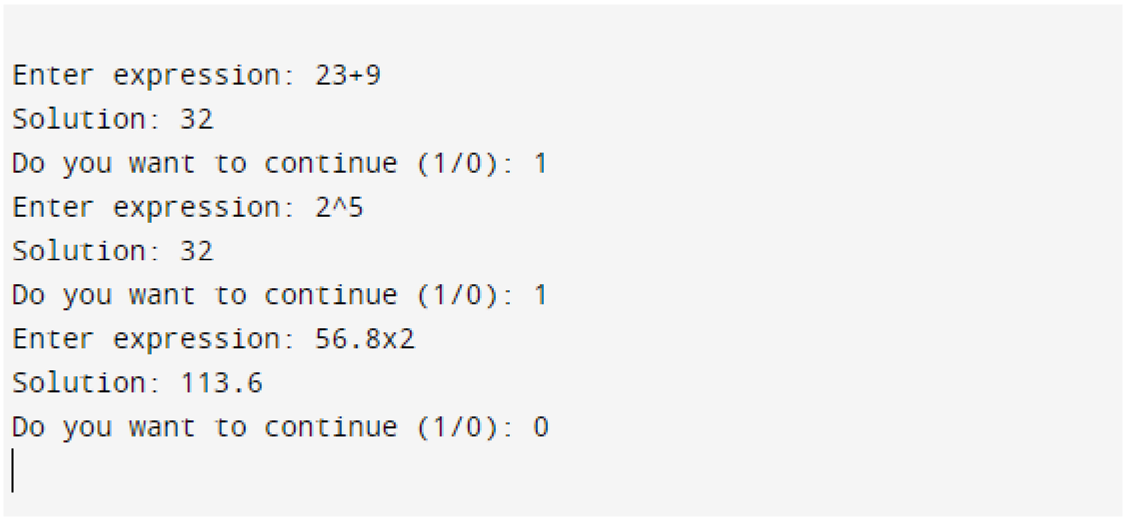
Program 10

Aim: Create the equivalent of a four-function calculator. The program should request the user to enter a number, an operator and another number. It should then carry out the specified arithmetical operation: adding. subtracting, multiplying or dividing the two numbers (It should use a switch statement to select the operation). Finally, it should display the result. When it finishes the calculation, the program should ask if the user wants to do another calculation.

Code:

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

int main(){
 float a,b;
 char c;
 while(true){
     cout<<"Enter expression: ";
     cin>>a>>c>>b;
     cout<<"Solution: ";
     switch(c){
      case '+':
         cout<<a+b;
         break;
      case '-':
         cout<<a-b;
         break;
      case '*':
      case 'x':
         cout<<a*b;
         break;
      case '/':
         cout<<a/b;
         break;
      case '^':
         cout<<pow(a,b);
         break;
      default:
         cout<<"No such operator found";
     }
     cout<<endl;
     bool ask;
     cout<<"Do you want to continue (1/0): ";
     cin>>ask;
     if (ask==0) break;
 }
 return 0;
}
```

Output:



```
Enter expression: 23+9
Solution: 32
Do you want to continue (1/0): 1
Enter expression: 2^5
Solution: 32
Do you want to continue (1/0): 1
Enter expression: 56.8x2
Solution: 113.6
Do you want to continue (1/0): 0
```

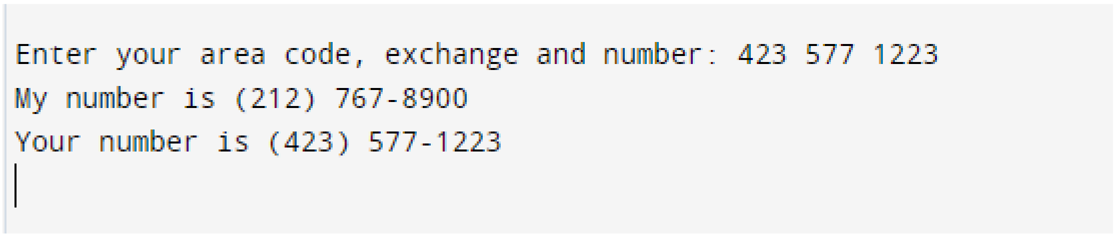
Program 11

Aim: Phone number, such as (212) 767-8900, can be thought of as having three parts: the area code (212), the exchange (767) and the number (8900). Write a program that uses a structure to store these three parts of a phone number separately. Call the structure phone. Create two structure variables of type phone. Initialize one and have the user input a number for the other one. Then display both numbers.

Code:

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

struct phone{
  int area_code;
  int exchange;
  int number;
};
int main(){
    struct phone p1,p2;
    p1.area_code=212;
    p1.exchange=767;
    p1.number=8900;

    cout<<"Enter your area code, exchange and number: ";
    cin>>p2.area_code>>p2.exchange>>p2.number;

    cout<<"My number is ";
    cout<<"("<<p1.area_code<<") "<<p1.exchange<<"-"<<p1.number<<endl;
    cout<<"Your number is ";
    cout<<"("<<p2.area_code<<") "<<p2.exchange<<"-"<<p2.number<<endl;
    return 0;
}
```

Output:



```
Enter your area code, exchange and number: 423 577 1223
My number is (212) 767-8900
Your number is (423) 577-1223
```

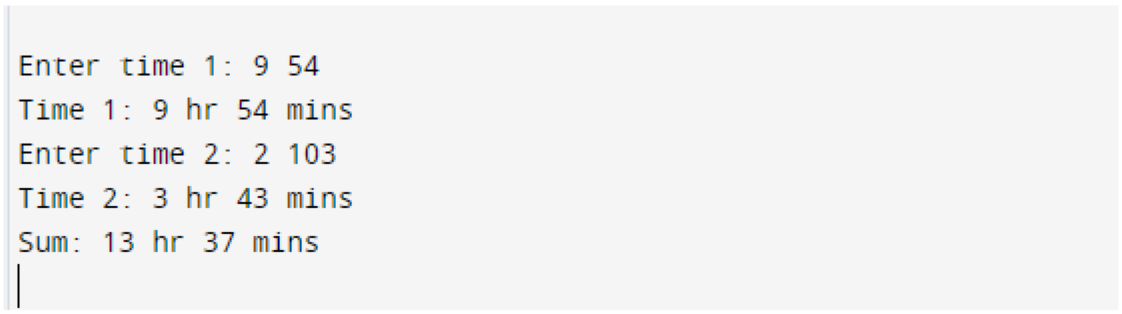
Program 12

Aim: Create a class Time that store hours and minutes. Inside it create a function that returns sum of two time.

Code:

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

class Time{
    private:
      int hour;
      int min;
    public:
     void set_time(int h,int m){
         min=m%60;
         hour=h%24+m/60;
     }
     Time add_time(Time b){
         Time c;
         c.min=(min+b.min)%60;
         c.hour=(hour+b.hour)%24+(min+b.min)/60;
         return c;
     }
     void print_time(){
         cout<<hour<<" hr "<<min<<" mins"<<endl;
     }
};

int main(){
    Time a,b;
    int hour,min;
    cout<<"Enter time 1: ";
    cin>>hour>>min;
    a.set_time(hour,min);
    cout<<"Time 1: ";
    a.print_time();

    cout<<"Enter time 2: ";
    cin>>hour>>min;
    b.set_time(hour,min);
    cout<<"Time 2: ";
    b.print_time();

    Time t=a.add_time(b);
    cout<<"Sum: ";
    t.print_time();
    return 0;
}
```

Output:



```
Enter time 1: 9 54
Time 1: 9 hr 54 mins
Enter time 2: 2 103
Time 2: 3 hr 43 mins
Sum: 13 hr 37 mins
```

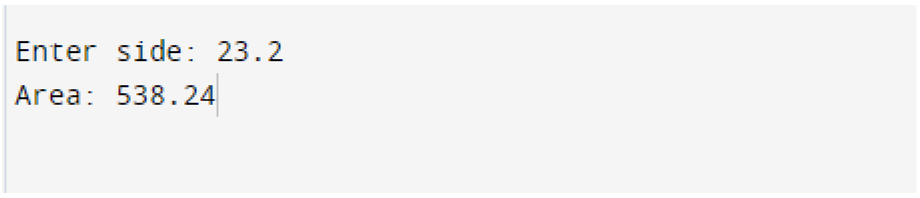
Program 13

Aim: Create a class ‘Square’ and find area of a square using friend function.

Code:

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

class Square{
    private:
      float side;
    public:
      Square(float side){
          this->side=side;
      }
      friend float area(Square);
};

float area(Square sq){
    return sq.side*sq.side;
}

int main(){
    float a;
    cout<<"Enter side: ";
    cin>>a;
    Square sq(a);
    cout<<"Area: "<<area(sq);
    return 0;
}
```

Output:



```
Enter side: 23.2
Area: 538.24
```

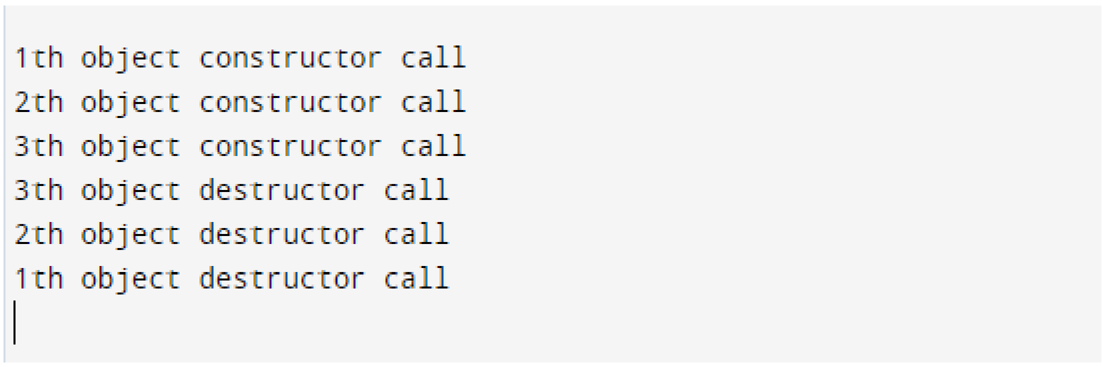
Program 14

Aim: Create a class and implement concept of destructors.

Code:

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

class Books{
  private:
    static int i;
    int data;
  public:
    Books(){
        i++;
        cout<<i<<"th object constructor call\n";
    }
    ~Books(){
        cout<<i<<"th object destructor call\n";
        i--;
    }
};
int Books::i=0;

int main(){
    Books b1,b2,b3;
    return 0;
}
```

Output:



```
1th object constructor call
2th object constructor call
3th object constructor call
3th object destructor call
2th object destructor call
1th object destructor call
```

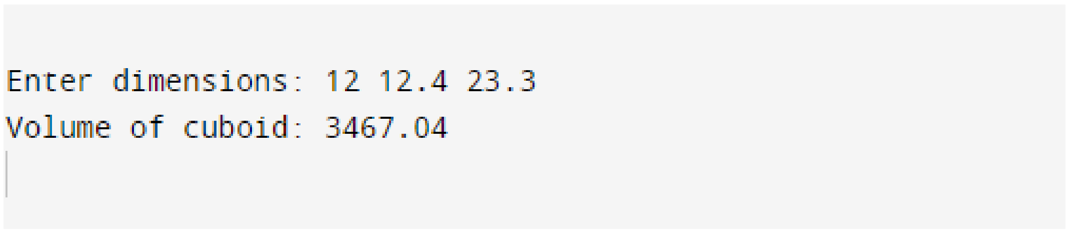
Program 15

Aim: C++ program to implement inline function.

Code:

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

inline void vol(float &a,float &b,float &c){
    cout<<"Volume of cuboid: "<<a*b*c<<endl;
}

int main(){
    float a,b,c;
    cout<<"Enter dimensions: ";
    cin>>a>>b>>c;
    vol(a,b,c);
    return 0;
}
```

Output:



```
Enter dimensions: 12 12.4 23.3
Volume of cuboid: 3467.04
```

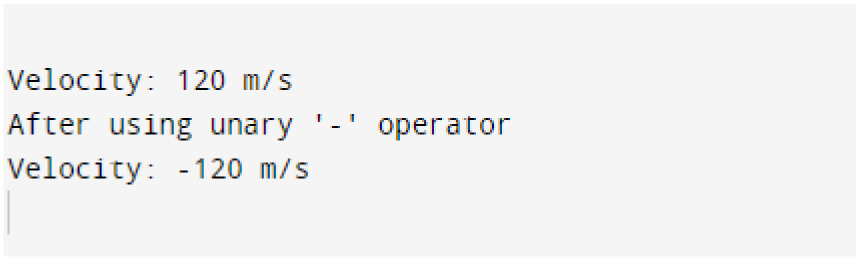
Program 16

Aim: C++ program to overload a unary '-' operator.

Code:

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

class Velocity{
    private:
       float v;
    public:
       Velocity(){
           v=0;
       }
       Velocity(float v){
           this->v=v;
       }
       void display(){
           cout<<"Velocity: "<<v<<" m/s"<<endl;
       }
       Velocity operator-();
};

Velocity Velocity::operator-(){
    Velocity temp;
    temp.v=-v;
    return temp;
}

int main(){
    Velocity v1(120);
    v1.display();
    Velocity v2=-v1;
    v2.display();
    return 0;
}
```

Output:



```
Velocity: 120 m/s
After using unary '-' operator
Velocity: -120 m/s
```

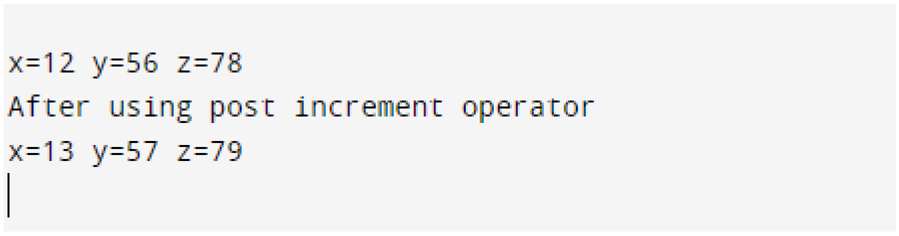
Program 17

Aim: C++ program to overload unary operator using friend function.

Code:

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

class Point{
  private:
     int x,y,z;
  public:
     Point(){
         x=y=z=0;
     }
     Point(int x, int y, int z){
         this->x=x;
         this->y=y;
         this->z=z;
     }
     void display(){
         cout<<"x="<<x<<" y="<<y<<" z="<<z<<endl;
     }
     friend Point operator++(Point &);
};

Point operator++(Point &p){
    p.x++;
    p.y++;
    p.z++;
    return p;
}

int main(){
    Point p1(12,56,78);
    p1.display();
    cout<<"After using post increment operator\n";
    ++p1;
    p1.display();
    return 0;
}
```

Output:



```
x=12 y=56 z=78
After using post increment operator
x=13 y=57 z=79
```

Program 18

Aim: Create two classes DM and DB which store value of distances. DM stores distances in meters and centimeters and DB in feet and inches. Write a program that can read values for the class objects and add one object of DM with another object of DB. Use a friend function to carry out the addition operator. The object that stores the results maybe a DM object or DB object depending on the units in which the results are required. The display should be in the format of feet and inches or meters and centimeters depending on object on display.

Code:

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

class DB;
class DM{
    float meters,centimeters;
    public:
       DM(float m,float cm){
           meters=m;
           centimeters=cm;
       }
       void display(){
           cout<<"Distance: "<<round(meters)<<" m "<<round(centimeters)<<"
cm\n";
       }
       friend DM add(DM,DB);
       friend DB add(DB,DM);
};

class DB{
    float feet,inches;
    public:
       DB(float f,float i){
           feet=f;
           inches=i;
       }
       void display(){
           cout<<"Distance: "<<round(feet)<<" feet "<<round(inches)<<"
inches\n";
       }
       friend DM add(DM,DB);
       friend DB add(DB,DM);
};
DM add(DM d1, DB d2){
    DM temp(0,0);
    temp.meters=d1.meters+(3.2808*d2.feet);
    temp.centimeters=d1.centimeters+(d2.inches/2.54);
    return temp;
}
```

```
DB add(DB d2, DM d1){
    DB temp(0,0);
    temp.feet=d2.feet+d1.meters*0.3048;
    temp.inches=d2.inches+d1.centimeters*2.54;
    return temp;
}

int main(){
    DM a(2,3);
    DB b(10,5);
    DM c=add(a,b);
    c.display();
    DB d=add(b,a);
    d.display();
    return 0;
}
```

Output:

```
Sum:
Distance: 35 m 5 cm
Distance: 11 feet 13 inches
```

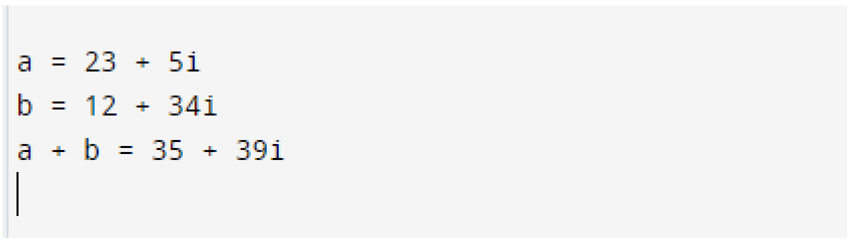
Program 19

Aim: C++ program to overload binary '+' operator to add two complex numbers.

Code:

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

class Complex{
    private:
        float real;
        float img;
    public:
        Complex(float r=0, float i=0){
            real=r;
            img=i;
        }
        void display(){
            cout<<real<<" + "<<img<<"i"<<endl;
        }
        Complex operator+(Complex&);
};

Complex Complex::operator+(Complex &c){
    Complex temp;
    temp.real=real+c.real;
    temp.img=img+c.img;
    return temp;
}

int main(){
    Complex a(23,5);
    Complex b(12,34);
    Complex c=a+b;
    cout<<"a = ";
    a.display();
    cout<<"b = ";
    b.display();
    cout<<"a + b = ";
    c.display();
    return 0;
}
```

Output:



```
a = 23 + 5i
b = 12 + 34i
a + b = 35 + 39i
```

Program 20

Aim: C++ program to overload binary '+' operator to add two complex numbers using friend function.

Code:

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

class Complex{
    private:
        float real;
        float img;
    public:
        Complex(float r=0, float i=0){
            real=r;
            img=i;
        }
        void display(){
            cout<<real<<" + "<<img<<"i"<<endl;
        }
        friend Complex operator+(Complex&,Complex&);
};

Complex operator+(Complex &c,Complex &d){
    Complex temp;
    temp.real=d.real+c.real;
    temp.img=d.img+c.img;
    return temp;
}

int main(){
    Complex a(9.12,90);
    Complex b(22.21,5.6);
    Complex c=a+b;
    cout<<"a = ";
    a.display();
    cout<<"b = ";
    b.display();
    cout<<"a + b = ";
    c.display();
    return 0;
}
```

Output:

```
a = 9.12 + 90i
b = 22.21 + 5.6i
a + b = 31.33 + 95.6i
```

Program 21

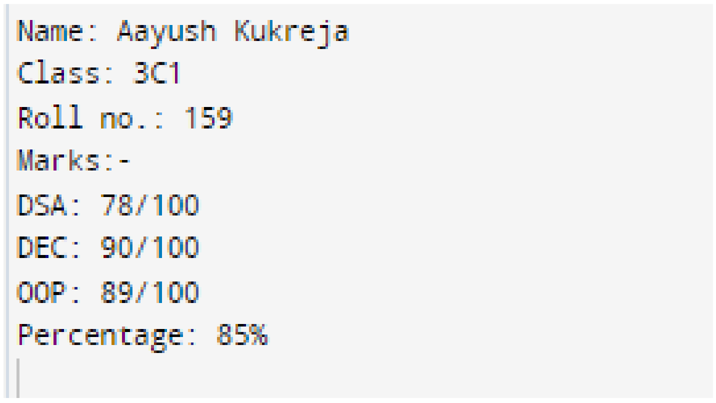
Aim: C++ program to inherit Student class using single inheritance.

Code:

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

class Student{
  protected:
     string Name;
     string Class;
     int roll;
  public:
     void set(string n,string c,int r){
         Name=n;
         Class=c;
         roll=r;
     }
};

class Result : public Student{
  private:
     int DSA;
     int OOP;
     int DEC;
     int total;
     double percentage;
  public:
     void set_marks(int dsa, int oop, int dec){
         DSA=dsa;
         OOP=oop;
         DEC=dec;
         total=dsa+oop+dec;
         percentage=total/3;
     }
     void details(){
         cout<<"Name: "<<Name<<endl;
         cout<<"Class: "<<Class<<endl;
         cout<<"Roll no.: "<<roll<<endl;
         cout<<"Marks:-\n";
         cout<<"DSA: "<<DSA<<"/100"<<endl;
         cout<<"DEC: "<<DEC<<"/100"<<endl;
         cout<<"OOP: "<<OOP<<"/100"<<endl;
         cout<<"Percentage: "<<percentage<<"%"<<endl;
     }
};
```

```
int main(){
    Result obj;
    obj.set("Aayush Kukreja","3C1",159);
    obj.set_marks(78,89,90);
    obj.details();
    return 0;
}
```

Output:



```
Name: Aayush Kukreja
Class: 3C1
Roll no.: 159
Marks:-
DSA: 78/100
DEC: 90/100
OOP: 89/100
Percentage: 85%
```

Program 22

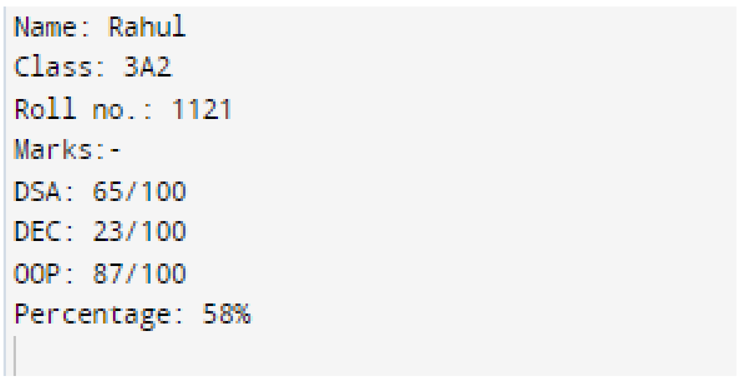
Aim: Write a C++ program to demonstrate multilevel inheritance.

Code:

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

class Student{
  protected:
     string Name;
     string Class;
     int roll;
  public:
     void set(string n,string c,int r){
         Name=n;
         Class=c;
         roll=r;
     }
};

class Test : public Student{
  protected:
     int DSA;
     int OOP;
     int DEC;
  public:
     void set_marks(int dsa, int oop, int dec){
         DSA=dsa;
         OOP=oop;
         DEC=dec;
     }
};

class Result: public Test{
    private:
      int total;
      double percentage;
    public:
      void calculate(){
          total=DSA+DEC+OOP;
          percentage=total/3;
      }
      void details(){
         cout<<"Name: "<<Name<<endl;
         cout<<"Class: "<<Class<<endl;
         cout<<"Roll no.: "<<roll<<endl;
         cout<<"Marks:-\n";
         cout<<"DSA: "<<DSA<<"/100"<<endl;
```

```
            cout<<"DEC: "<<DEC<<"/100"<<endl;
            cout<<"OOP: "<<OOP<<"/100"<<endl;
            cout<<"Percentage: "<<percentage<<"%"<<endl;
        }
};
int main(){
    Result ra;
    ra.set("Rahul","3A2",1121);
    ra.set_marks(65,87,23);
    ra.calculate();
    ra.details();
    return 0;
}
```

Output:



```
Name: Rahul
Class: 3A2
Roll no.: 1121
Marks:-
DSA: 65/100
DEC: 23/100
OOP: 87/100
Percentage: 58%
```

Program 23

Aim: Create two classes named Mammals and Marine Animals. Create another class named Blue Whale which inherits both the above classes. Now, create a function in each of these classes which prints “I am mammal”, “I am a marine animal” and “I belong to both the categories: Mammals as well as Marine Animals” respectively. Now, create an object for each of the above class and try calling

1) function of Mammals by the object of Mammal

2) function of Marine Animal by the object of Marine Animal

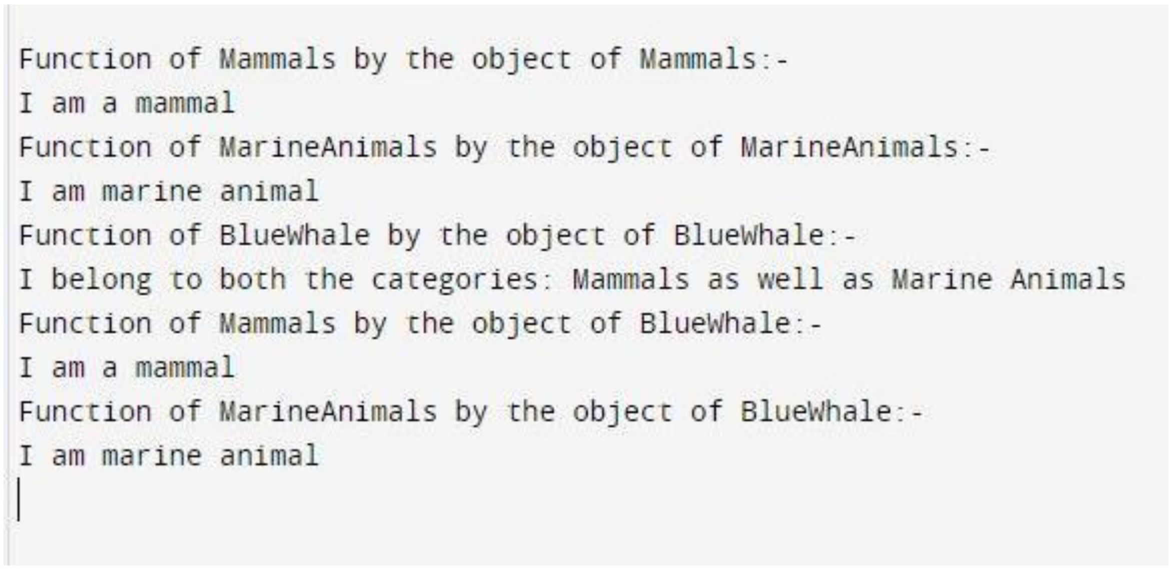
3) function of Blue Whale by the object of Blue Whale

4) function of each of its parent by the object of Blue Whale

Code:

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

class Mammals{
    public:
       void who(){
           cout<<"I am a mammal\n";
       }
};

class MarineAnimals{
    public:
       void who(){
           cout<<"I am marine animal\n";
       }
};

class BlueWhale: public Mammals, public MarineAnimals{
    public:
       void who(){
           cout<<"I belong to both the categories: Mammals as well as Marine
Animals\n";
       }
};

int main(){
    Mammals mam;
    MarineAnimals mar;
    BlueWhale blue;
    cout<<"Function of Mammals by the object of Mammals:-\n";
    mam.who();
    cout<<"Function of MarineAnimals by the object of MarineAnimals:-\n";
    mar.who();
    cout<<"Function of BlueWhale by the object of BlueWhale:-\n";
    blue.who();
    cout<<"Function of Mammals by the object of BlueWhale:-\n";
    blue.Mammals::who();
    cout<<"Function of MarineAnimals by the object of BlueWhale:-\n";
    blue.MarineAnimals::who();
    return 0;
}
```

Output:



```
Function of Mammals by the object of Mammals:-
I am a mammal
Function of MarineAnimals by the object of MarineAnimals:-
I am marine animal
Function of BlueWhale by the object of BlueWhale:-
I belong to both the categories: Mammals as well as Marine Animals
Function of Mammals by the object of BlueWhale:-
I am a mammal
Function of MarineAnimals by the object of BlueWhale:-
I am marine animal
```

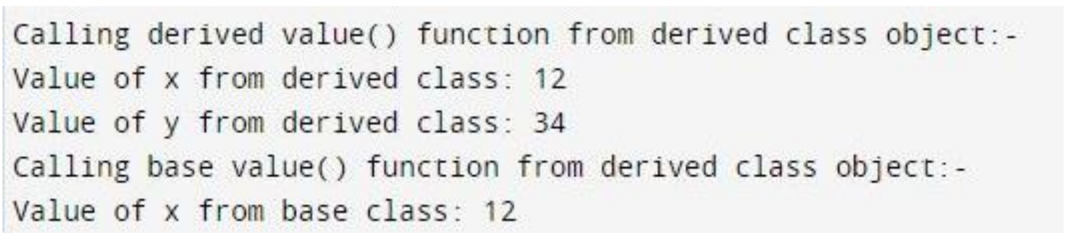
Program 24

Aim: Write a C++ program to demonstrate how arguments can be passed from a derived class constructor to a base class constructor.

Code:

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

class Base{
  protected:
    int x;
  public:
    Base(int x){
        this->x=x;
    }
    void value(){
        cout<<"Value of x from base class: "<<x<<endl;
    }
};

class Derived:public Base{
  int y;
  public:
    Derived(int x,int y):Base(x){
        this->y=y;
    }
    void value(){
        cout<<"Value of x from derived class: "<<x<<endl;
        cout<<"Value of y from derived class: "<<y<<endl;
    }
};

int main(){
    cout<<"Calling base value() function from base class object:-\n";
    Base b(8);
    b.value();
    cout<<"Calling derived value() function from derived class object:-\n";
    Derived d(12,34);
    d.value();
    cout<<"Calling base value() function from derived class object:-\n";
    d.Base::value();
    return 0;
}
```

Output:



```
Calling derived value() function from derived class object:-
Value of x from derived class: 12
Value of y from derived class: 34
Calling base value() function from derived class object:-
Value of x from base class: 12
```

Program 25

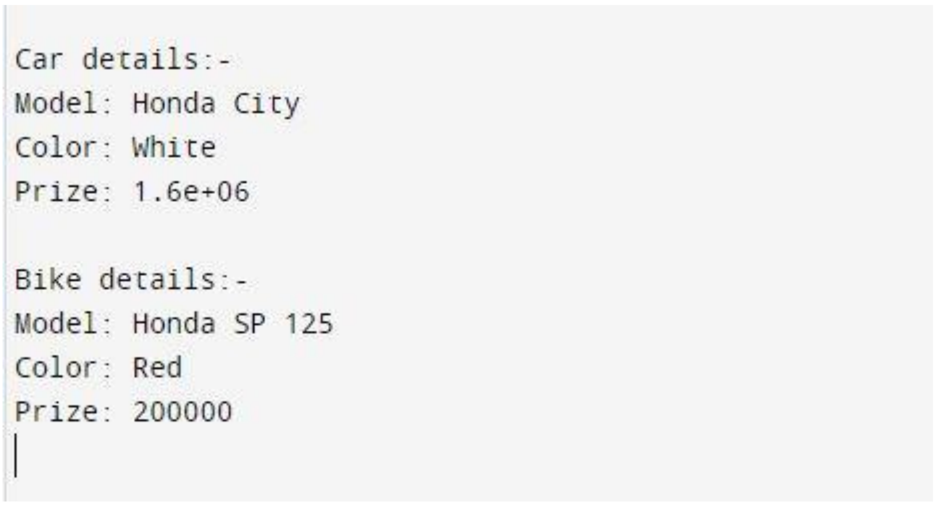
Aim: Write a C++ program to demonstrate the concept of abstract class and pure virtual function.

Code:

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

class Vehicle{
  public:
     virtual void details()=0;
};

class Car:public Vehicle{
  string Model;
  string color;
  float price;
  public:
     Car(string m,string c, float p){
         Model=m;
         color=c;
         price=p;
     }
     void details(){
         cout<<"Car details:-\n";
         cout<<"Model: "<<Model<<endl;
         cout<<"Color: "<<color<<endl;
         cout<<"Prize: "<<price<<endl;
     }
};

class Bike:public Vehicle{
  string Model;
  string color;
  float price;
  public:
    Bike(string m,string c, float p){
         Model=m;
         color=c;
         price=p;
     }
     void details(){
         cout<<"Bike details:-\n";
         cout<<"Model: "<<Model<<endl;
         cout<<"Color: "<<color<<endl;
         cout<<"Prize: "<<price<<endl;
     }
};
```

```
int main(){
    Vehicle* a=new Car("Honda City","White",1600000);
    a->details();
    cout<<endl;
    Vehicle* b=new Bike("Honda SP 125","Red",200000);
    b->details();
    return 0;
}
```

Output:



```
Car details:-
Model: Honda City
Color: White
Prize: 1.6e+06

Bike details:-
Model: Honda SP 125
Color: Red
Prize: 200000
```

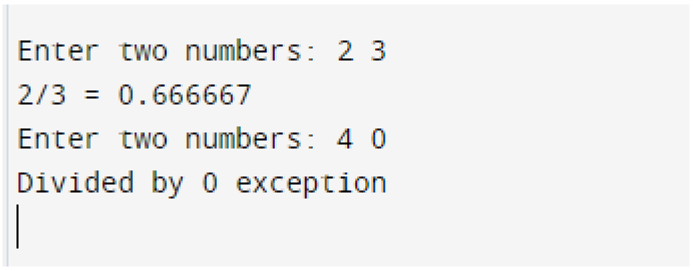
Program 26

Aim: Write a C++ program for throwing an exception.

Code:

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

int main(){
    float a,b;
    printf("Enter two numbers: ");
    cin>>a>>b;

    try{
        if (b==0){
            throw "Divided by 0 exception";
        }
        cout<<a<<"/"<<b<<" = "<<a/b<<endl;
    }
    catch(const char* error){
        cout<<error<<endl;
    }
    catch(...){
        cout<<"Some exceptional error has occurred"<<endl;
    }
}
```

Output:



```
Enter two numbers: 2 3
2/3 = 0.666667
Enter two numbers: 4 0
Divided by 0 exception
```

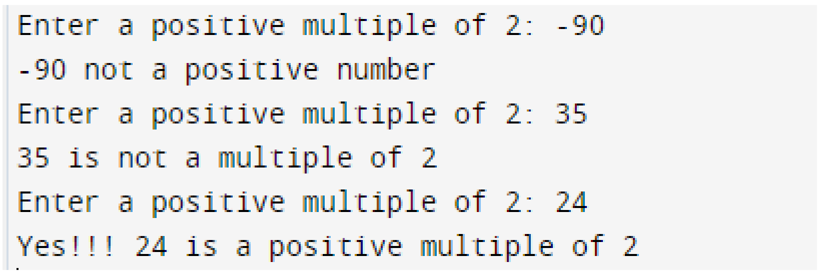
Program 27

Aim: Write a C++ program to throw multiple exception.

Code:

```
#include <iostream>
using namespace std;
class range{
    int a;
    public:
       range(int var){a=var;}
       void what(){
        cout<<a<<" not a positive number"<<endl;
       }
};
class multiple{
    int a;
    public:
       multiple(int v){a=v;}
       void what(){
        cout<<a<<" is not a multiple of 2"<<endl;
       }
};
int main(){
    int a;
    printf("Enter a positive multiple of 2: ");
    cin>>a;
    try{
        if (a<=0){
            throw range(a);
        }
        if (a%2!=0){
            throw multiple(a);
        }
        cout<<"Yes!!! "<<a<<" is a positive multiple of 2"<<endl;
    }
    catch(range a){a.what();}
    catch(multiple c){c.what();}
    catch(...){
        cout<<"Some exception has occurred"<<endl;
     }
    return 0;
}
```

Output:



```
Enter a positive multiple of 2: -90
-90 not a positive number
Enter a positive multiple of 2: 35
35 is not a multiple of 2
Enter a positive multiple of 2: 24
Yes!!! 24 is a positive multiple of 2
```

Program 28

Aim: Write a C++ program for re-throwing an exception.

Code:

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

int main(){
    float a,b;
    cout<<"Enter two numbers (2nd should be +ve): ";
    cin>>a>>b;
    try{
        if (b<0){
            throw b;
        }
        try{
            if (b==0){
                throw "Divide by zero exception";
            }
            cout<<a<<"/"<<b<<" = "<<a/b;
        }
        catch(const char* c){
            throw;
        }
    }
    catch(float b){
        cout<<b<<" is a negative number";
    }
    catch(const char *c){
        cout<<c;
    }
    catch(...){
        cout<<"Some exception has occurred"<<endl;
    }
    return 0;
}
```

Output:

```
Enter two numbers (2nd should be +ve): 56 -3
-3 is a negative number
Enter two numbers (2nd should be +ve): 45 0
Divide by zero exception
Enter two numbers (2nd should be +ve): 23 4
23/4 = 5.75
```

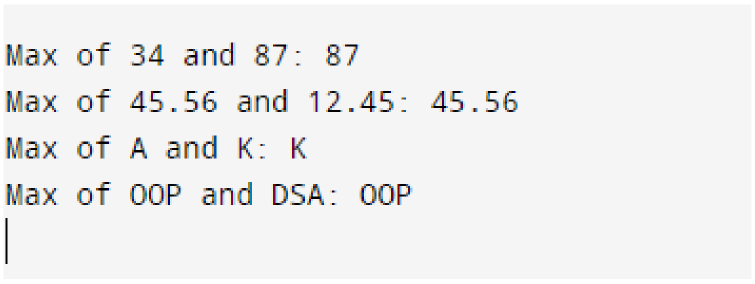
Program 29

Aim: Write a C++ program to create a function template.

Code:

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

template <typename type>
type fmax(type a,type b){
    cout<<"Max of "<<a<<" and "<<b<<": ";
    if (a>=b) return a;
    if (a<b) return b;
}

int main(){
    cout<<fmax(34,87)<<endl;
    cout<<fmax(45.56,12.45)<<endl;
    cout<<fmax('A','K')<<endl;
    cout<<fmax("OOP","DSA")<<endl;
    return 0;
}
```

Output:



```
Max of 34 and 87: 87
Max of 45.56 and 12.45: 45.56
Max of A and K: K
Max of OOP and DSA: OOP
```

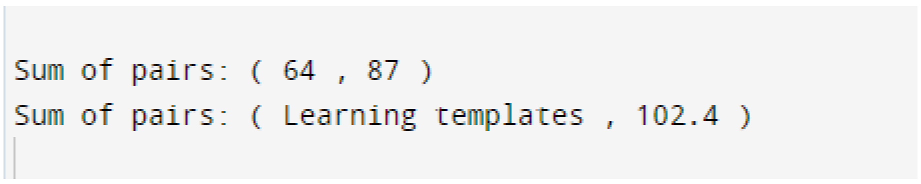
Program 30

Aim: Write a C++ program to create a class template.

Code:

```cpp
#include <iostream>
using namespace std;
template <typename t1,typename t2>
class Pair{
  t1 x;
  t2 y;
  public:
   Pair(){}
   Pair(t1 a,t2 b){
    x=a;
    y=b;
   }
   void show();
   template <typename t3,typename t4>
   friend Pair<t3,t4> operator+(Pair<t3,t4>,Pair<t3,t4>);
};
template <typename t1,typename t2>
void Pair<t1,t2>::show(){
  cout<<"( "<<x<<" , "<<y<<" )\n";
}
template <typename t3,typename t4>
Pair<t3,t4> operator+(Pair<t3,t4> p1,Pair<t3,t4> p2){
  Pair<t3,t4> pair;
  pair.x=p1.x+p2.x;
  pair.y=p1.y+p2.y;
  return pair;
}
int main(){
  Pair <int,int> a(8,9);
  Pair <int,int> b(56,78);
  Pair <int,int> c=a+b;
  cout<<"Sum of pairs: ";
  c.show();
  cout<<"Sum of pairs: ";
  Pair<string,float> st1("Learning ",23.5);
  Pair<string,float> st2("templates",78.9);
  Pair<string,float> st3=st1+st2;
  st3.show();
  return 0;
}
```

Output:



```
Sum of pairs: ( 64 , 87 )
Sum of pairs: ( Learning templates , 102.4 )
```