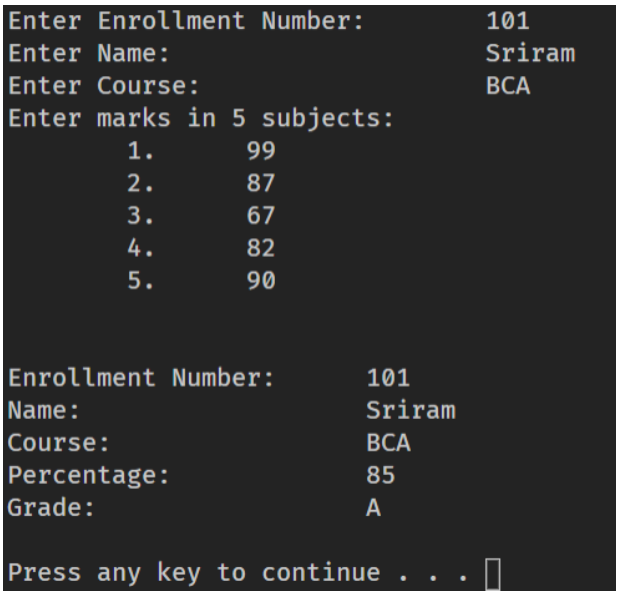
C++ PROGRAMS

Program - 1

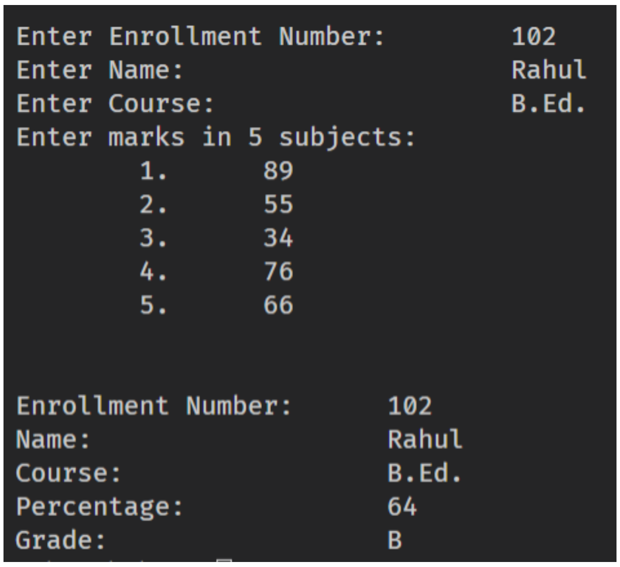
Declare a student class (data members: name, age, array to store marks of 5 subjects; member functions: input(), output()). Print name, age, percentage and grade for 2 student objects.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
#include <windows.h>
using namespace std;

class Student
{
private:
    char name[50], enrolno[15], course[50], grade;
    float marks[5], percentage, total;

    void calculate()
    {
        percentage = total / 5;
        grade = calcGrade(percentage);
    }

    char calcGrade(float percentage)
    {
        if (percentage >= 90)
            return 'O';
        else if (percentage >= 75)
            return 'A';
        else if (percentage >= 60)
            return 'B';
        else if (percentage >= 50)
            return 'C';
        else
            return 'F';
    }

public:
    void input()
    {
        cout << "Enter Enrollment Number:\t";
        cin >> enrolno;

        cout << "Enter Name:\t\t\t";
        cin >> name;

        cout << "Enter Course:\t\t\t";
        cin >> course;

        cout << "Enter marks in 5 subjects:\n";
        for (int i = 1; i <= 5; ++i)
        {
            cout << "\t" << i << ".\t";
            cin >> marks[i - 1];
```

```cpp
            total += marks[i - 1];
        }

        calculate();
    }

    void output()
    {
        cout << "Enrollment Number:\t" << enrolno;
        cout << "\nName:\t\t\t" << name;
        cout << "\nCourse:\t\t\t" << course;
        cout << "\nPercentage:\t\t" << percentage;
        cout << "\nGrade:\t\t\t" << grade;
    }
};

int main()
{
    Student s1, s2;
    s1.input();
    cout << "\n\n";
    s1.output();

    cout << "\n\n";
    system("pause");
    system("cls");

    s2.input();
    cout << "\n\n";
    s2.output();

    return 0;
}
```



```
Enter Enrollment Number:        101
Enter Name:                     Sriram
Enter Course:                   BCA
Enter marks in 5 subjects:
        1.      99
        2.      87
        3.      67
        4.      82
        5.      90


Enrollment Number:      101
Name:                   Sriram
Course:                 BCA
Percentage:             85
Grade:                  A

Press any key to continue . . .
```

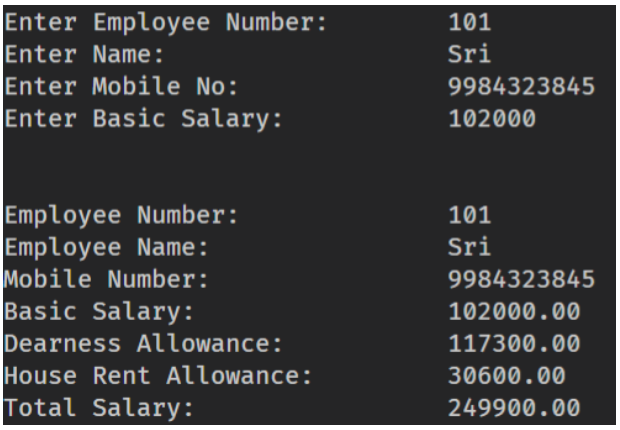


```
Enter Enrollment Number:        102
Enter Name:                     Rahul
Enter Course:                   B.Ed.
Enter marks in 5 subjects:
        1.      89
        2.      55
        3.      34
        4.      76
        5.      66


Enrollment Number:      102
Name:                   Rahul
Course:                 B.Ed.
Percentage:             64
Grade:                  B
```

Program - 2

Declare employee class. Data members are name, empid, mobile number, basic salary, DA,HRA,Gross salary. Member functions are input(), output(). Accept name, empid, mobile number and basic salary. Display all data members and gross salary= basic + HRA+DA. HRA is 30% of basic and DA is 115% of basic.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Employee
{
private:
    int empno;
    double Basic, HRA, DA, Total;
    char empname[30], mobileNo[11];

    void calculate()
    {
        HRA = Basic * 0.3;
        DA = Basic * 1.15;
        Total = Basic + DA + HRA;
    }

public:
    void input()
    {
        printf("%-30s", "Enter Employee Number:");
        cin >> empno;

        printf("%-30s", "Enter Name:");
        cin >> empname;

        printf("%-30s", "Enter Mobile No:");
        cin >> mobileNo;

        printf("%-30s", "Enter Basic Salary:");
        cin >> Basic;

        calculate();
    }

    void output()
    {
        printf("%-30s%d", "Employee Number:", empno);
        printf("\n%-30s%s", "Employee Name:", empname);
        printf("\n%-30s%s", "Mobile Number:", mobileNo);
        printf("\n%-30s%.2lf", "Basic Salary:", Basic);
        printf("\n%-30s%.2lf", "Dearness Allowance:", DA);
        printf("\n%-30s%.2lf", "House Rent Allowance:", HRA);
        printf("\n%-30s%.2lf", "Total Salary:", Total);
    }
};

int main()
{
    Employee e1;
    e1.input();
    cout << "\n\n";
    e1.output();

    return 0;
}
```

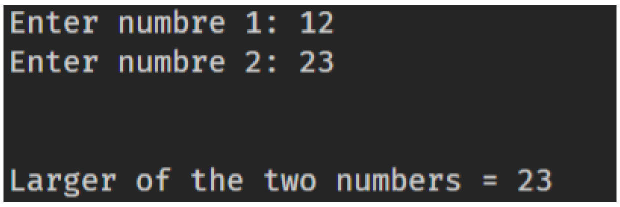


```
Enter Employee Number:         101
Enter Name:                    Sri
Enter Mobile No:               9984323845
Enter Basic Salary:            102000


Employee Number:               101
Employee Name:                 Sri
Mobile Number:                 9984323845
Basic Salary:                  102000.00
Dearness Allowance:            117300.00
House Rent Allowance:          30600.00
Total Salary:                  249900.00
```

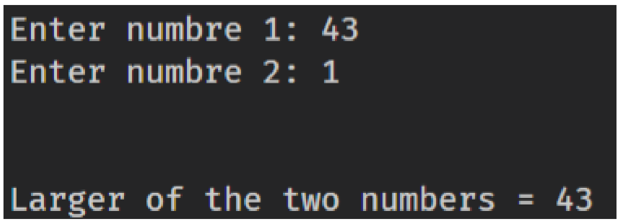
Program - 3

Declare a class number. Data members are I,j (2 integers). Member functions are get(), put()—both public. Add one private function named return_larger() thatreturns larger of I and j and this function should be invoked by put() function (nesting of member functions)Q4. WAP to find out absolute value of an integer, a long int and a floating point number using C language.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Number
{
private:
    int i, j;

    int return_larger()
    {
        return i > j ? i : j;
    }

public:
    void get()
    {
        cout << "Enter numbre 1:\t";
        cin >> i;

        cout << "Enter numbre 2:\t";
        cin >> j;
    }

    void put()
    {
        cout << "Larger of the two numbers = " << return_larger();
    }
};

int main()
{
    Number n1;
    n1.get();
    cout << "\n\n";
    n1.put();

    return 0;
}
```



```
Enter numbre 1: 12
Enter numbre 2: 23


Larger of the two numbers = 23
```

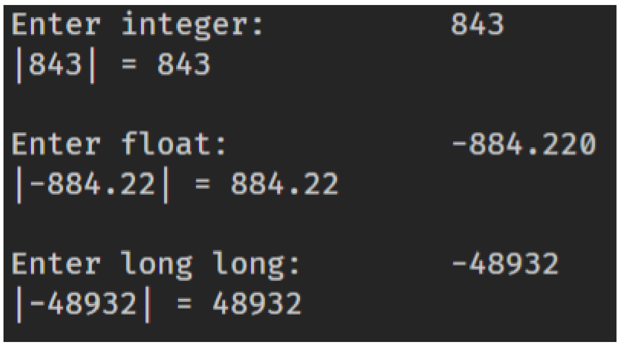


```
Enter numbre 1: 43
Enter numbre 2: 1


Larger of the two numbers = 43
```

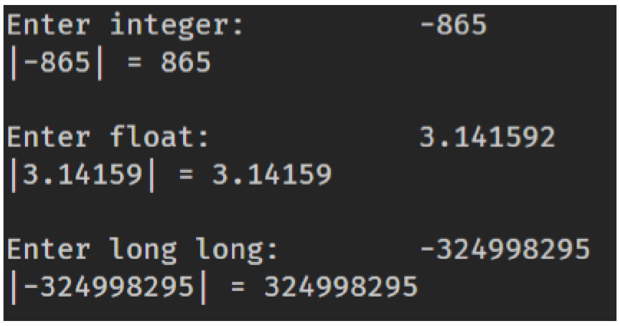
Program - 5

WAP to demonstrate the concept of function overloading to find out the absolute value of an integer, a long and a floating point.

```cpp
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

void absolute(int i)
{
    cout << "|" << i << "| = ";
    cout << ((i < 0) ? -i : i) << "\n\n";
}

void absolute(float f)
{
    cout << "|" << f << "| = ";
    cout << ((f < 0) ? -f : f) << "\n\n";
}

void absolute(long long ll)
{
    cout << "|" << ll << "| = ";
    cout << ((ll < 0) ? -ll : ll) << "\n\n";
}

int main()
{
    int i;
    cout << "Enter integer:\t\t";
    cin >> i;
    absolute(i);

    float f;
    cout << "Enter float:\t\t";
    cin >> f;
    absolute(f);

    long long ll;
    cout << "Enter long long:\t";
    cin >> ll;
    absolute(ll);

    return 0;
}
```



```
Enter integer:          843
|843| = 843

Enter float:            -884.220
|-884.22| = 884.22

Enter long long:        -48932
|-48932| = 48932
```

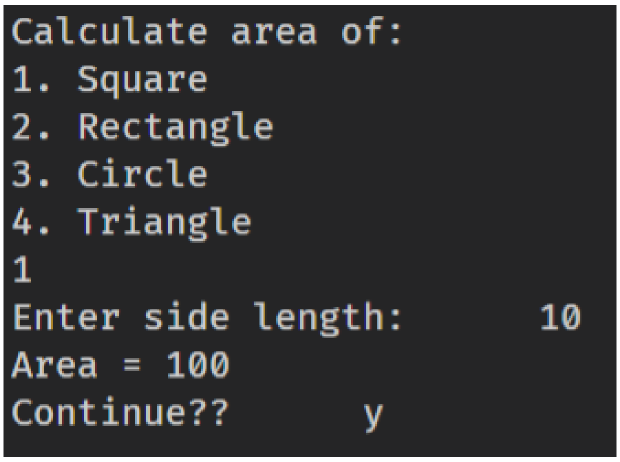


```
Enter integer:          -865
|-865| = 865

Enter float:            3.141592
|3.14159| = 3.14159

Enter long long:        -324998295
|-324998295| = 324998295
```

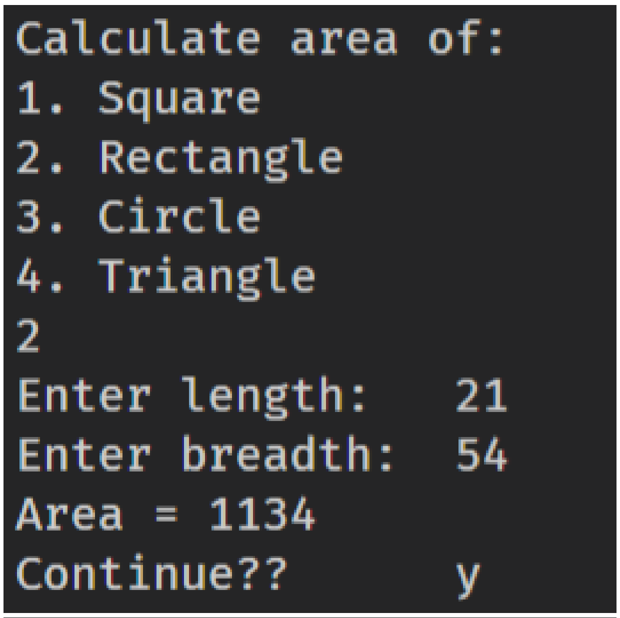
Program - 6

WAP to find outthe area of a square,a rectangle,a circle and a triangle using function overloading concept.Make a menu driven programme, using while loop and within that switch-case.

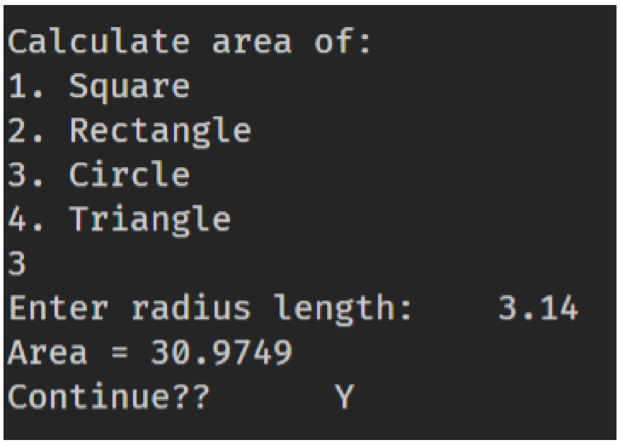
```cpp
/**
* V. Sriram
* 10314902019
*/

/*
* V. Sriram
* 10314902019
*/

#include <iostream>
#define PI 3.14159265358979323846264

using namespace std;

// function overloading used

// square
void area(int side)
{
    cout << "Area = " << side * side << "\n";
}

// circle
void area(float radius)
{
    cout << "Area = " << radius * radius * PI << "\n";
}

// rectangle and triangle
// both rectangle and triangle require 2 parameters for area
// pass half=0.5 to calculate area of triangle
void area(int x, int y, float half = 1.0)
{
    cout << "Area = " << x * y * half << "\n";
}

int main()
{
    // this is a menu driven program

    // the loop breaks when user enters
    // anything other than y or Y
    char toContinue = 'y';
    while (toContinue == 'y' || toContinue == 'Y')
    {
        cout << "Calculate area of:\n";
        cout << "1. Square\n";
        cout << "2. Rectangle\n";
        cout << "3. Circle\n";
        cout << "4. Triangle\n";

        // choice is the index of options printed above
        int choice;
        cin >> choice;
        switch (choice)
        {
        case 1:
            int side;
            cout << "Enter side length:\t";
            cin >> side;
            area(side);
            break;

        case 2:
            int length, breadth;
            cout << "Enter length:\t";
            cin >> length;
            cout << "Enter breadth:\t";
            cin >> breadth;
            area(length, breadth);
            break;

        case 3:
            float radius;
            cout << "Enter radius length:\t";
            cin >> radius;
            area(radius);
            break;

        case 4:
            int height, base;
            cout << "Enter height:\t\t";
            cin >> height;
            cout << "Enter base length:\t";
            cin >> base;
            area(height, base, 0.5);
            break;

        default:
            cout << "Invalid input\n";
        }

        // ask user if they wanna continue with the application
        cout << "Continue??\t";
        cin >> toContinue;

        cout << "\n\n";
    }

    return 0;
}
```



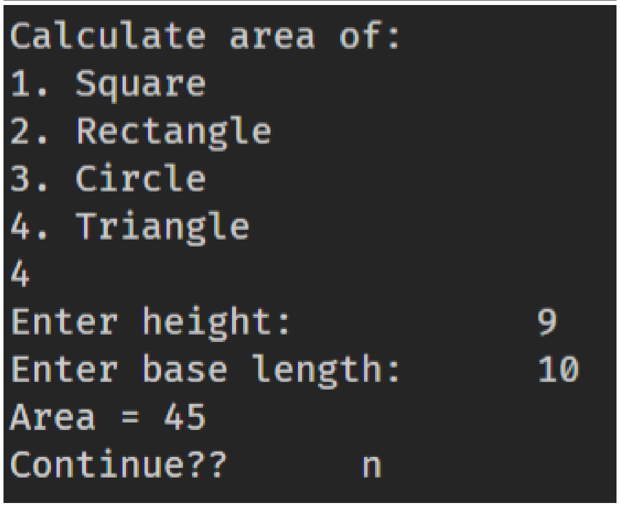
```
Calculate area of:
1. Square
2. Rectangle
3. Circle
4. Triangle
1
Enter side length:      10
Area = 100
Continue??      y
```



```
Calculate area of:
1. Square
2. Rectangle
3. Circle
4. Triangle
2
Enter length:    21
Enter breadth:   54
Area = 1134
Continue??      y
```



```
Calculate area of:
1. Square
2. Rectangle
3. Circle
4. Triangle
3
Enter radius length:    3.14
Area = 30.9749
Continue??      Y
```

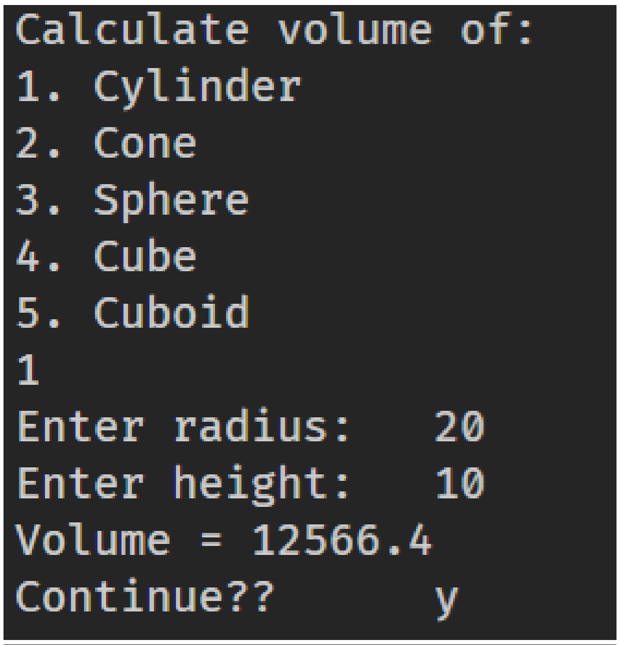


```
Calculate area of:
1. Square
2. Rectangle
3. Circle
4. Triangle
4
Enter height:           9
Enter base length:      10
Area = 45
Continue??      n
```

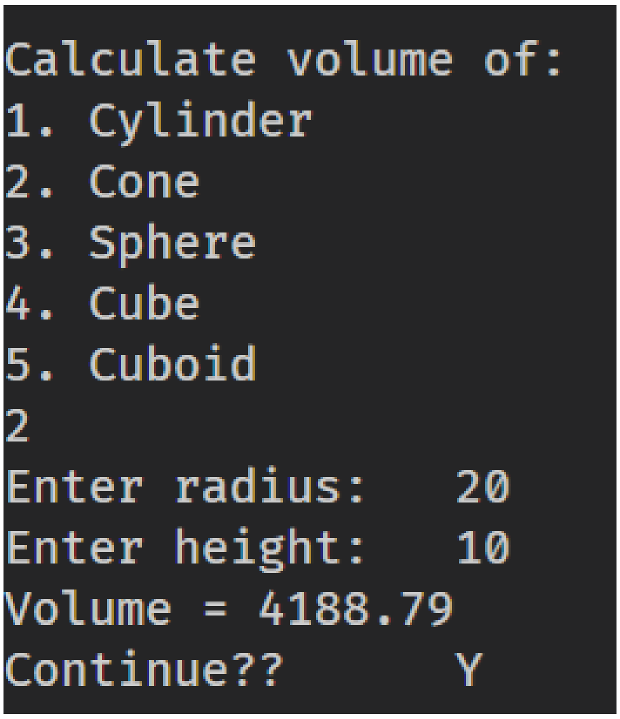
Program - 7

WAP to demonstrate the concept of function overloading to calculate and print the volume of a cone,cube, cuboid, sphere and a cylinder.Make a menu driven programme, using while loop and within that switch-case.

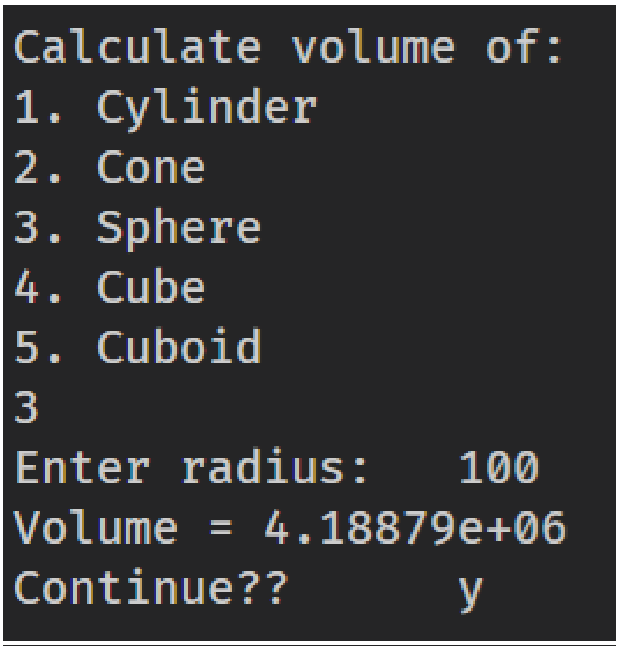
```
/**
* V. Sriram
* 10314902019
*/

/*
* V. Sriram
* 10314902019
*/

/* cone: pi * r * r * h / 3
* cylinder: pi * r * r * h
* sphere: 4 * pi * r * r * r / 3
* cube: a * a * a
* cuboid: l * b * h
*/
#include <iostream>
#include <stdio.h>
#define PI 3.14159265358979323846264338З

using namespace std;

// cone and cylinder
// cone and cylinder both take 2 arguments(radius, height)
// and have similar formula for valume
// pass fracion = 1/3 for cone's volume
void volume(double radius, double height, double fraction = 1.0)
{
    cout << "Volume = " << PI * radius * radius * height * fraction << "\n";
}

// sphere
void volume(double radius)
{
    cout << "Volume = " << 4.0 / 3.0 * radius * radius * radius * PI << "\n";
}

// cube
void volume(long long side)
{
    cout << "Volume = " << side * side * side << "\n";
}

// cuboid
void volume(long long length, long long breadth, long long depth)
{
    cout << "Volume = " << length * breadth * depth << "\n";
}

int main()
{
    double radius, height;
    long long side, length, breadth, depth;
    char toContinue;
    // its a menu driven program
    // program ends when user enter anything
    // other than y or Y when asked
    do
    {
        cout << "Calculate volume of:\n";
        cout << "1. Cylinder\n";
        cout << "2. Cone\n";
        cout << "3. Sphere\n";
        cout << "4. Cube\n";
        cout << "5. Cuboid\n";

        // choice is the index of above printed options
        int choice;
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "Enter radius:\t";
            cin >> radius;
            cout << "Enter height:\t";
            cin >> height;
            volume(radius, height);
            break;
        case 2:
            cout << "Enter radius:\t";
            cin >> radius;
            cout << "Enter height:\t";
            cin >> height;
            volume(radius, height, (1.0 / 3.0));
            break;
        case 3:
            cout << "Enter radius:\t";
            cin >> radius;
            volume(radius);
            break;
        case 4:
            cout << "Enter side length:\t";
            cin >> side;
            volume(side);
            break;
        case 5:
            cout << "Enter length:\t";
            cin >> length;
            cout << "Enter breadth:\t";
            cin >> breadth;
            cout << "Enter depth:\t";
            cin >> depth;
            volume(length, breadth, depth);
            break;
        }

        // ask user if they wanna continue or end the prog
        cout << "Continue??\t";
        cin >> toContinue;
        cout << "\n\n";

    } while (toContinue == 'y' || toContinue == 'Y');

    return 0;
}
```



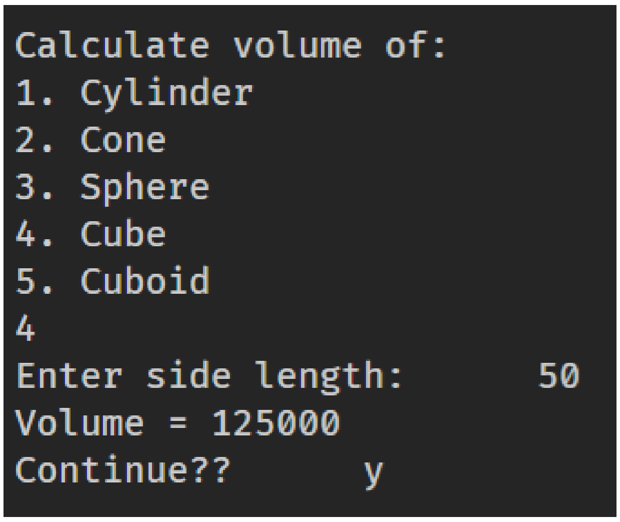
```
Calculate volume of:
1. Cylinder
2. Cone
3. Sphere
4. Cube
5. Cuboid
1
Enter radius:   20
Enter height:   10
Volume = 12566.4
Continue??      y
```



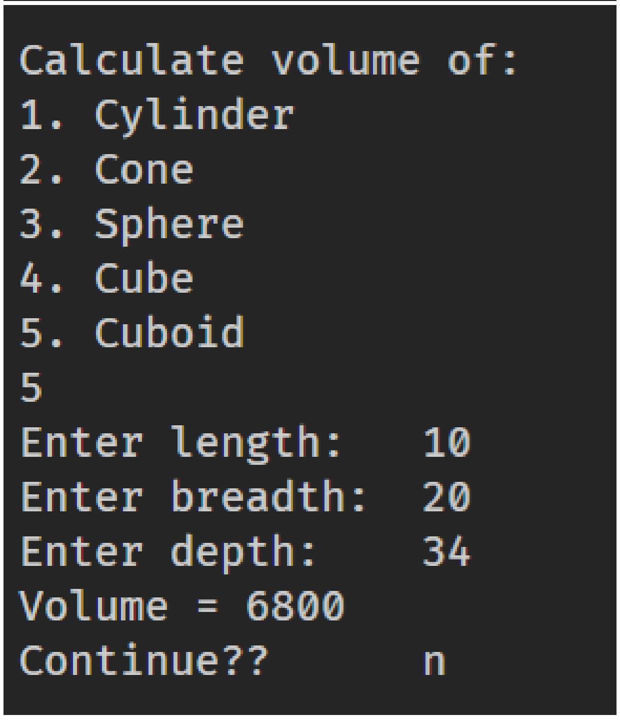
```
Calculate volume of:
1. Cylinder
2. Cone
3. Sphere
4. Cube
5. Cuboid
2
Enter radius:   20
Enter height:   10
Volume = 4188.79
Continue??      Y
```



```
Calculate volume of:
1. Cylinder
2. Cone
3. Sphere
4. Cube
5. Cuboid
3
Enter radius:   100
Volume = 4.18879e+06
Continue??      y
```



```
Calculate volume of:
1. Cylinder
2. Cone
3. Sphere
4. Cube
5. Cuboid
4
Enter side length:      50
Volume = 125000
Continue??      y
```

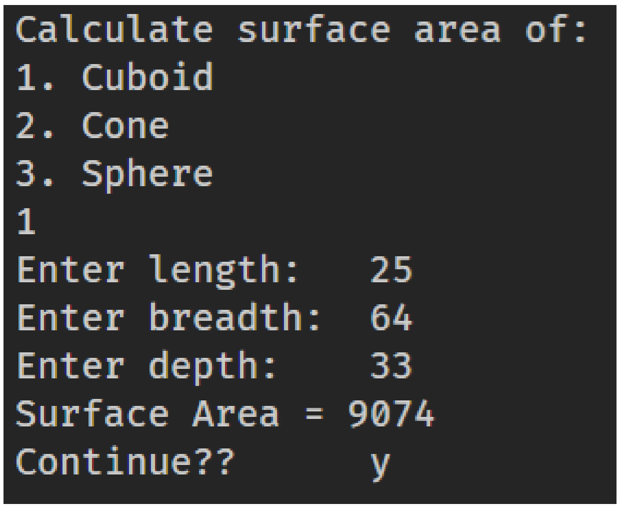


```
Calculate volume of:
1. Cylinder
2. Cone
3. Sphere
4. Cube
5. Cuboid
5
Enter length:   10
Enter breadth:  20
Enter depth:    34
Volume = 6800
Continue??      n
```

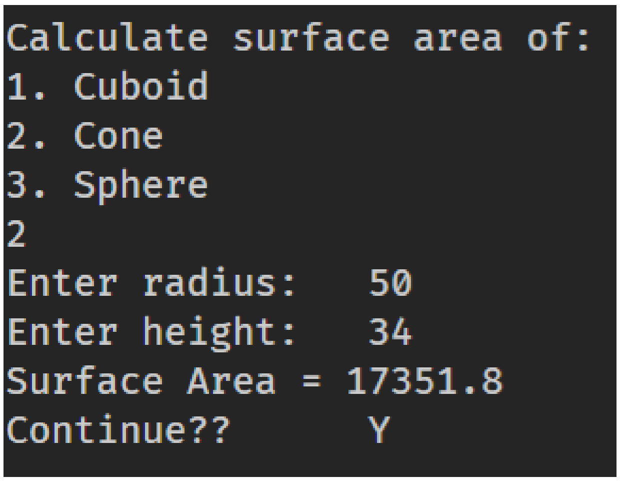
Program - 8

WAP to demonstrate function overloading to calculate surface area of cone,cyclinder and sphere.Make a menu driven programme, using while loop and within that switch-case.

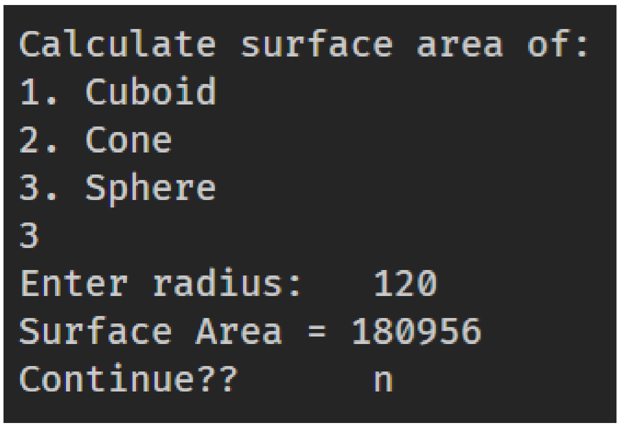
```
/**
* V. Sriram
* 10314902019
*/

/*
* V. Sriram
* 10314902019
*/

// Cuboid: 2(hd + lh + ld)
// Cone: PI*r*(r + sqrt(h^2 + r^2))
// Sphere: 4*pi*r*r

#include <iostream>
#include <cmath> // for sqrt() function
#define PI 3.14159265357989323846264
using namespace std;

// overload surface_area() to calculate surface area
// of different shapes

// cuboid
void surface_area(int l, int b, int d)
{
    int CSA = 2 * (l * b + l * d + b * d);
    cout << "Surface Area = " << CSA << "\n";
}

// cone
void surface_area(double r, double h)
{
    double CSA = PI * r * (r + sqrt(h * h + r * r));
    cout << "Surface Area = " << CSA << "\n";
}

// sphere
void surface_area(double r)
{
    double CSA = 4 * PI * r * r;
    cout << "Surface Area = " << CSA << "\n";
}

int main()
{
    // its a menu driven program
    // program ends when user enter anything
    // other than y or Y when asked
    char toContinue;

    do
    {
        cout << "Calculate surface area of:\n";
        cout << "1. Cuboid\n";
        cout << "2. Cone\n";
        cout << "3. Sphere\n";
        // choice is the index of above printed options
        int choice;
        cin >> choice;

        int length, breadth, depth;
        double radius, height;

        switch (choice)
        {
        case 1:
            cout << "Enter length:\t";
            cin >> length;
            cout << "Enter breadth:\t";
            cin >> breadth;
            cout << "Enter depth:\t";
            cin >> depth;
            surface_area(length, breadth, depth);
            break;

        case 2:
            cout << "Enter radius:\t";
            cin >> radius;
            cout << "Enter height:\t";
            cin >> height;
            surface_area(radius, height);
            break;

        case 3:
            cout << "Enter radius:\t";
            cin >> radius;
            surface_area(radius);
            break;

        default:
            cout << "Invalid input\n";
            break;
        }

        // ask user if they wanna continue or end the prog
        cout << "Continue??\t";
        cin >> toContinue;
        cout << "\n\n";
    } while (toContinue == 'Y' || toContinue == 'y');

    return 0;
}
```



```
Calculate surface area of:
1. Cuboid
2. Cone
3. Sphere
1
Enter length:   25
Enter breadth:  64
Enter depth:    33
Surface Area = 9074
Continue??      y
```



```
Calculate surface area of:
1. Cuboid
2. Cone
3. Sphere
2
Enter radius:   50
Enter height:   34
Surface Area = 17351.8
Continue??      Y
```

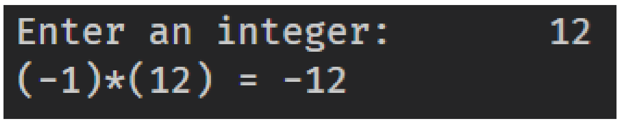


```
Calculate surface area of:
1. Cuboid
2. Cone
3. Sphere
3
Enter radius:   120
Surface Area = 180956
Continue??      n
```

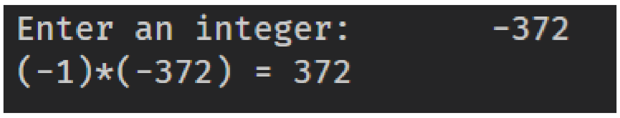
Program - 9

A program to negate the value of an integer by using call by reference.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>

// use pass by reference to direcly invert the input's sign
void negate(int &i)
{
    i = -i;
}

int main()
{
    int i;
    std::cout << "Enter an integer:\t";
    std::cin >> i;

    std::cout << "(-1)*(" << i << ") = ";
    negate(i);
    std::cout << i << "\n";

    return 0;
}
```



```
Enter an integer:       12
(-1)*(12) = -12
```

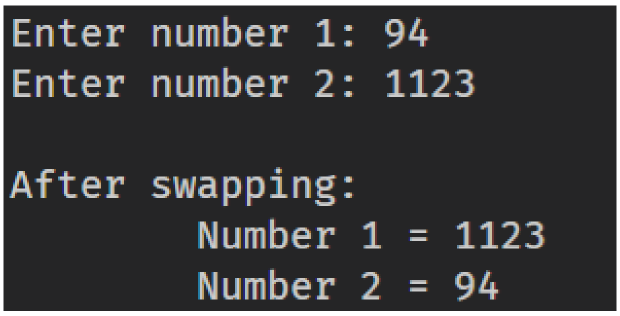


```
Enter an integer:       -372
(-1)*(-372) = 372
```

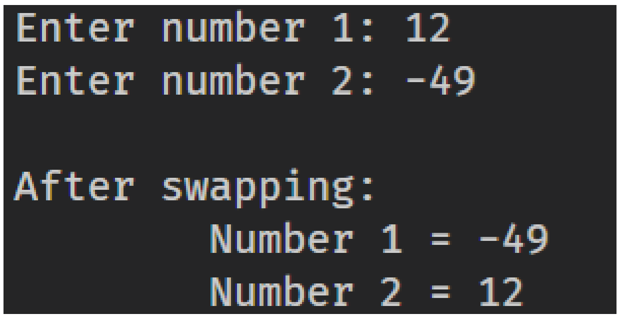
Program - 10

WAP to swap two integers variables using call by reference with reference variable.

```cpp
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

// use pass by reference to swap two integers
void swapByReference(int &a, int &b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}

int main()
{
    int a, b;
    cout << "Enter number 1:\t";
    cin >> a;
    cout << "Enter number 2:\t";
    cin >> b;

    swapByReference(a, b);
    cout << "\nAfter swapping:";
    cout << "\n\tNumber 1 = " << a;
    cout << "\n\tNumber 2 = " << b;

    return 0;
}
```



```
Enter number 1: 94
Enter number 2: 1123

After swapping:
        Number 1 = 1123
        Number 2 = 94
```

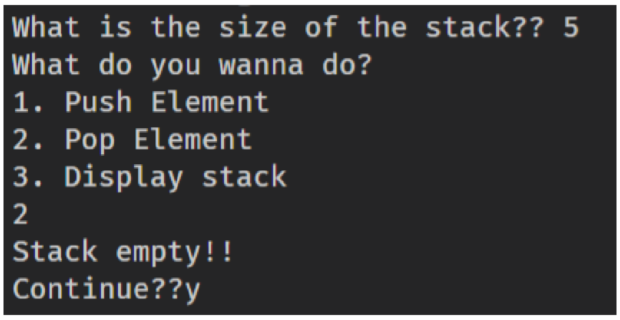


```
Enter number 1: 12
Enter number 2: -49

After swapping:
        Number 1 = -49
        Number 2 = 12
```

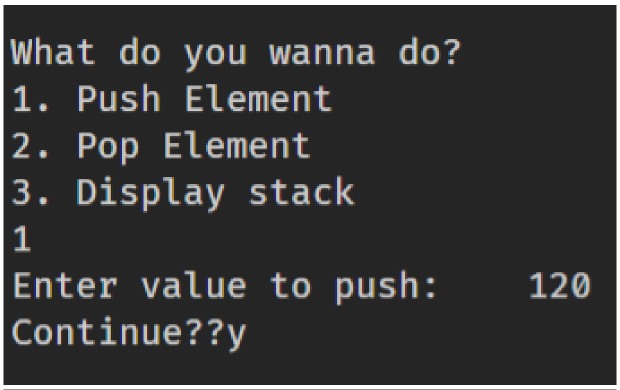
Program - 11

Implementastack. (data members:an array of int, int top_of_stack; member functions push() and pop(), check underflow and overflow).

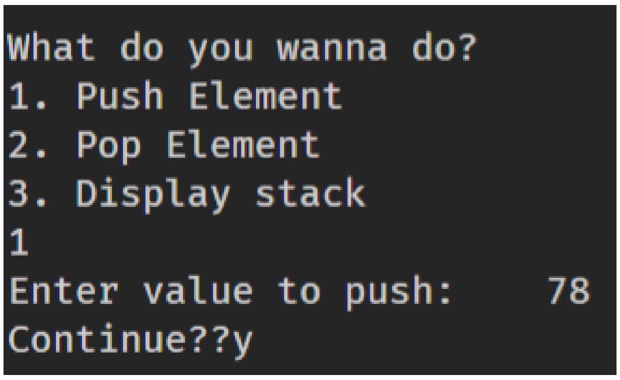
```cpp
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Stack
{
private:
    /*
    * arr is the stack storage
    * top points to last pushed element
    * stackSize is the max-size of stack
    */
    int *arr;
    int top, stackSize;

public:
    Stack(int size)
    {
        // use constructor to initialize max-size of stack
        // using dynamic allocation, deallocating memory
        // in destructor
        stackSize = size;
        arr = new int[size];
        top = -1;
    }

    // insert element in stack
    void push(int el)
    {
        if (top >= stackSize)
        {
            cout << "Stack full!!\n";
            return;
        }
        arr[++top] = el;
    }

    // remove last pushed element from stack
    void pop()
    {
        if (top < 0)
        {
            cout << "Stack empty!!\n";
            return;
        }
        cout << arr[top--] << " was popped\n";
    }

    // print all elements in stack
    // int the direction they'll be popped
    void display()
    {
        if (top < 0)
        {
            cout << "Stack empty!!\n";
            return;
        }

        int counter = top;
        while (counter >= 0)
        {
            cout << arr[counter--] << "->";
        }
        cout << '\n';
    }

    int peep(int index)
    {
        if (top < index)
        {
            cout << "Stack full!!\n";
            return -1;
        }

        return arr[index];
    }

    ~Stack()
    {
        // deallocate arr memory which was allocated
        // in constructor
        cout << "Destructor";
        delete[] arr;
    }
};

int main()
{
    int size;
    cout << "What is the size of the stack??\t";
    cin >> size;

    // initialize stack size as entered by user
    Stack s(size);

    do
    {
        cout << "What do you wanna do?\n";
        cout << "1. Push Element\n";
        cout << "2. Pop Element\n";
        cout << "3. Display stack\n";
        cout << "4. Exit\n";

        // choice is the index of above printed options
        int choice;
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "Enter value to push:\t";
            int el;
            cin >> el;
            s.push(el);
            break;
        case 2:
            s.pop();
            break;

        case 3:
            s.display();
            break;

        case 4:
            return 0;

        default:
            cout << "Invalid input!!";
            break;
        }

        cout << "\n\n";

    } while (true);

    return 0;
}
```



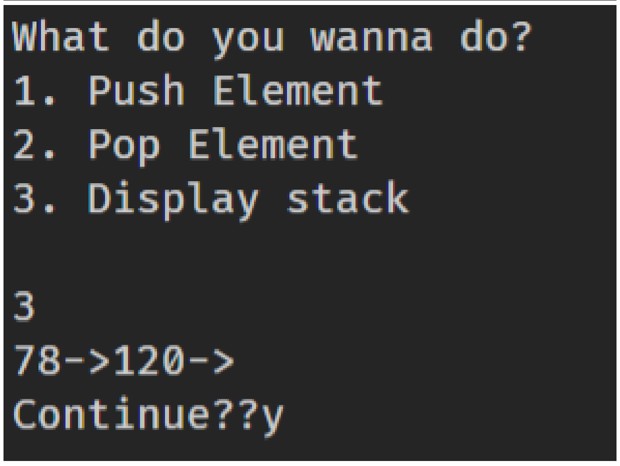
```
What is the size of the stack?? 5
What do you wanna do?
1. Push Element
2. Pop Element
3. Display stack
2
Stack empty!!
Continue??y
```



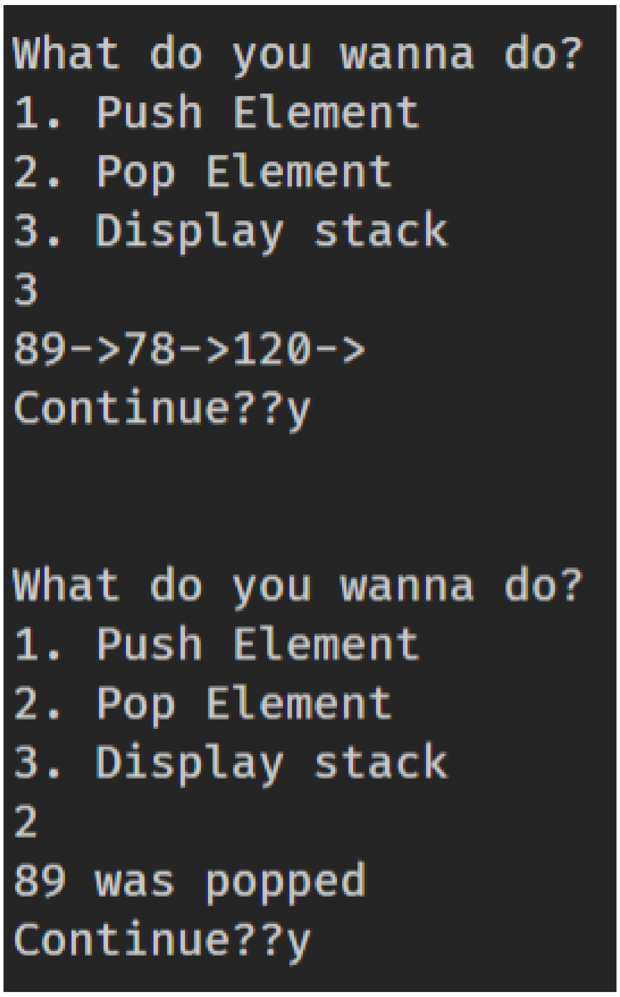
```
What do you wanna do?
1. Push Element
2. Pop Element
3. Display stack
1
Enter value to push:    120
Continue??y
```



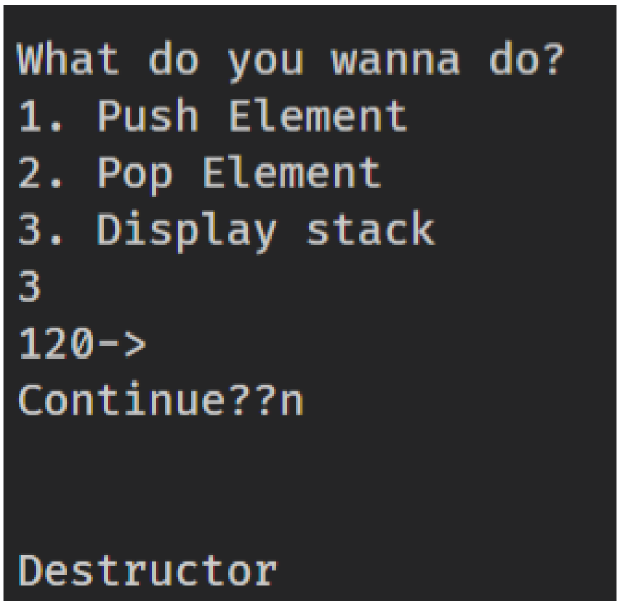
```
What do you wanna do?
1. Push Element
2. Pop Element
3. Display stack
1
Enter value to push:    78
Continue??y
```



```
What do you wanna do?
1. Push Element
2. Pop Element
3. Display stack

3
78->120->
Continue??y
```



```
What do you wanna do?
1. Push Element
2. Pop Element
3. Display stack
3
89->78->120->
Continue??y


What do you wanna do?
1. Push Element
2. Pop Element
3. Display stack
2
89 was popped
Continue??y
```

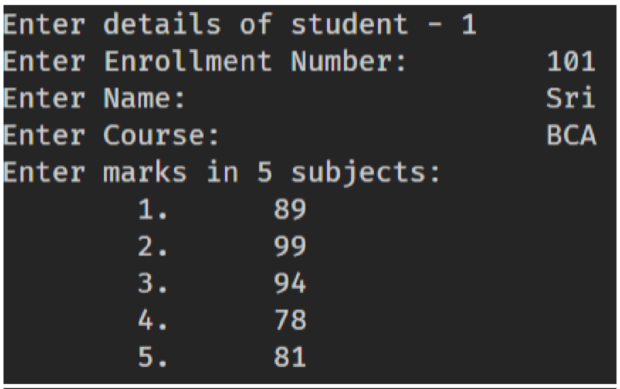


```
What do you wanna do?
1. Push Element
2. Pop Element
3. Display stack
3
120->
Continue??n


Destructor
```

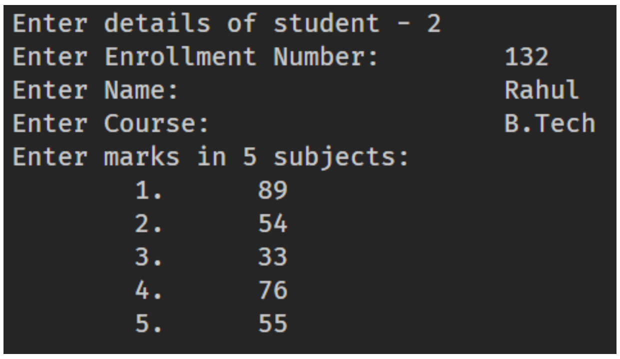
Program - 12

Array of 5 student objects(student class same as Q.1). Also find average percentage of all 5 students.

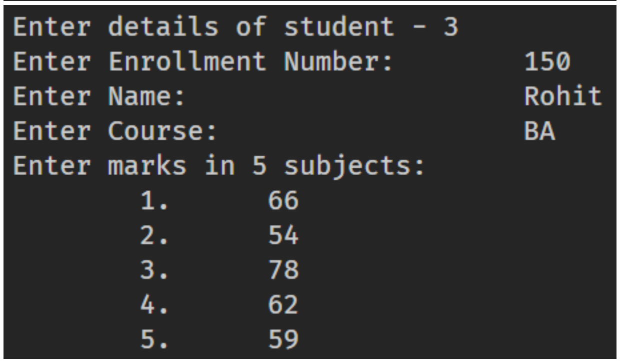
```
/**
* V. Sriram
* 10314902019
*/

/**
 * V. Sriram
 * 10314902019
*/

// Please refer to question 1 for detailed explanation of class

#include <iostream>
using namespace std;

class Student
{
private:
    char name[20], enrolno[15], course[20], grade;
    float marks[5], percentage, total;

    void calculate()
    {
        percentage = total / 5;
        grade = calcGrade(percentage);
    }

    char calcGrade(float percentage)
    {
        if (percentage >= 90)
            return 'O';
        else if (percentage >= 75)
            return 'A';
        else if (percentage >= 60)
            return 'B';
        else if (percentage >= 50)
            return 'C';
        else
            return 'F';
    }

public:
    void input()
    {
        cout << "Enter Enrollment Number:\t";
        cin >> enrolno;

        cout << "Enter Name:\t\t\t";
        cin >> name;

        cout << "Enter Course:\t\t\t";
        cin >> course;

        cout << "Enter marks in 5 subjects:\n";
        for (int i = 1; i <= 5; ++i)
        {
            cout << "\t" << i << ".\t";
            cin >> marks[i - 1];

            total += marks[i - 1];
        }

        calculate();
    }

    void output()
    {
        cout << "Enrollment Number:\t" << enrolno;
        cout << "\nName:\t\t\t" << name;
        cout << "\nCourse:\t\t\t" << course;
        cout << "\nPercentage:\t\t" << percentage;
        cout << "\nGrade:\t\t\t" << grade;
    }

    float getPercentage()
    {
        // return unaccessible private member
        return percentage;
    }

    void printRow()
    {
        // print a row in a tabular form
        printf("%-20s%-20s%-12.2lf%c\n", name, course, percentage, grade);
    }
};

float avgPercentage(Student arr[], int n)
{
    // calculate average percentage of every student
    float totalPercentage = 0;
    for (int i = 0; i < n; ++i)
    {
        totalPercentage += arr[i].getPercentage();
    }
    return totalPercentage / (float)n;
}

int main()
{
    // array of objects of type Student
    Student stuArr[5];

    for (int i = 0; i < 5; ++i)
    {
        cout << "Enter details of student - " << i + 1 << "\n";
        stuArr[i].input();
        cout << "\n\n";
    }

    cout << "Percentage of each student:\n";
    printf("%-20s%-20s%-12s%s\n", "Name", "Course", "Percentage", "Grade");

    for (int i = 0; i < 5; ++i)
        stuArr[i].printRow();

    cout << "Average percentage of all students:\t" << avgPercentage(stuArr, 5);

    return 0;
}
```



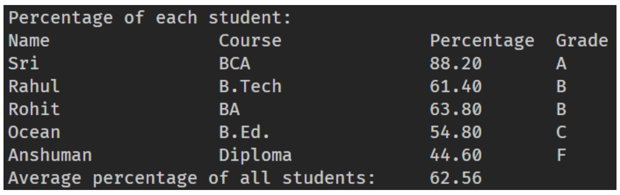
```
Enter details of student - 1
Enter Enrollment Number:        101
Enter Name:                     Sri
Enter Course:                   BCA
Enter marks in 5 subjects:
        1.      89
        2.      99
        3.      94
        4.      78
        5.      81
```



```
Enter details of student - 2
Enter Enrollment Number:        132
Enter Name:                     Rahul
Enter Course:                   B.Tech
Enter marks in 5 subjects:
        1.      89
        2.      54
        3.      33
        4.      76
        5.      55
```



```
Enter details of student - 3
Enter Enrollment Number:        150
Enter Name:                     Rohit
Enter Course:                   BA
Enter marks in 5 subjects:
        1.      66
        2.      54
        3.      78
        4.      62
        5.      59
```

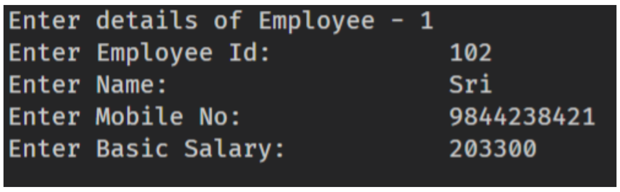


```
Percentage of each student:
Name                Course              Percentage  Grade
Sri                 BCA                 88.20       A
Rahul               B.Tech              61.40       B
Rohit               BA                  63.80       B
Ocean               B.Ed.               54.80       C
Anshuman            Diploma             44.60       F
Average percentage of all students:     62.56
```

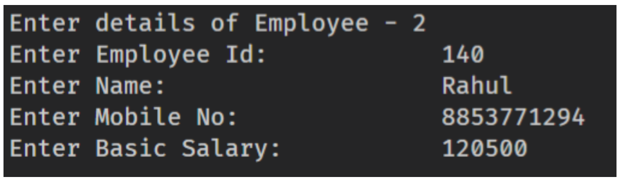
Program - 13

Array of 5 employee objects. (class same as in Q.2)

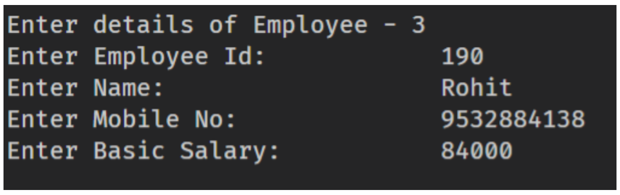
```
/**
* V. Sriram
* 10314902019
*/

/**
 * V. Sriram
 * 10314902019
*/

#include <iostream>
using namespace std;

// see program - 2 for detailed explanation of Employee class
class Employee
{
private:
    int empId;
    double Basic, HRA, DA, Gross;
    char empName[20], mobileNo[11];

    void calculate()
    {
        HRA = Basic * 0.3;
        DA = Basic * 1.15;
        Gross = Basic + DA + HRA;
    }

public:
    void input()
    {
        printf("%-30s", "Enter Employee Id:");
        cin >> empId;

        printf("%-30s", "Enter Name:");
        cin >> empName;

        printf("%-30s", "Enter Mobile No:");
        cin >> mobileNo;

        printf("%-30s", "Enter Basic Salary:");
        cin >> Basic;

        calculate();
    }

    void output()
    {
        printf("%-30s%d", "Employee Id:", empId);
        printf("\n%-30s%s", "Employee Name:", empName);
        printf("\n%-30s%s", "Mobile Id:", mobileNo);
        printf("\n%-30s%.2lf", "Basic Salary:", Basic);
        printf("\n%-30s%.2lf", "Dearness Allowance:", DA);
        printf("\n%-30s%.2lf", "House Rent Allowance:", HRA);
        printf("\n%-30s%.2lf", "Gross Salary:", Gross);
    }
    void outputRow()
    {
        // print a row of data in object
        // in a tabular form
        printf("%-5d%-20s%-13.2lf%-13.2lf%-13.2lf%-13.2lf\n",
               empId, empName, Basic, HRA, DA, Gross);
    }

    double getGross()
    {
        // return inaccessible member of class
        return Gross;
    }
};

void tablulate(Employee eArr[], int n)
{
    // prints data of all employees
    // in a tabular form
    printf("%-5s%-20s%-13s%-13s%-13s%-13s\n",
           "ID", "Name", "Basic", "HRA", "DA", "Gross");
    for (int i = 0; i < n; ++i)
        eArr[i].outputRow();
}

double meanSalary(Employee eArr[], int n)
{
    // calculate mean gross salary of all employees
    double total = 0;
    for (int i = 0; i < n; ++i)
        total += eArr[i].getGross();

    return total / (double)n;
}

int main()
{
    // array of object of type Employee
    Employee eArr[5];

    for (int i = 0; i < 5; ++i)
    {
        cout << "Enter details of Employee - " << i + 1 << '\n';
        eArr[i].input();
        cout << "\n\n";
    }

    cout << "\n\n";
    tablulate(eArr, 5);

    cout << "\n\nMean Salary of all employees  = " << meanSalary(eArr, 5);

    return 0;
}
```



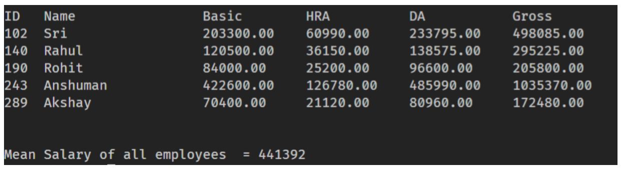
```
Enter details of Employee - 1
Enter Employee Id:            102
Enter Name:                   Sri
Enter Mobile No:              9844238421
Enter Basic Salary:           203300
```



```
Enter details of Employee - 2
Enter Employee Id:            140
Enter Name:                   Rahul
Enter Mobile No:              8853771294
Enter Basic Salary:           120500
```



```
Enter details of Employee - 3
Enter Employee Id:           190
Enter Name:                  Rohit
Enter Mobile No:             9532884138
Enter Basic Salary:          84000
```

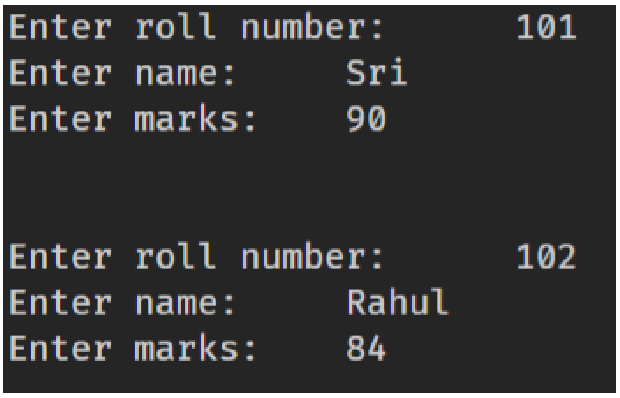


```
ID   Name                 Basic        HRA          DA           Gross
102  Sri                  203300.00    60990.00     233795.00    498085.00
140  Rahul                120500.00    36150.00     138575.00    295225.00
190  Rohit                84000.00     25200.00     96600.00     205800.00
243  Anshuman             422600.00    126780.00    485990.00    1035370.00
289  Akshay               70400.00     21120.00     80960.00     172480.00


Mean Salary of all employees  = 441392
```

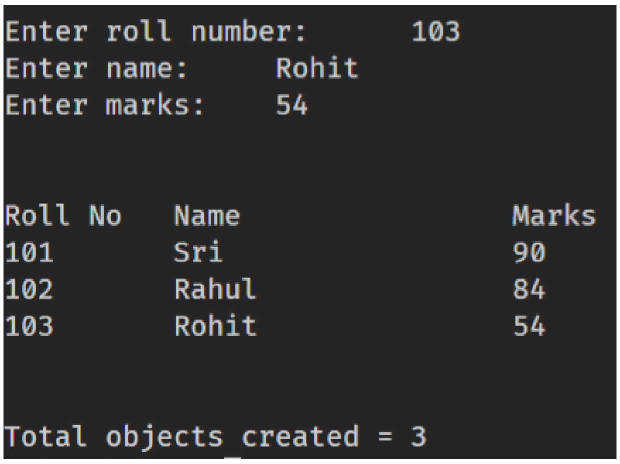
Program - 14

WAP to demonstrate a concept of static data member.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>

using namespace std;

class Student
{
private:
    int rollNo;
    char name[10];
    int marks;
    // objectCount is a variable of class
    // only 1 instance is made
    static int objectCount;

public:
    Student()
    {
        // objectCount is incremented every time
        // an object of type Student is declared
        objectCount++;
    }

    void get()
    {
        // get data as input from user
        cout << "Enter roll number:\t";
        cin >> rollNo;
        cout << "Enter name:\t";
        cin >> name;
        cout << "Enter marks:\t";
        cin >> marks;
    }

    void put()
    {
        // print object data
        printf("%-10d%-20s%-10d\n", rollNo, name, marks);
    }

    static int getObjectCount()
    {
        // static member function can only access
        // static data members
        return objectCount;
    }
};
int Student::objectCount = 0;

int main()
{
    // program shows that value of objectCount remains
    // incrementing after each object declaration
    Student s1;
    s1.get();
    cout << "\n\n";

    Student s2;
    s2.get();
    cout << "\n\n";

    Student s3;
    s3.get();
    cout << "\n\n";

    printf("%-10s%-20s%-10s\n", "Roll No", "Name", "Marks");
    s1.put();
    s2.put();
    s3.put();

    cout << "\n\n";
    // access static member function
    // and thus the static data member
    cout << "Total objects created = " << Student::getObjectCount();
    return 0;
}
```



```
Enter roll number:      101
Enter name:     Sri
Enter marks:    90


Enter roll number:      102
Enter name:     Rahul
Enter marks:    84
```

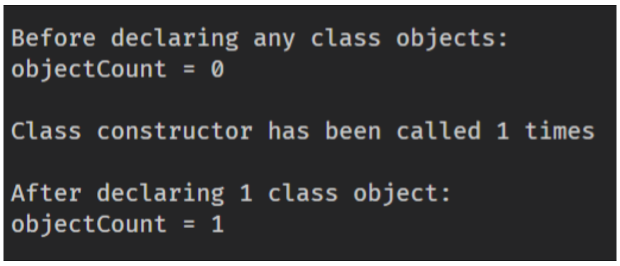


```
Enter roll number:      103
Enter name:     Rohit
Enter marks:    54


Roll No   Name                Marks
101       Sri                 90
102       Rahul               84
103       Rohit               54


Total objects created = 3
```

Program - 15/16

Demonstrate that the static variable will exist even before creating a object and Demonstrate static member function concept.

```cpp
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

// a test class
class Class
{
private:
    // static data member
    static int objectCount;
    char name[10];

public:
    Class()
    {
        // constructor of class increments ObjectCount
        // each time new object is declared
        objectCount++;
        cout << "Class constructor has been called " << objectCount << " times\n\n";
    }

    void get()
    {
        cout << "Enter name";
        cin >> name;
    }

    void put()
    {
        cout << "Name:\t" << name;
    }

    static int getObjectCount()
    {
        // statuc member function can only access
        // status data members
        return objectCount;
    }
};
int Class::objectCount = 0;

int main()
{
    // prog demonstrated that static data member
    // exists even before an object is of class
    // is created
    cout << "\n\nBefore declaring any class objects:\n";
    cout << "objectCount = " << Class::getObjectCount() << "\n\n";

    Class c;

    cout << "After declaring 1 class object:\n";
    cout << "objectCount = " << Class::getObjectCount() << "\n\n";

    return 0;
}
```



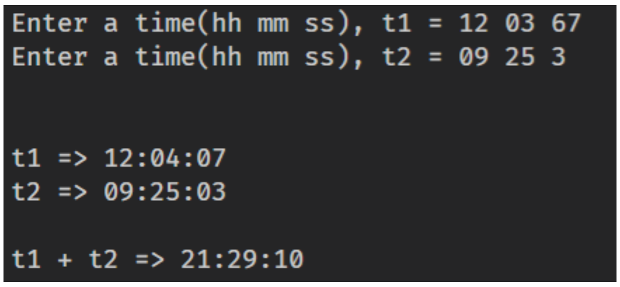
```
Before declaring any class objects:
objectCount = 0

Class constructor has been called 1 times

After declaring 1 class object:
objectCount = 1
```

Program - 17

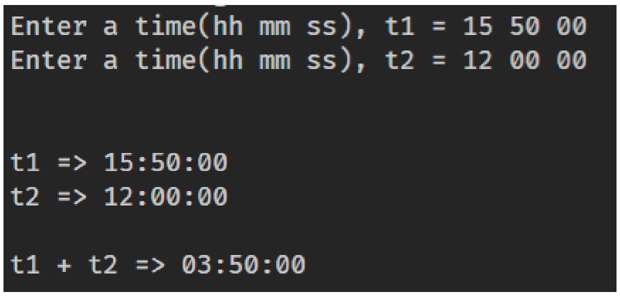
Pass Object(Time) as function arguments,find sum(hrs,min,sec)using member function of Time class.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Time
{
private:
    int h, m, s;

public:
    Time(int hours = 0, int minutes = 0, int seconds = 0)
    {
        if (hours < 0 || minutes < 0 || seconds < 0)
            throw "Invalid time. Values can't be negative";

        h = hours;
        m = minutes;
        s = seconds;
        normlize();
    }

    // convert overflowed values to standard
    // seconds <= 60
    // minutes <= 60
    // hours <= 24
    void normlize()
    {
        if (s >= 60)
        {
            m += s / 60;
            s %= 60;
        }
        if (m >= 60)
        {
            h += m / 60;
            m %= 60;
        }
        if (h >= 24)
        {
            h %= 24;
        }
    }

    // display in format hh:mm:ss
    void display()
    {
        printf("%02d:%02d:%02d\n", h, m, s);
    }

    // sum of 2 time objects
    Time sum(const Time &t)
    {
```

```cpp
        return Time(this->h + t.h, this->m + t.m, this->s + t.s);
    };
};

int main()
{
    // get input
    int h, m, s;
    cout << "Enter a time(hh mm ss), t1 = ";
    cin >> h >> m >> s;
    Time t1(h, m, s);

    cout << "Enter a time(hh mm ss), t2 = ";
    cin >> h >> m >> s;
    Time t2(h, m, s);

    // display
    cout << "\n\nt1 => ";
    t1.display();

    cout << "t2 => ";
    t2.display();

    Time t3 = t1.sum(t2);
    cout << "\nt1 + t2 => ";
    t3.display();

    return 0;
}
```



```
Enter a time(hh mm ss), t1 = 12 03 67
Enter a time(hh mm ss), t2 = 09 25 3


t1 => 12:04:07
t2 => 09:25:03

t1 + t2 => 21:29:10
```

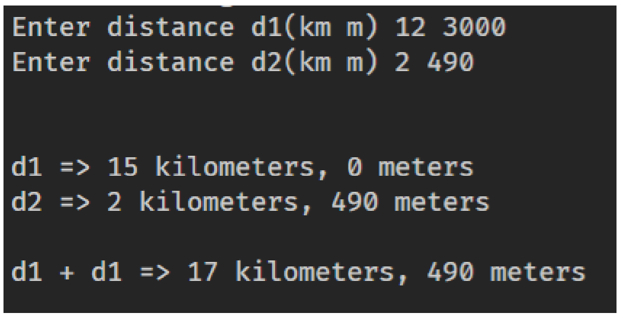


```
Enter a time(hh mm ss), t1 = 15 50 00
Enter a time(hh mm ss), t2 = 12 00 00


t1 => 15:50:00
t2 => 12:00:00

t1 + t2 => 03:50:00
```

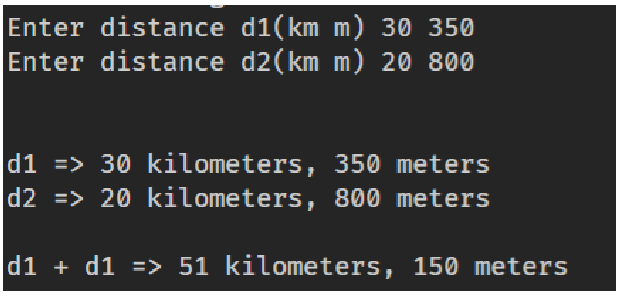
Program - 18

Pass Object(Distance) as function arguments,find sum(km,m)using member function of Distanceclass.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Distance
{
private:
    int km, m;

public:
    Distance(int kilometer = 0, int meter = 0)
    {
        km = kilometer;
        m = meter;

        normalize();
    }

    // convert overflowed values to standard
    // meters <= 1000
    void normalize()
    {
        if (m >= 1000)
        {
            km += m / 1000;
            m %= 1000;
        }
    }

    // display in format km kilometers, m meters
    void display()
    {
        cout << km << " kilometers, " << m << " meters\n";
    }

    // get sum of 2 distance objects
    Distance sum(Distance &d)
    {
        return Distance(this->km + d.km, this->m + d.m);
    }
};

int main()
{
    // get input
    int km, m;
    cout << "Enter distance d1(km m)\t";
    cin >> km >> m;
    Distance d1(km, m);

    cout << "Enter distance d2(km m)\t";
    cin >> km >> m;
    Distance d2(km, m);

    // diplay results
    cout << "\n\nd1 => ";
    d1.display();

    cout << "d2 => ";
    d2.display();

    Distance d3 = d1.sum(d2);
    cout << "\nd1 + d1 => ";
    d3.display();

    return 0;
}
```



```
Enter distance d1(km m) 12 3000
Enter distance d2(km m) 2 490


d1 => 15 kilometers, 0 meters
d2 => 2 kilometers, 490 meters

d1 + d1 => 17 kilometers, 490 meters
```



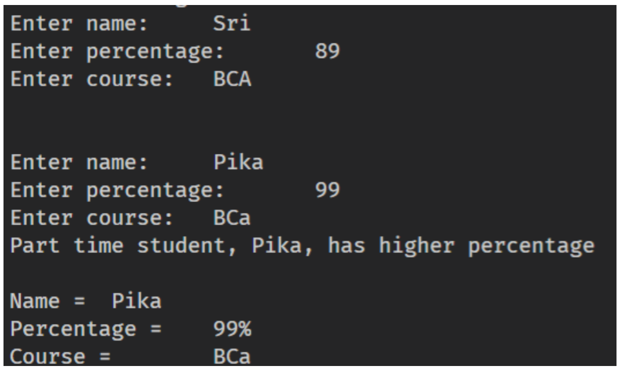
```
Enter distance d1(km m) 30 350
Enter distance d2(km m) 20 800


d1 => 30 kilometers, 350 meters
d2 => 20 kilometers, 800 meters

d1 + d1 => 51 kilometers, 150 meters
```

Program - 19

Create two classes pt_std and ft_std. Thereis a friend fun() which is friend of both the classes and it displays all the details of the student having greater marks.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>

using namespace std;

class PT; // declaration of part time class
class FT
{ // declaration of full time class and it's definition
    char name[15];
    float percentage;
    char course[10];

public:
    void input()
    { // Getting input from user
        cout << "Enter name:\t";
        cin >> name;
        cout << "Enter percentage:\t";
        cin >> percentage;
        cout << "Enter course:\t";
        cin >> course;
    }
    void output()
    { // showing output to the user
        cout << "\nName = \t" << name;
        cout << "\nPercentage =\t" << percentage << "%";
        cout << "\nCourse =\t" << course;
    }
    friend void Compare(FT, PT); // friend function signature
};

class PT
{
    char name[15];
    float percentage;
    char course[10];

public:
    void input()
    { // Getting input into data members
        cout << "Enter name:\t";
        cin >> name;
        cout << "Enter percentage:\t";
        cin >> percentage;
        cout << "Enter course:\t";
        cin >> course;
    }
    void output()
    { // Showing output to the user
        cout << "\nName = \t" << name;
        cout << "\nPercentage = \t" << percentage << "%";
        cout << "\nCourse =\t" << course;
```

```cpp
    }
    friend void Compare(FT, PT); // friend function signature
};

void Compare(FT f, PT p)
{ // friend function definition
    if (f.percentage > p.percentage)
    { // Condition to check highest marks
        cout << "Full time student, " << f.name << ", has higher percentage" << endl;
        f.output();
    }
    else if (f.percentage < p.percentage)
    {
        cout << "Part time student, " << p.name << ", has higher percentage" << endl;
        p.output();
    }
    else
    {
        cout << "Both have same percentage" << endl;
        f.output();
        cout << "\n";
        p.output();
    }
}

int main()
{
    FT ff; // object initialization
    PT pp;

    ff.input();
    cout << "\n\n";
    pp.input();
    Compare(ff, pp); // friend function calling

    return 0;
}
```



```
Enter name:     Sri
Enter percentage:       89
Enter course:   BCA


Enter name:     Pika
Enter percentage:       99
Enter course:   BCa
Part time student, Pika, has higher percentage

Name =  Pika
Percentage =    99%
Course =        BCa
```

Program - 20

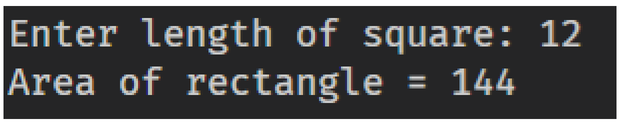
Demonstrate the concept of Member function of one class as a friend function of other class.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class A; // defined later

class B
{
public:
    void change(A &yc, int x); // defined later
};

class A
{
private:
    int topSecret;

public:
    A()
    {
        topSecret = 0;
    }
    void printMember() { cout << "Top Secret Code = " << topSecret << endl; }

    // member function of B is a friend of A class
    friend void B::change(A &, int);
};

// can access all members of class A
void B::change(A &yc, int x) { yc.topSecret = x; }

int main()
{
    A a;
    B b;

    a.printMember();
    b.change(a, 5);
    a.printMember();

    return 0;
}
```

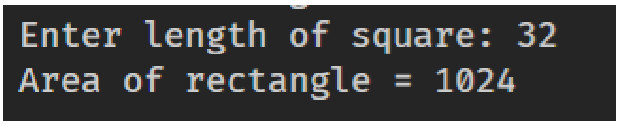
Program - 21

Demonstrate the concept of friend class.

```cpp
/**
* V. Sriram
* 10314902019
*/

#include <iostream>

using namespace std;

class Square
{
    friend class Rectangle; // declaring Rectangle as friend class
    int side;

public:
    Square(int s = 0)
    {
        side = s;
    }

    void input()
    {
        cout << "Enter length of square:\t";
        cin >> side;
    }
};

class Rectangle
{
    int length;
    int breadth;

public:
    int getArea()
    {
        return length * breadth;
    }
    void shape(Square s) // can access private members of Square object s
    {
        s.input();
        length = s.side;
        breadth = s.side;
    }
};

int main()
{
    Square square;
    Rectangle rectangle;

    rectangle.shape(square);
    cout << "Area of rectangle = " << rectangle.getArea() << endl;

    return 0;
}
```



```
Enter length of square: 12
Area of rectangle = 144
```



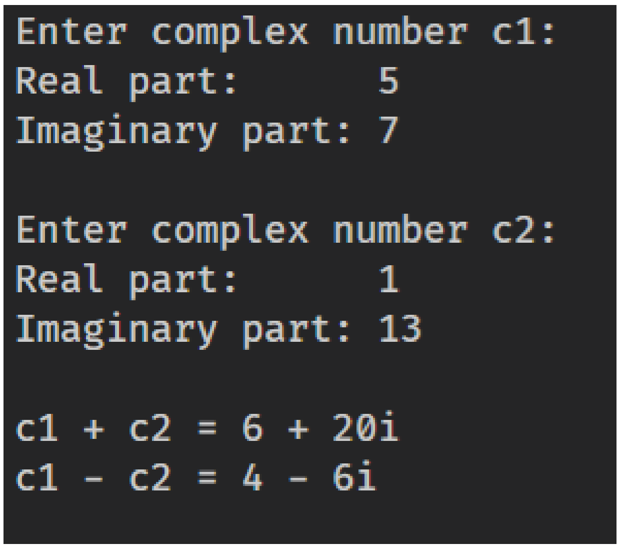
```
Enter length of square: 32
Area of rectangle = 1024
```

Program - 22

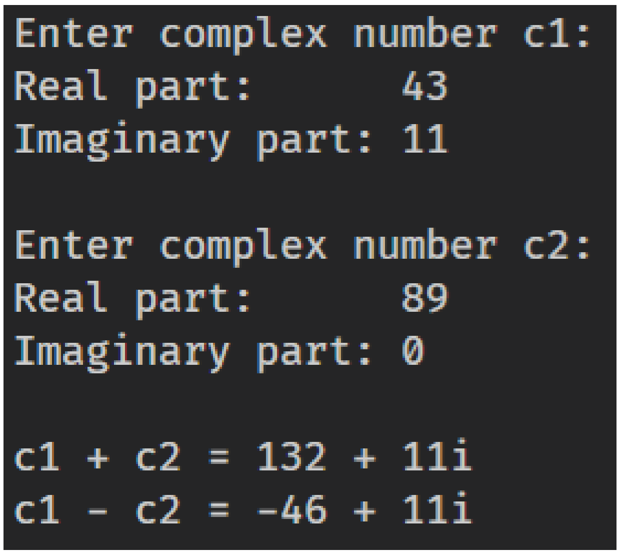
WAP to create a class named complex(data mem-real & imag of int type,Member fun()-input() & output(), friend fun()-add() & subtract()).

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Complex
{
private:
    int real, imag;

public:
    Complex() {}

    Complex(int r, int i)
    {
        real = r;
        imag = i;
    }

    void input()
    {
        cout << "Real part:\t";
        cin >> real;

        cout << "Imaginary part:\t";
        cin >> imag;
        cout << '\n';
    }

    void output()
    {
        cout << real;
        if (imag < 0)
            cout << " - " << -imag;
        else
            cout << " + " << imag;

        cout << "i\n";
    }

    // using friend classes to add and substract 2 Complex instances
    friend Complex add(const Complex &c1, const Complex &c2);
    friend Complex subtract(const Complex &c1, const Complex &c2);
};

// can access all members of any Complex class object
Complex add(const Complex &c1, const Complex &c2)
{
    return Complex(c1.real + c2.real, c1.imag + c2.imag);
}

// can access all members of any Complex class object
Complex subtract(const Complex &c1, const Complex &c2)
```

```cpp
{
    return Complex(c1.real - c2.real, c1.imag - c2.imag);
}

int main()
{
    Complex c1, c2;

    cout << "Enter complex number c1:\n";
    c1.input();

    cout << "Enter complex number c2:\n";
    c2.input();

    Complex c3 = add(c1, c2);
    Complex c4 = subtract(c1, c2);

    cout << "c1 + c2 = ";
    c3.output();

    cout << "c1 - c2 = ";
    c4.output();

    return 0;
}
```



```
Enter complex number c1:
Real part:      5
Imaginary part: 7

Enter complex number c2:
Real part:      1
Imaginary part: 13

c1 + c2 = 6 + 20i
c1 - c2 = 4 - 6i
```

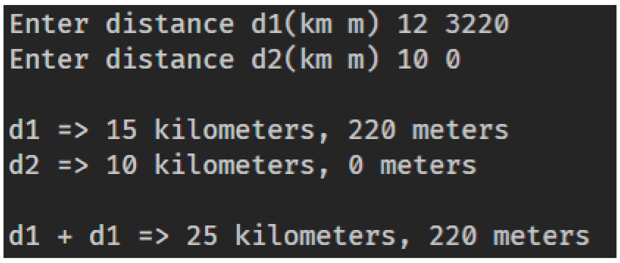


```
Enter complex number c1:
Real part:      43
Imaginary part: 11

Enter complex number c2:
Real part:      89
Imaginary part: 0

c1 + c2 = 132 + 11i
c1 - c2 = -46 + 11i
```

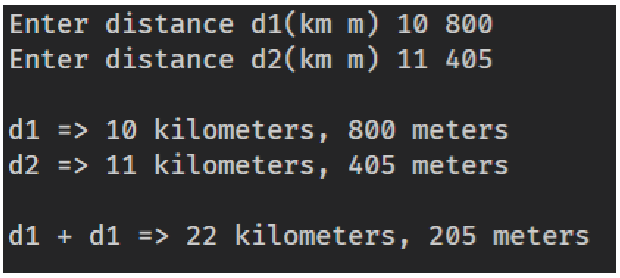
Program - 23

WAP to add two distance objects using friend function.

```cpp
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Distance
{
private:
    int km, m;

public:
    Distance(int kilometer, int meter)
    {
        km = kilometer;
        m = meter;

        normalize();
    }

    // convert overflowed values to standard
    // meters <= 1000
    void normalize()
    {
        if (m >= 1000)
        {
            km += m / 1000;
            m %= 1000;
        }
    }

    // display in format km kilometers, m meters
    void display()
    {
        cout << km << " kilometers, " << m << " meters\n";
    }

    // using friend function to calculate sum of 2 Distance objects
    friend Distance sum(Distance &, Distance &);
};

Distance sum(Distance &d1, Distance &d2)
{
    // returns a nameless instance
    return Distance(d1.km + d2.km, d1.m + d2.m);
}
int main()
{
    int km, m;
    cout << "Enter distance d1(km m)\t";
    cin >> km >> m;
    Distance d1(km, m);

    cout << "Enter distance d2(km m)\t";
    cin >> km >> m;
    Distance d2(km, m);

    cout << "\nd1 => ";
    d1.display();

    cout << "d2 => ";
    d2.display();

    Distance d3 = sum(d1, d2);
    cout << "\nd1 + d1 => ";
    d3.display();

    return 0;
}
```



```
Enter distance d1(km m) 12 3220
Enter distance d2(km m) 10 0

d1 => 15 kilometers, 220 meters
d2 => 10 kilometers, 0 meters

d1 + d1 => 25 kilometers, 220 meters
```

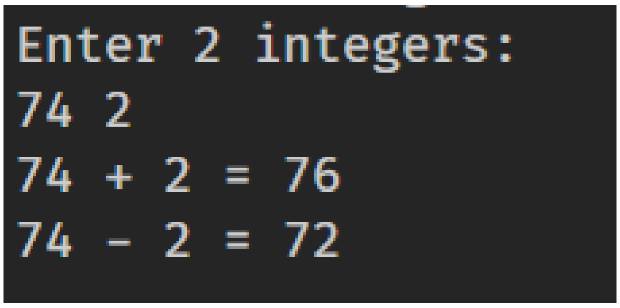


```
Enter distance d1(km m) 10 800
Enter distance d2(km m) 11 405

d1 => 10 kilometers, 800 meters
d2 => 11 kilometers, 405 meters

d1 + d1 => 22 kilometers, 205 meters
```

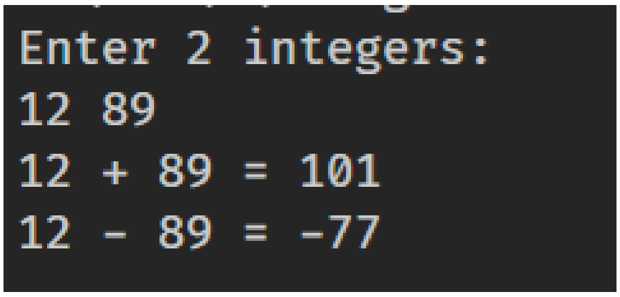
Program - 24

WAP to illustrate dynamic memory allocation and deallocation using new and delete operator to find out the sum of two integers and their difference.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

int main()
{
    int *p1, *p2;
    // dynamically allocate 2 integers
    p1 = new int;
    p2 = new int;

    cout << "Enter 2 integers:\n";
    cin >> *p1 >> *p2;

    cout << *p1 << " + " << *p2 << " = " << (*p1 + *p2) << '\n';
    cout << *p1 << " - " << *p2 << " = " << (*p1 - *p2) << '\n';

    // deallocate the integers
    delete p1;
    delete p2;

    return 0;
}
```



```
Enter 2 integers:
74 2
74 + 2 = 76
74 - 2 = 72
```

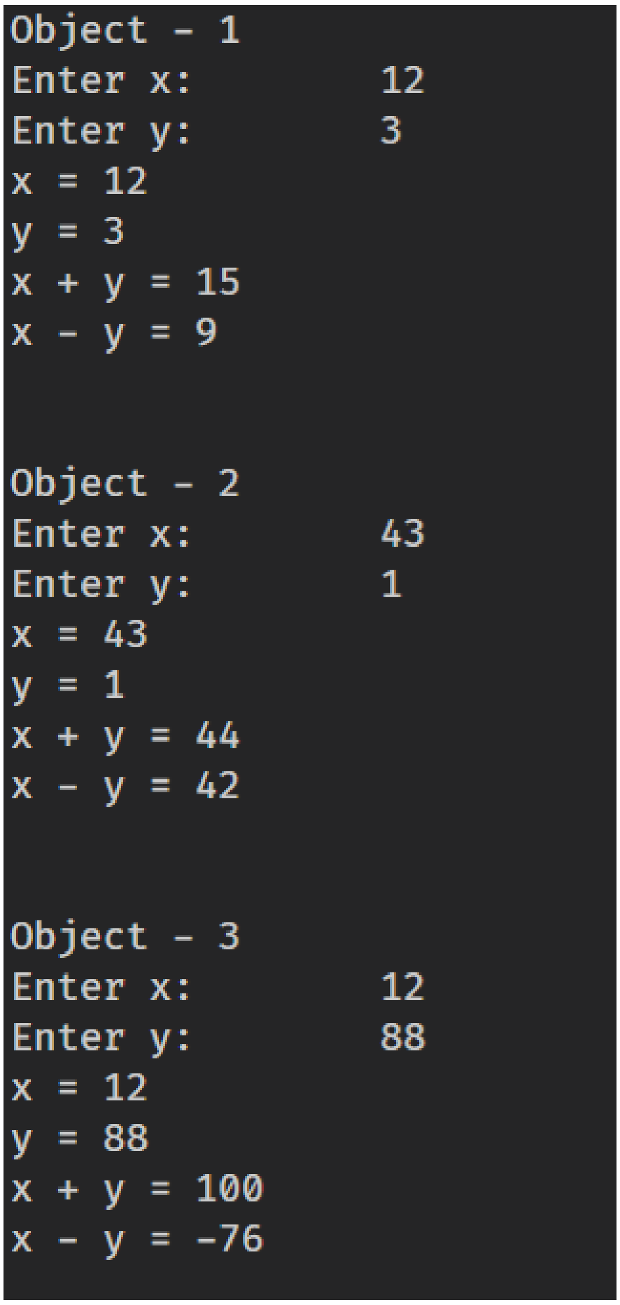


```
Enter 2 integers:
12 89
12 + 89 = 101
12 - 89 = -77
```

Program - 25

WAP to demonstrate the concept of array of objects allocated dynamically , the member functions are to be invoked using 3 different ways.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Dynam
{
private:
    int x, y;

public:
    void input()
    {
        cout << "Enter x:\t";
        cin >> x;

        cout << "Enter y:\t";
        cin >> y;
    }

    void output()
    {
        cout << "x = " << x << "\n";
        cout << "y = " << y << "\n";

        cout << "x + y"
             << " = " << x + y;
        cout << '\n';

        cout << "x - y"
             << " = " << x - y;
        cout << '\n';
    }
};

int main()
{
    int size = 3;

    // dynamically allocate block of memory
    // each unit is of type Dynam
    Dynam *p = new Dynam[size];

    cout << "Object - " << 1 << endl;
    p[0].input();
    p[0].output();
    cout << "\n\n";

    cout << "Object - " << 2 << endl;
    (p + 1)->input();
    (p + 1)->output();
    cout << "\n\n";
    cout << "Object - " << 3 << endl;
    (*(p + 2)).input();
    (*(p + 2)).output();
    cout << "\n\n";

    // deallocate the block of memory
    delete[] p;

    return 0;
}
```

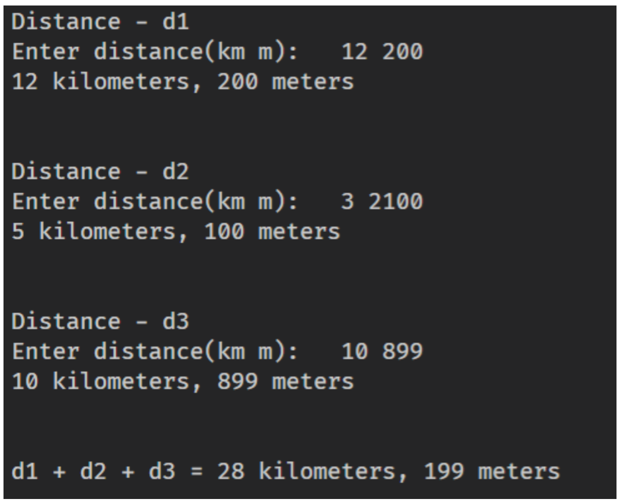


```
Object - 1
Enter x:        12
Enter y:        3
x = 12
y = 3
x + y = 15
x - y = 9


Object - 2
Enter x:        43
Enter y:        1
x = 43
y = 1
x + y = 44
x - y = 42


Object - 3
Enter x:        12
Enter y:        88
x = 12
y = 88
x + y = 100
x - y = -76
```

Program - 26

For the class distance request for a block of memory dynamically and demonstrate invoking the member functions using 3 different ways.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Distance
{
private:
    int km, m;

public:
    Distance(int kilometer = 0, int meter = 0)
    {
        km = kilometer;
        m = meter;

        normalize();
    }

    void input()
    {
        cout << "Enter distance(km m):\t";
        cin >> km >> m;
        normalize();
    }

    // convert overflowed values to standard
    // meters <= 1000
    void normalize()
    {
        if (m >= 1000)
        {
            km += m / 1000;
            m %= 1000;
        }
    }

    // display in format km kilometers, m meters
    void display()
    {
        cout << km << " kilometers, " << m << " meters\n";
    }

    // get sum of 2 distance objects
    Distance sum(Distance &d)
    {
        return Distance(this->km + d.km, this->m + d.m);
    }
};

int main()
{
    Distance *p = new Distance[4];
    cout << "Distance - d1\n";
    p[0].input();
    p[0].display();
    cout << "\n\n";

    cout << "Distance - d2\n";
    (p + 1)->input();
    (p + 1)->display();
    cout << "\n\n";

    cout << "Distance - d3\n";
    (*(p + 2)).input();
    (*(p + 2)).display();
    cout << "\n\n";

    // sum of all three
    cout << "d1 + d2 + d3 = ";
    p[3] = p[0].sum(p[1]).sum(p[2]);
    p[3].display();

    delete[] p;

    return 0;
}
```

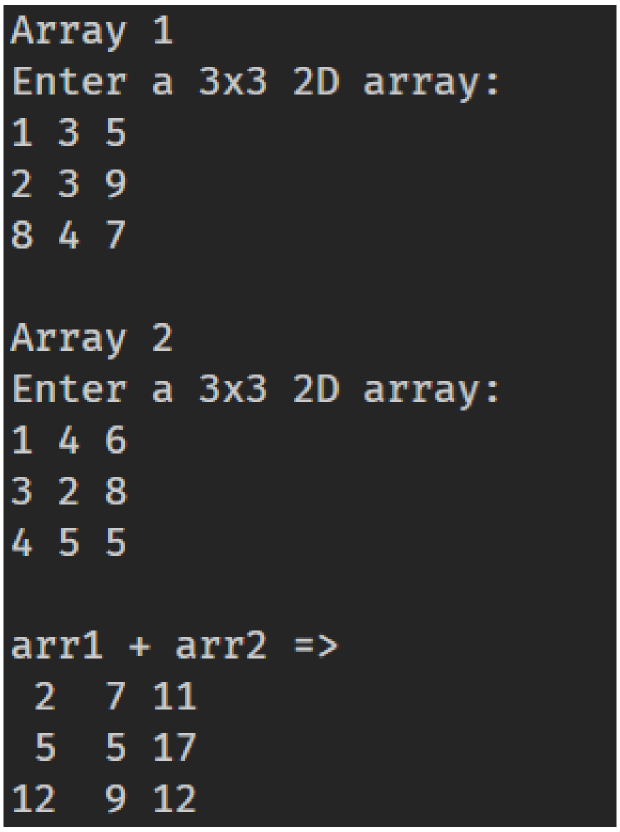


```
Distance - d1
Enter distance(km m):   12 200
12 kilometers, 200 meters


Distance - d2
Enter distance(km m):   3 2100
5 kilometers, 100 meters


Distance - d3
Enter distance(km m):   10 899
10 kilometers, 899 meters


d1 + d2 + d3 = 28 kilometers, 199 meters
```

Program - 27

ARRAY OF ARRAYS(2D Array)

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
#include <iomanip>
#define R 3
#define C 3

using namespace std;

// take a 2D array input
void get2DArray(int arr[][C], int r, int c)
{
    cout << "Enter a " << r << "x" << c << " 2D array:\n";

    for (int i = 0; i < r; ++i)
        for (int j = 0; j < c; ++j)
            cin >> arr[i][j];
}

// calculate sum of given arrays and
// store in res which can be accessed by the caller
void sumOf2DArrays(int arr1[][C], int arr2[][C], int r, int c, int res[][C])
{
    for (int i = 0; i < r; ++i)
        for (int j = 0; j < c; ++j)
            res[i][j] = arr1[i][j] + arr2[i][j];
}

// print an array
void print2DArray(int arr[][C], int r, int c)
{
    for (int i = 0; i < r; ++i)
    {
        for (int j = 0; j < c; ++j)
            cout << right << setw(2) << arr[i][j] << ' ';
        cout << endl;
    }
}

int main()
{
    // Array of Arrays
    int arr1[R][C], arr2[R][C], sumArr[R][C];

    cout << "Array 1\n";
    get2DArray(arr1, R, C);

    cout << "\nArray 2\n";
    get2DArray(arr2, R, C);

    cout << "\narr1 + arr2 => \n";
    sumOf2DArrays(arr1, arr2, R, C, sumArr);
    print2DArray(sumArr, R, C);
    return 0;
}
```

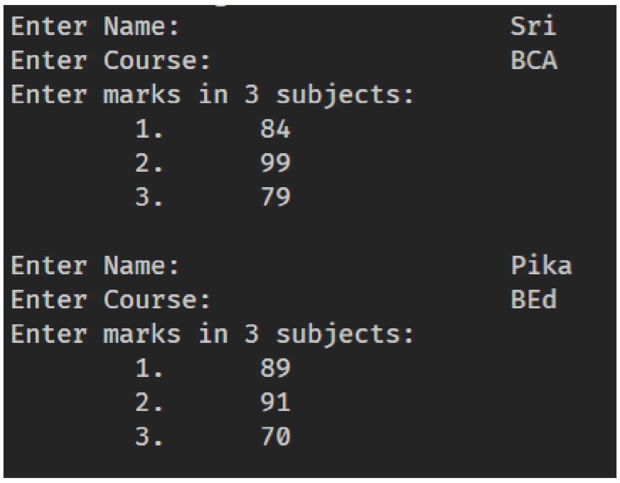
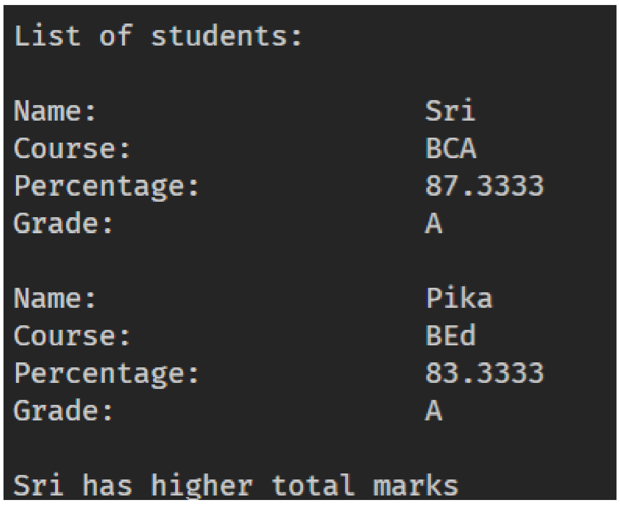


```
Array 1
Enter a 3x3 2D array:
1 3 5
2 3 9
8 4 7

Array 2
Enter a 3x3 2D array:
1 4 6
3 2 8
4 5 5

arr1 + arr2 =>
 2  7 11
 5  5 17
12  9 12
```

Program - 28

Compare two objects of student class using this pointer.

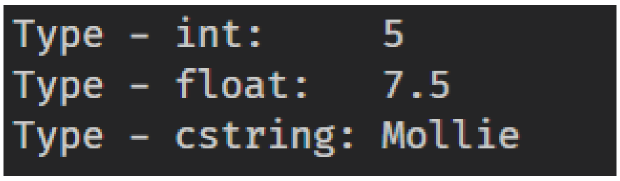
```cpp
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Student
{
private:
    char name[50], course[50], grade;
    float marks[3], percentage, total;

    void calculate()
    {
        // calculate percentage as
        // sum of marks / no. of subjects
        // stores values in data members
        percentage = total / 3;
        grade = calcGrade(percentage);
    }

    char calcGrade(float percentage)
    {
        /* grade limits:
        * >= 90 : O
        * >= 75 : A
        * >= 60 : B
        * >= 50 : C
        * < 50> : F
        */
        if (percentage >= 90)
            return 'O';
        else if (percentage >= 75)
            return 'A';
        else if (percentage >= 60)
            return 'B';
        else if (percentage >= 50)
            return 'C';
        else
            return 'F';
    }

public:
    void input()
    {
        // get input into all data members of the object
        cout << "Enter Name:\t\t\t";
        cin >> name;

        cout << "Enter Course:\t\t\t";
        cin >> course;

        cout << "Enter marks in 3 subjects:\n";
        for (int i = 1; i <= 3; ++i)
        {
            cout << "\t" << i << ".\t";
            cin >> marks[i - 1];

            total += marks[i - 1];
        }

        calculate();
    }

    void output()
    {
        // print output to console after calculation
        cout << "\nName:\t\t\t" << name;
        cout << "\nCourse:\t\t\t" << course;
        cout << "\nPercentage:\t\t" << percentage;
        cout << "\nGrade:\t\t\t" << grade;
    }

    Student &max(Student &s)
    {
        // compare and return the student object
        // with highest total marks
        if (this->total < s.total)
            return s;

        return *this;
    }

    char *getName()
    {
        // return student name
        return name;
    }
};

int main()
{
    Student s1, s2;
    s1.input();
    cout << "\n";
    s2.input();

    cout << "\n\nList of students:\n";

    s1.output();
    cout << "\n";
    s2.output();

    cout << "\n\n";

    cout << s1.max(s2).getName() << " has higher total marks";

    return 0;
}
```


```
Enter Name:                      Sri
Enter Course:                    BCA
Enter marks in 3 subjects:
        1.      84
        2.      99
        3.      79

Enter Name:                      Pika
Enter Course:                    BEd
Enter marks in 3 subjects:
        1.      89
        2.      91
        3.      70
```

```
List of students:

Name:                   Sri
Course:                 BCA
Percentage:             87.3333
Grade:                  A

Name:                   Pika
Course:                 BEd
Percentage:             83.3333
Grade:                  A

Sri has higher total marks
```

Program - 29

Demonstate the concept of void pointer.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

enum class Type
{
    INT,
    FLOAT,
    CSTRING
};

void printValue(void *ptr, Type type)
{
    switch (type)
    {
    case Type::INT:
        cout << "Type - int:\t" << *(int *)(ptr) << '\n'; // cast to int pointer and
        dereference
        break;
    case Type::FLOAT:
        cout << "Type - float:\t" << *(float *)(ptr) << '\n'; // cast to float
        pointer and dereference
        break;
    case Type::CSTRING:
        cout << "Type - cstring:\t" << (char *)(ptr) << '\n'; // cast to char
        pointer (no dereference)
        // cout knows to treat char* as a C-style string
        // if we were to dereference the result, then we'd just print the single
        char that ptr is pointing to
        break;
    }
}

int main()
{
    int nValue = 5;
    float fValue = 7.5f;
    char szValue[] = "Mollie";

    printValue(&nValue, Type::INT);
    printValue(&fValue, Type::FLOAT);
    printValue(szValue, Type::CSTRING);

    return 0;
}
```

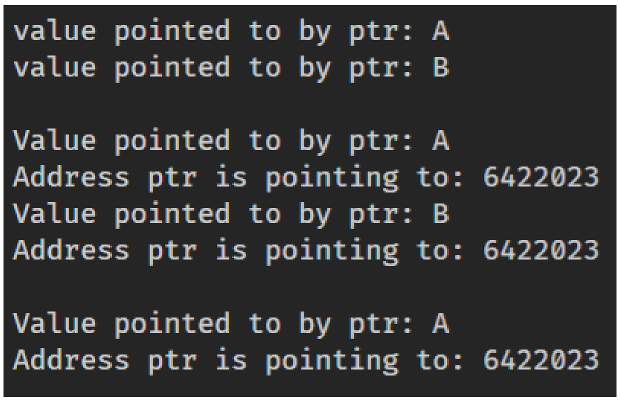


```
Type - int:     5
Type - float:   7.5
Type - cstring: Mollie
```

Program - 30

Demonstate the concept of constantpointer, pointer to constant, constant pointer to constant.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

int main()
{
    // POINTER TO CONSTANT
    char a = 'A', b = 'B';
    const char *ptc = &a;

    //*ptr = b; illegal statement (assignment of read-only location *ptr)

    // ptr can be changed
    printf("value pointed to by ptr: %c\n", *ptc);
    ptc = &b;
    printf("value pointed to by ptr: %c\n\n", *ptc);

    // ==========================================================================
    // CONST POINTER
    a = 'A', b = 'B';
    char *const cp = &a;
    printf("Value pointed to by ptr: %c\n", *cp);
    printf("Address ptr is pointing to: %d\n", cp);

    //ptr = &b; illegal statement (assignment of read-only variable ptr)

    // changing the value at the address ptr is pointing to
    *cp = b;
    printf("Value pointed to by ptr: %c\n", *cp);
    printf("Address ptr is pointing to: %d\n\n", cp);

    // ==========================================================================
    // CONST POINTER TO CONST
    a = 'A', b = 'B';
    const char *const cptc = &a;

    printf("Value pointed to by ptr: %c\n", *cptc);
    printf("Address ptr is pointing to: %d\n\n", cptc);

    // ptr = &b; illegal statement (assignment of read-only variable ptr)
    // *ptr = b; illegal statement (assignment of read-only location *ptr)

    return 0;
}
```

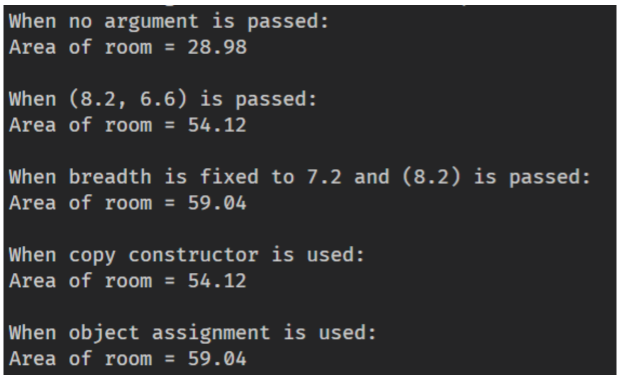


```
value pointed to by ptr: A
value pointed to by ptr: B

Value pointed to by ptr: A
Address ptr is pointing to: 6422023
Value pointed to by ptr: B
Address ptr is pointing to: 6422023

Value pointed to by ptr: A
Address ptr is pointing to: 6422023
```

Program - 31

Demonstrate the concept of constructor overloading. Also demonstrate copy constructor and object assignment concept.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Room
{
private:
    double length, breadth;

public:
    // 1. Constructor with no arguments
    Room()
    {
        length = 6.9;
        breadth = 4.2;
    }

    // 2. Constructor with two arguments
    Room(double l, double b)
    {
        length = l;
        breadth = b;
    }

    // 3. Constructor with one argument
    Room(double len)
    {
        length = len;
        breadth = 7.2;
    }

    // 4. Copy constructor
    Room(const Room &r)
    {
        this->length = r.length;
        this->breadth = r.breadth;
    }

    double calcArea()
    {
        return length * breadth;
    }
};

int main()
{
    Room room1, room2(8.2, 6.6), room3(8.2);

    // NO ARGUMENT
    cout << "When no argument is passed:" << endl;
    cout << "Area of room = " << room1.calcArea() << endl;
    // ==========================================================================
    // LENGTH AND BREADTH PASSED
    cout << "\nWhen (8.2, 6.6) is passed:" << endl;
    cout << "Area of room = " << room2.calcArea() << endl;

    // ==========================================================================
    // LENGTH PASSED, BREADTH FIXED
    cout << "\nWhen breadth is fixed to 7.2 and (8.2) is passed:" << endl;
    cout << "Area of room = " << room3.calcArea() << endl;

    // ==========================================================================
    Room room4(room2), room5 = room3;
    // COPY CONSTRUCTOR
    cout << "\nWhen copy constructor is used:" << endl;
    cout << "Area of room = " << room4.calcArea() << endl;

    // ==========================================================================
    // OBJECT ASSIGNMENT
    cout << "\nWhen object assignment is used:" << endl;
    cout << "Area of room = " << room5.calcArea() << endl;

    return 0;
}
```

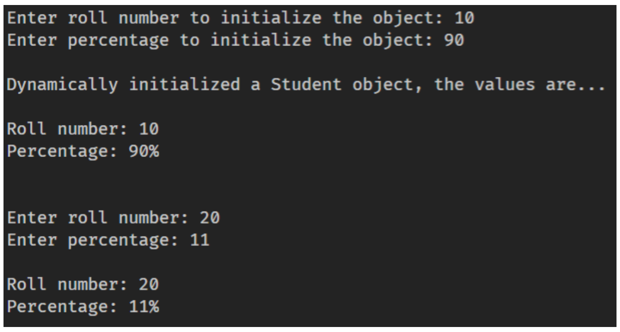


```
When no argument is passed:
Area of room = 28.98

When (8.2, 6.6) is passed:
Area of room = 54.12

When breadth is fixed to 7.2 and (8.2) is passed:
Area of room = 59.04

When copy constructor is used:
Area of room = 54.12

When object assignment is used:
Area of room = 59.04
```

Program - 32

Demonstrate dynamic initialisation of object.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

//structure definition with private and public members
class Student
{
private:
    int rNo;
    float perc;

public:
    //constructor
    Student(int r, float p)
    {
        rNo = r;
        perc = p;
    }

    //function to read details
    void read(void)
    {
        cout << "Enter roll number: ";
        cin >> rNo;
        cout << "Enter percentage: ";
        cin >> perc;
    }

    //function to print details
    void print(void)
    {
        cout << endl;
        cout << "Roll number: " << rNo << endl;
        cout << "Percentage: " << perc << "%" << endl;
    }
};

//Main code
int main()
{
    //reading roll number and percentage to initialize
    //the members while creating object
    cout << "Enter roll number to initialize the object: ";
    int roll_number;
    cin >> roll_number;
    cout << "Enter percentage to initialize the object: ";
    float percentage;
    cin >> percentage;

    // DYNAMIC INITIALISATION
    Student s1(roll_number, percentage);
    //print the value
    cout << "\nDynamically initialized a Student object, the values are..." << endl;
    s1.print();
    cout << "\n\n";

    //reading and printing student details
    //by calling public member functions of the structure
    s1.read();
    s1.print();

    return 0;
}
```

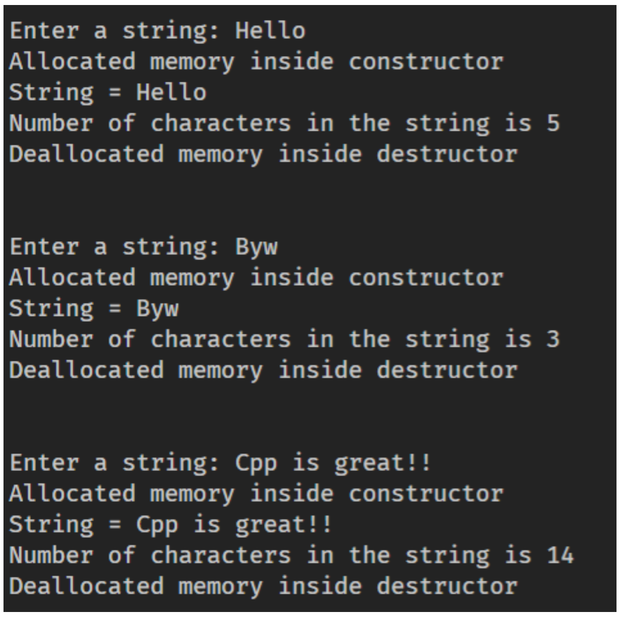


```
Enter roll number to initialize the object: 10
Enter percentage to initialize the object: 90

Dynamically initialized a Student object, the values are...

Roll number: 10
Percentage: 90%


Enter roll number: 20
Enter percentage: 11

Roll number: 20
Percentage: 11%
```

Program - 33

Demonstratedynamic constructor.

```
/**
* V. Sriram
* 10314902019
*/

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

class dyncons
{
    char *str;
    int length;

public:
    // a dynamic constructor is one which dynamically allocates memory
    // for a variable on object initialization
    dyncons()
    {
        length = 0;
        str = new char[length + 1];
        cout << "Allocated memory inside constructor\n";
    }

    dyncons(char *s)
    {
        length = strlen(s);
        str = new char[length + 1];
        strcpy(str, s);
        cout << "Allocated memory inside constructor\n";
    }

    void output()
    {
        cout << "String = " << str << "\n"
             << "Number of characters in the string is " << length << "\n";
    }

    // deallocate the memory to avoid memory leaks
    ~dyncons()
    {
        delete[] str;
        cout << "Deallocated memory inside destructor\n";
    }
};

int main()
{
    // create 3 objects of dyncons
    // which get destroyed after each loop
    for (int i = 0; i < 3; ++i)
    {
        cout << "\n\n";

        char s[50];
        cout << "Enter a string:\t";
        cin.getline(s, 50, '\n');
        dyncons obj(s);
        obj.output();
    }

    return 0;
}
```

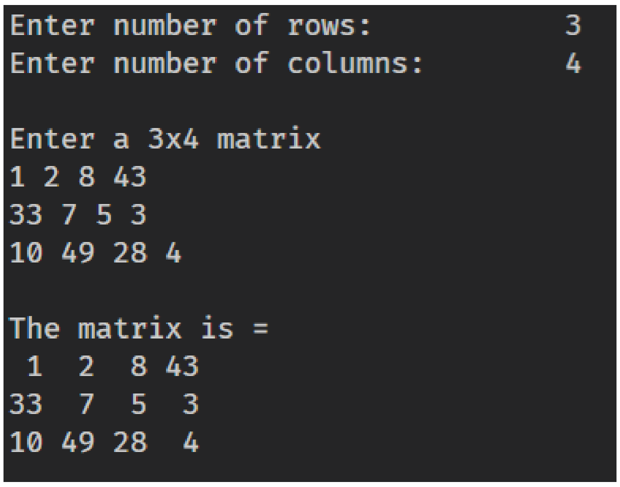


```
Enter a string: Hello
Allocated memory inside constructor
String = Hello
Number of characters in the string is 5
Deallocated memory inside destructor


Enter a string: Byw
Allocated memory inside constructor
String = Byw
Number of characters in the string is 3
Deallocated memory inside destructor


Enter a string: Cpp is great!!
Allocated memory inside constructor
String = Cpp is great!!
Number of characters in the string is 14
Deallocated memory inside destructor
```

Program - 34

WAP to create a class Matrix to represent a 2D array to be allocated the memory dynamically by using concept of array of array pointers. Write down a parametric constructor that receives no. of rows and no. of columnsas arguments. Write a destructor to release a memory. Write down two member function input() and output() for elements of the 2D Array.

```
/**
* V. Sriram
* 10314902019
*/

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

class Matrix
{
private:
    // a pointer to pointer
    int **arr;
    int row, col;

    void inputDimensions()
    {
        cout << "Enter number of rows:\t\t";
        cin >> row;
        cout << "Enter number of columns:\t";
        cin >> col;
        cout << "\n";
    }

public:
    Matrix()
    {
        inputDimensions();
        dynamicallyAllocateMatrix();
    }

    Matrix(int r, int c)
    {
        row = r;
        col = c;

        dynamicallyAllocateMatrix();
    }

    void dynamicallyAllocateMatrix()
    {
        // dynamically allocate an array of arrays
        arr = new int *[row];
        for (int i = 0; i < row; ++i)
            arr[i] = new int[col];
    }

    void inputValues()
    {
        cout << "Enter a " << row << "x" << col << " matrix\n";
        for (int i = 0; i < row; ++i)
            for (int j = 0; j < col; ++j)
                cin >> arr[i][j];
    }

    void output()
    {
        cout << "\nThe matrix is = \n";
        for (int i = 0; i < row; ++i)
        {
            for (int j = 0; j < col; ++j)
                cout << right << setw(2) << arr[i][j] << ' ';

            cout << "\n";
        }
    }

    ~Matrix()
    {
        // deallocate an array of arrays of size row/col
        for (int i = 0; i < col; ++i)
            delete[] arr[i];

        delete[] arr;
    }
};

int main()
{
    Matrix m1;
    m1.inputValues();
    m1.output();

    return 0;
}
```

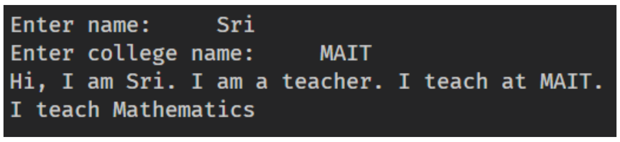


```
Enter number of rows:         3
Enter number of columns:      4

Enter a 3x4 matrix
1 2 8 43
33 7 5 3
10 49 28 4

The matrix is =
 1  2  8 43
33  7  5  3
10 49 28  4
```

Program - 35

Demonstrate single inheritance with visibility mode of base class as public.

```
/**
* V. Sriram
* 10314902019
*/

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

class Teacher
{
protected:
    char name[50];
    char collegeName[50];

public:
    void input()
    {
        cout << "Enter name:\t";
        cin >> name;
        cout << "Enter college name:\t";
        cin >> collegeName;
    }

    void sayHi()
    {
        cout << "Hi, I am " << name << ". ";
        cout << "I am a teacher. ";
        cout << "I teach at " << collegeName << ".\n";
    }
};

//This class inherits Teacher class with public vidibility mode
// MathTeacher inherits all members except public ones
// as its own public/protected members respectively
class MathTeacher : public Teacher
{
    char mainSub[30];

public:
    MathTeacher()
    {
        strcpy(mainSub, "Mathematics");
    }

    void sayHi()
    {
        Teacher::sayHi();
        cout << "I teach " << mainSub;
    }
};

int main()
{
    MathTeacher obj;

    cout << "\n\n";
    obj.input(); // member of Teacher class can be called outside class definition
    obj.sayHi();
    cout << "\n\n";

    return 0;
}
```



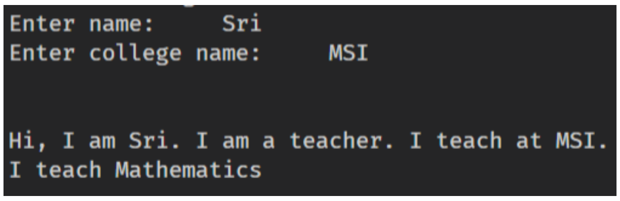
```
Enter name:     Sri
Enter college name:     MAIT
Hi, I am Sri. I am a teacher. I teach at MAIT.
I teach Mathematics
```

Program - 36

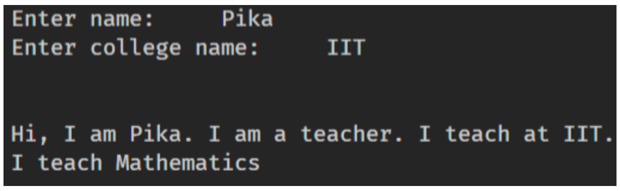
Demonstrate single inheritance with Visibility modeof base class as Private.

```
/**
* V. Sriram
* 10314902019
*/

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

class Teacher
{
protected:
    char name[50];
    char collegeName[50];

public:
    void input()
    {
        cout << "Enter name:\t";
        cin >> name;
        cout << "Enter college name:\t";
        cin >> collegeName;
    }

    void sayHi()
    {
        cout << "Hi, I am " << name << ". ";
        cout << "I am a teacher. ";
        cout << "I teach at " << collegeName << ".\n";
    }
};

//This class inherits Teacher class with private vidibility mode
// MathTeacher inherits all members except private ones
// as its own private members
class MathTeacher : private Teacher
{
    char mainSub[30];

public:
    MathTeacher()
    {
        strcpy(mainSub, "Mathematics");
        input();
    }

    void sayHi()
    {
        Teacher::sayHi();
        cout << "I teach " << mainSub;
    }
};

int main()
{
    MathTeacher obj;
```

```cpp
    cout << "\n\n";
    // obj.input(); cannot be called as its a private member
    obj.sayHi();
    cout << "\n\n";

    return 0;
}
```



```
Enter name:     Sri
Enter college name:     MSI


Hi, I am Sri. I am a teacher. I teach at MSI.
I teach Mathematics
```

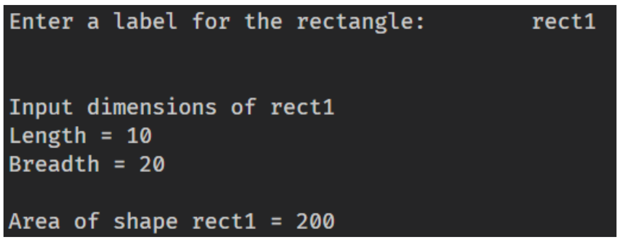


```
Enter name:     Pika
Enter college name:     IIT


Hi, I am Pika. I am a teacher. I teach at IIT.
I teach Mathematics
```

Program - 37

Demonstrate single inheritance(shape is base class and rectangleis derived class).

```
/**
* V. Sriram
* 10314902019
*/

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

class Shape // can be any shape
{
protected:
    char label[15];

public:
    Shape(char label[])
    {
        strcpy(this->label, label);
    }
};

// Rectangle inherits from Shape
class Rectangle : private Shape
{
    double length, breadth;

public:
    Rectangle(char label[], int l = 0, int b = 0) : Shape(label) // calling the
        Shape contructor
    {
        length = l;
        breadth = b;
    }

    void inputDimensions()
    {
        cout << "Input dimensions of " << label;
        cout << "\nLength = ";
        cin >> length;

        cout << "Breadth = ";
        cin >> breadth;
    }

    double calcArea()
    {
        return length * breadth;
    }

    char *getLabel()
    {
        return label;
    }
};

int main()
{
    char label[20];
    cout << "Enter a label for the rectangle:\t";
    cin >> label;
    cout << "\n\n";

    Rectangle r1(label);
    r1.inputDimensions();

    cout << "\nArea of shape " << r1.getLabel() << " = " << r1.calcArea();

    return 0;
}
```

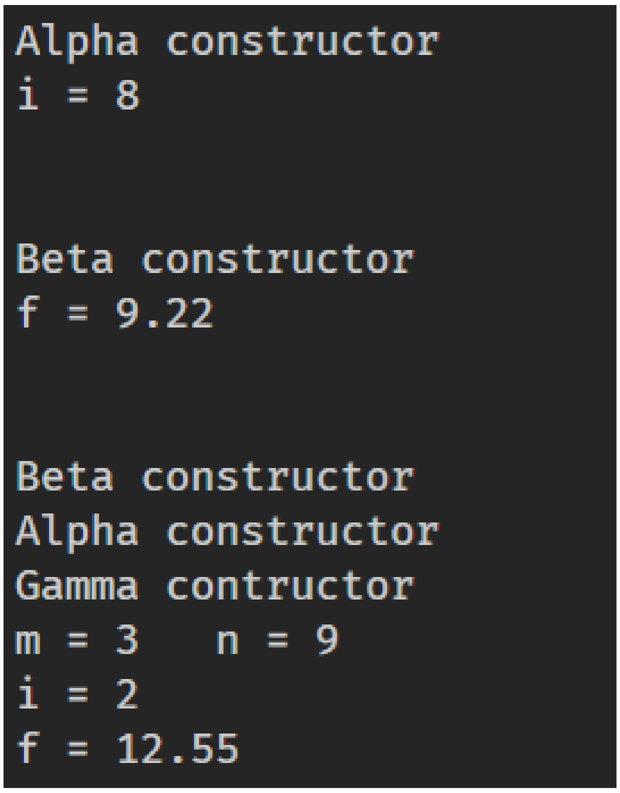


```
Enter a label for the rectangle:        rect1


Input dimensions of rect1
Length = 10
Breadth = 20

Area of shape rect1 = 200
```

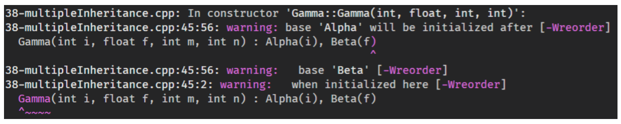
Program - 38

Demonstrate multiple inheritanceusing any 3 classes of your choice.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Alpha
{
protected:
    int i;

public:
    Alpha(int i)
    {
        this->i = i;
        cout << "Alpha constructor\n";
    }

    void show_alpha()
    {
        cout << "i = " << i << "\n";
    }
};

class Beta
{
protected:
    float f;

public:
    Beta(float f)
    {
        this->f = f;
        cout << "Beta constructor\n";
    }

    void show_beta()
    {
        cout << "f = " << f << "\n";
    }
};

// Gamma inherits both Beta and Alpha
class Gamma : public Beta, public Alpha
{
    int m, n;

public:
    // Beta will be initialized before Alpha
    // i.e., according to the sequence of inheritance
    Gamma(int i, float f, int m, int n) : Alpha(i), Beta(f)
    {
        this->m = m;
        this->n = n;
        cout << "Gamma contructor\n";
    }

    void show_gamma()
    {
        cout << "m = " << m << "\tn = " << n << "\n";
    }
};

int main()
{
    Alpha alpha(8);
    alpha.show_alpha();

    cout << "\n\n";

    Beta beta(9.22);
    beta.show_beta();

    cout << "\n\n";

    Gamma gamma(2, 12.55, 3, 9);

    // since Gamma inherits Alpha and Beta as public
    // all public members are inherited as public members too
    gamma.show_gamma();
    gamma.show_alpha();
    gamma.show_beta();

    return 0;
}
```



```
Alpha constructor
i = 8


Beta constructor
f = 9.22


Beta constructor
Alpha constructor
Gamma contructor
m = 3   n = 9
i = 2
f = 12.55
```



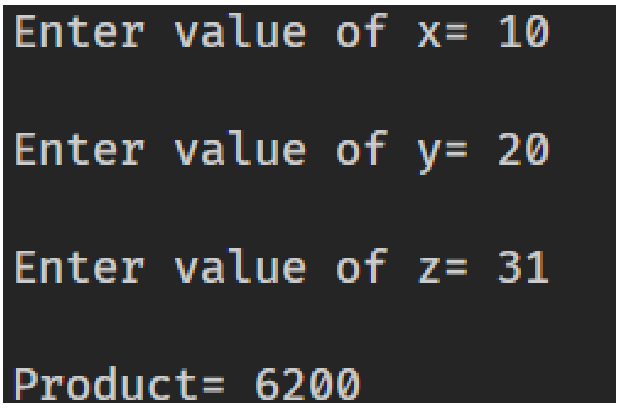
```
38-multipleInheritance.cpp: In constructor 'Gamma::Gamma(int, float, int, int)':
38-multipleInheritance.cpp:45:56: warning: base 'Alpha' will be initialized after [-Wreorder]
  Gamma(int i, float f, int m, int n) : Alpha(i), Beta(f)
                                                         ^
38-multipleInheritance.cpp:45:56: warning:   base 'Beta' [-Wreorder]
38-multipleInheritance.cpp:45:2: warning:   when initialized here [-Wreorder]
  Gamma(int i, float f, int m, int n) : Alpha(i), Beta(f)
  ^~~~~
```

Program - 39

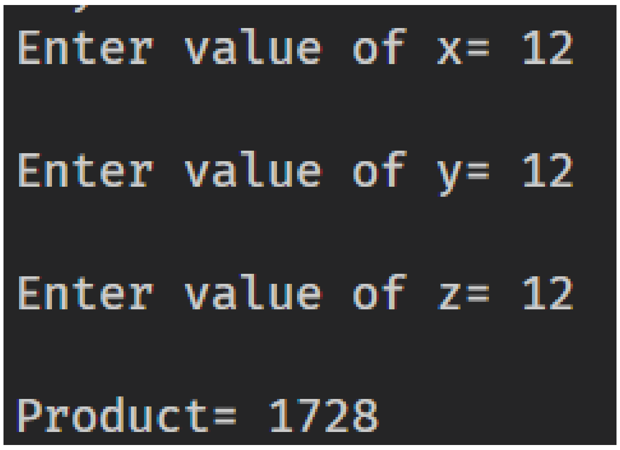
Demonstrate ambiguity resolution in multilevel inheritance(i.e. function in base class and derived class have same signature and you want to avoidoverriding ofbase class function by derived class function)

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class base //single base class
{
protected:
    int x;

public:
    void getdata()
    {
        cout << "Enter value of x= ";
        cin >> x;
    }
};

class derive1 : public base // derived class from base class
{
protected:
    int y;

public:
    void getdata() // get data is ambiguous with base's getdata
    {
        cout << "\nEnter value of y= ";
        cin >> y;
    }
};

class derive2 : public derive1 // derived from class derive1
{
private:
    int z;

public:
    void getdata() // get data is ambiguous with derive1's getdata
    {
        // to remove the ambiguity we use scope resolution operator
        derive1::getdata();

        cout << "\nEnter value of z= ";
        cin >> z;
    }
    void print()
    {
        cout << "\nProduct= " << x * y * z;
    }
};

int main()
```

```
{
    derive2 a; //object of derived class

    // to remove the ambiguity we use scope resolution operator
    a.base::getdata();
    a.getdata();
    a.print();

    return 0;
}
```



```
Enter value of x= 10

Enter value of y= 20

Enter value of z= 31

Product= 6200
```

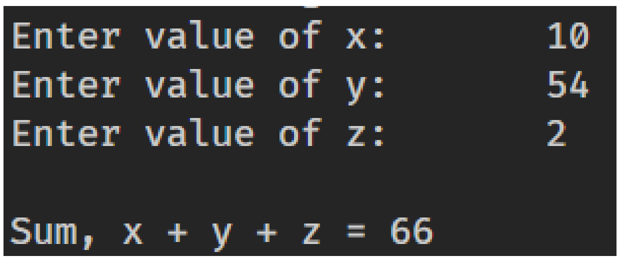


```
Enter value of x= 12

Enter value of y= 12

Enter value of z= 12

Product= 1728
```

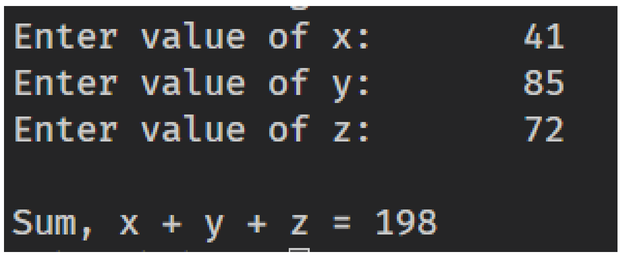
Program - 40

Demonstration of multiple inheritance without any ambiguity.(i.e. noneof the functions have same name in base and derived classes)

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class A
{
protected:
    int x;

public:
    void getx()
    {
        cout << "Enter value of x:\t";
        cin >> x;
    }
};

class B
{
protected:
    int y;

public:
    void gety()
    {
        cout << "Enter value of y:\t";
        cin >> y;
    }
};

// multiple inheritance
class C : public A, public B // C is derived from class A and class B
{
    int z;

public:
    void getz()
    {
        cout << "Enter value of z:\t";
        cin >> z;
    }
    void sum()
    {
        cout << "\nSum, x + y + z = " << x + y + z;
    }
};

int main()
{
    C obj; //object of derived class C

    obj.getx();
    obj.gety();
    obj.getz();
    obj.sum();

    return 0;
}
```



```
Enter value of x:       10
Enter value of y:       54
Enter value of z:       2

Sum, x + y + z = 66
```



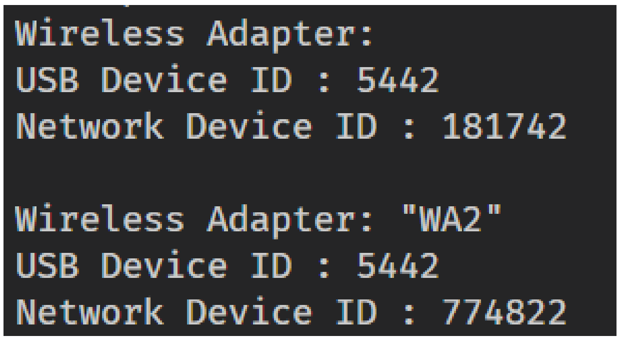
```
Enter value of x:       41
Enter value of y:       85
Enter value of z:       72

Sum, x + y + z = 198
```

Program - 41

Demonstration of multiple inheritance with ambiguity resolution.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class USBDevice
{
private:
    long m_id;

public:
    USBDevice(long id)
        : m_id(id)
    {
    }

    long getID() { return m_id; }
};

class NetworkDevice
{
private:
    long m_id;

public:
    NetworkDevice(long id)
        : m_id(id)
    {
    }

    long getID() { return m_id; }
};

// WirelessAdapter is derived from USBDevice and NetworkDevice
class WirelessAdapter : public USBDevice, public NetworkDevice
{
    const char *label;

public:
    WirelessAdapter(const char *label, long usbId, long networkId)
        : USBDevice(usbId), NetworkDevice(networkId), label(label)
    {
    }

    void print()
    {
        cout << "Wireless Adapter: \"" << label << "\"";
        cout << "\nUSB Device ID : " << USBDevice::getID();         // resolving
        ambiguity using scope resolution
        cout << "\nNetwork Device ID : " << NetworkDevice::getID(); // resolving
        ambiguity using scope resolution
    }
};
```

```
int main()
{
    WirelessAdapter wa1("WA1", 5442, 181742);
    // cout << wa1.getID(); // Which getID() do we call?

    cout << "Wireless Adapter:\n";
    cout << "USB Device ID : " << wa1.USBDevice::getID();          // resolving
        ambiguity using scope resolution
    cout << "\nNetwork Device ID : " << wa1.NetworkDevice::getID(); // resolving
        ambiguity using scope resolution

    WirelessAdapter wa2("WA2", 5442, 774822);
    cout << "\n\n";
    wa2.print();

    return 0;
}
```

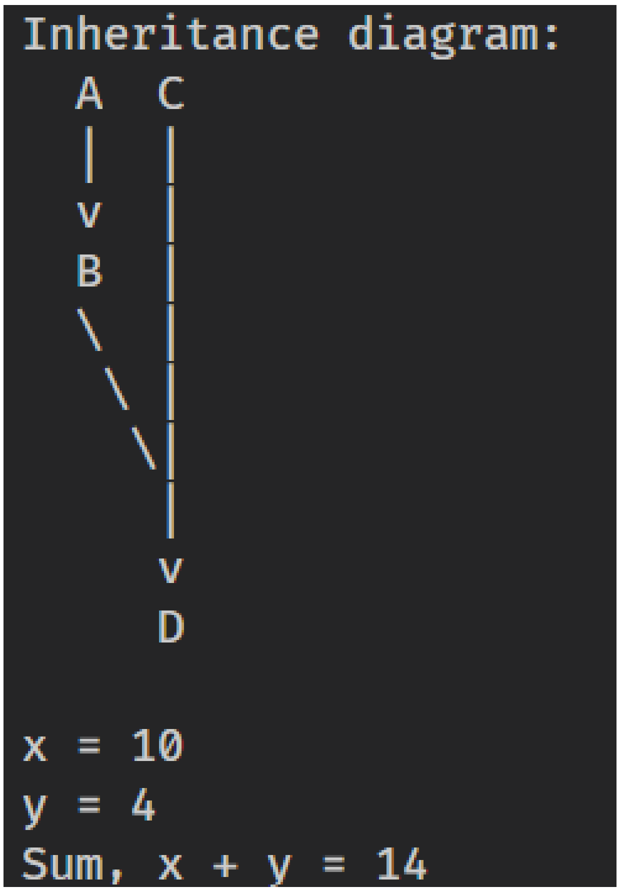


```
Wireless Adapter:
USB Device ID : 5442
Network Device ID : 181742

Wireless Adapter: "WA2"
USB Device ID : 5442
Network Device ID : 774822
```

Program - 42

Demonstration of hybrid inheritance.

```cpp
/**
* V. Sriram
* 10314902019
*/

#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 << "x = " << x;
        cout << "\ny = " << y;
        cout << "\nSum, x + y = " << x + y;
    }
};

int main()
{
    cout << "Inheritance diagram:\n";
    cout << "  A  C\n";
    cout << "  |  |\n";
    cout << "  v  |\n";
    cout << "  B  |\n";
    cout << "  \\  |\n";
    cout << "   \\ |\n";
    cout << "    \\|\n";
    cout << "     |\n";
    cout << "     v\n";
    cout << "     D\n\n";
    D obj1; //object of derived class D
    obj1.sum();

    return 0;
}
```

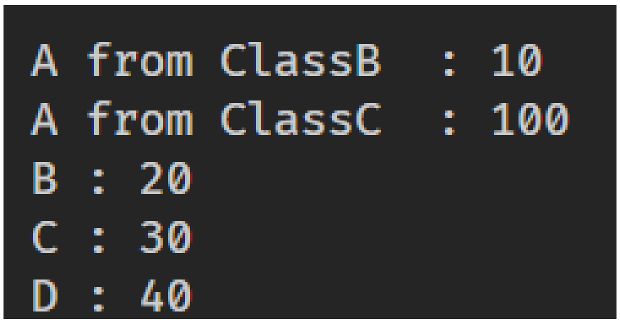


```
Inheritance diagram:
   A   C
   |   |
   v   |
   B   |
   \   |
    \  |
     \ |
       |
       v
       D

x = 10
y = 4
Sum, x + y = 14
```

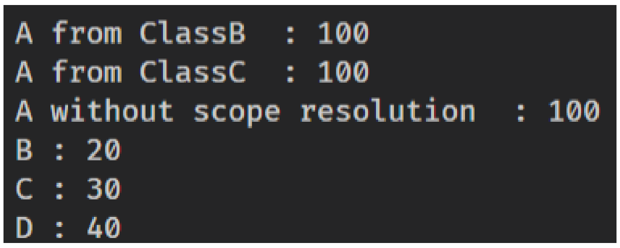
Program - 43

Demonstrate Multipath inheritance(with & without virtual base class).

```cpp
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class ClassA
{
public:
    int a;
};

class ClassB : virtual public ClassA
{
public:
    int b;
};
class ClassC : virtual public ClassA
{
public:
    int c;
};

class ClassD : public ClassB, public ClassC
{
public:
    int d;
};

int main()
{

    ClassD obj;

    /** without virtual base class,
     * there would be 2 copies of classA
     * thus, 2 copies of obj.a
    */

    // obj.a = 10;   // error without virtual class
    // obj.a = 100; // error without virtual class

    obj.ClassB::a = 10;  // resolve ambiguity when not using virtual scope resolution
    obj.ClassC::a = 100; // resolve ambiguity when not using virtual scope resolution

    obj.b = 20;
    obj.c = 30;
    obj.d = 40;

    cout << "\n A from ClassB  : " << obj.ClassB::a;
    cout << "\n A from ClassC  : " << obj.ClassC::a;
    cout << "\n A without scope resolution  : " << obj.a;

    cout << "\n B : " << obj.b;
    cout << "\n C : " << obj.c;
    cout << "\n D : " << obj.d;

    return 0;
}
```



```
A from ClassB  : 10
A from ClassC  : 100
B : 20
C : 30
D : 40
```

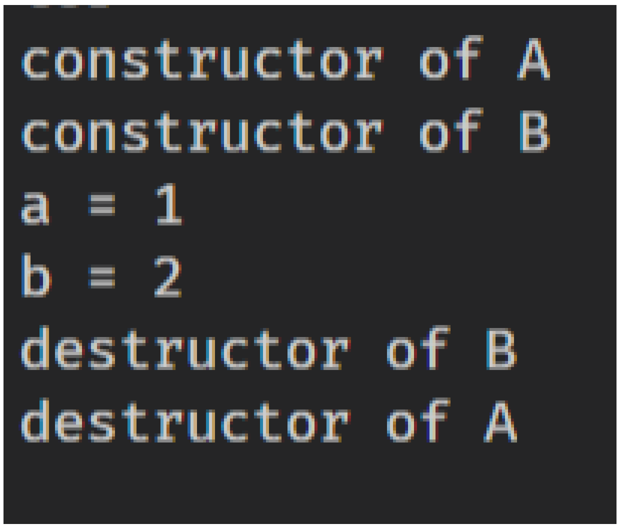


```
A from ClassB  : 100
A from ClassC  : 100
A without scope resolution  : 100
B : 20
C : 30
D : 40
```

Program - 44

Demonstrate the concept of constructors in the derived classes for single inheritance(use parameterised constructor).

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class A
{
protected:
    int a;

public:
    A(int i) : a(i)
    {
        cout << "constructor of A\n";
    }

    ~A()
    {
        cout << "destructor of A\n";
    }
};

class B : public A
{
protected:
    int b;

public:
    B(int ia, int ib) : A(ia)
    {
        b = ib;
        cout << "constructor of B\n";
    }

    void show()
    {
        cout << "a = " << a << "\n";
        cout << "b = " << b << "\n";
    }

    ~B()
    {
        cout << "destructor of B\n";
    }
};

int main()
{
    B objB(1, 2);
    objB.show();

    return 0;
}
```

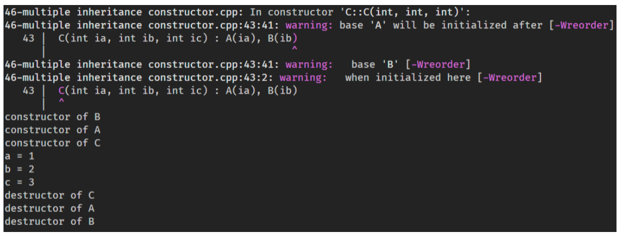

```
constructor of A
constructor of B
a = 1
b = 2
destructor of B
destructor of A
```

Program - 45

Demonstrate the concept of constructors in the derived class for multiple inheritance(use parameterised constructor).

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class A
{
protected:
    int a;

public:
    A(int i) : a(i)
    {
        cout << "constructor of A\n";
    }

    ~A()
    {
        cout << "destructor of A\n";
    }
};

class B
{
protected:
    int b;

public:
    B(int i) : b(i)
    {
        cout << "constructor of B\n";
    }

    ~B()
    {
        cout << "destructor of B\n";
    }
};

class C : public B, public A
{
    int c;

public:
    C(int ia, int ib, int ic) : A(ia), B(ib)
    {
        c = ic;
        cout << "constructor of C\n";
    }

    void show()
    {
        cout << "a = " << a << "\n";
```

```
            cout << "b = " << b << "\n";
            cout << "c = " << c << "\n";
        }

        ~C()
        {
            cout << "destructor of C\n";
        }
};

int main()
{
    C objC(1, 2, 3);
    objC.show();

    return 0;
}
```

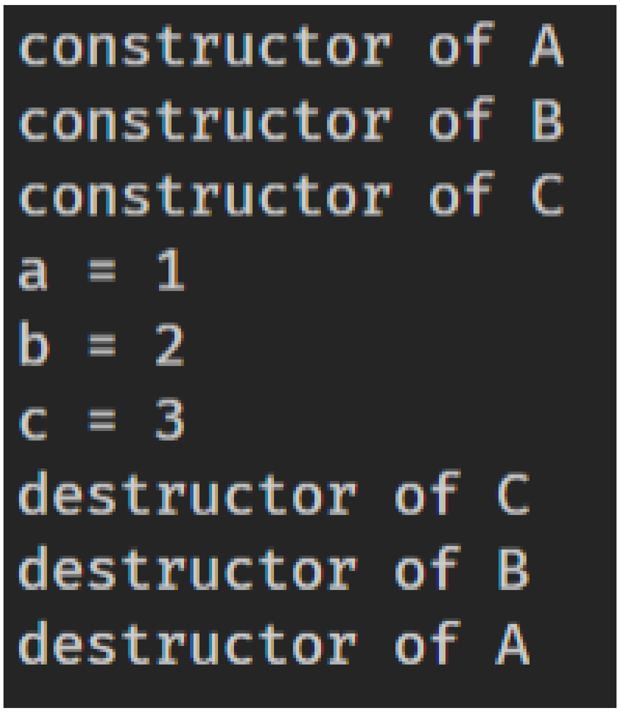


```
46-multiple inheritance constructor.cpp: In constructor 'C::C(int, int, int)':
46-multiple inheritance constructor.cpp:43:41: warning: base 'A' will be initialized after [-Wreorder]
   43 |   C(int ia, int ib, int ic) : A(ia), B(ib)
      |                                          ^
46-multiple inheritance constructor.cpp:43:41: warning:   base 'B' [-Wreorder]
46-multiple inheritance constructor.cpp:43:2: warning:   when initialized here [-Wreorder]
   43 |   C(int ia, int ib, int ic) : A(ia), B(ib)
      |   ^
constructor of B
constructor of A
constructor of C
a = 1
b = 2
c = 3
destructor of C
destructor of A
destructor of B
```

Program - 46

Demonstrate the concept of constructors in the derived class for multilevel inheritance(use parameterised constructor).

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class A
{
protected:
    int a;

public:
    A(int i) : a(i)
    {
        cout << "constructor of A\n";
    }

    ~A()
    {
        cout << "destructor of A\n";
    }
};

class B : public A
{
protected:
    int b;

public:
    B(int ia, int ib) : A(ia)
    {
        b = ib;
        cout << "constructor of B\n";
    }

    ~B()
    {
        cout << "destructor of B\n";
    }
};

class C : public B
{
    int c;

public:
    C(int ia, int ib, int ic) : B(ia, ib)
    {
        c = ic;
        cout << "constructor of C\n";
    }

    void show()
    {
        cout << "a = " << a << "\n";
        cout << "b = " << b << "\n";
        cout << "c = " << c << "\n";
    }

    ~C()
    {
        cout << "destructor of C\n";
    }
};

int main()
{
    C objC(1, 2, 3);
    objC.show();

    return 0;
}
```

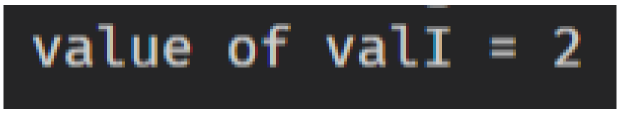


```
constructor of A
constructor of B
constructor of C
a = 1
b = 2
c = 3
destructor of C
destructor of B
destructor of A
```

Program - 47

Demonstrate the concept of nested class.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Outer
{
public:
    class Inner
    {
        int valI;

    public:
        Inner(int x = 0)
        {
            valI = x;
        }
        void show()
        {
            cout << "Value of valI = " << valI << endl;
        }
    };
};

int main()
{
    Outer::Inner objI(2);
    objI.show();

    return 0;
}
```

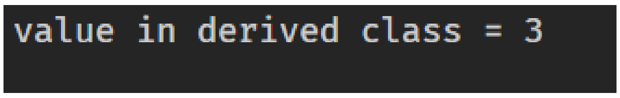


```
Value of valI = 2
```

Program - 48

Using Virtual functionto access private data of class.

```cpp
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Base
{
protected:
    int x;

public:
    Base(int x = 0) : x(x) {}

    virtual void show()
    {
        cout << "value in base class = " << x << endl;
    }
};

class Derived : public Base
{
    int y;
    void show()
    {
        cout << "value in derived class = " << y << endl;
    }

public:
    Derived(int y = 0) : y(y) {}
};

int main()
{
    Base *ptr;
    Derived obj(3);

    ptr = &obj;
    // ptr can access the private member function show()
    // of Derived object
    ptr->show();

    return 0;
}
```

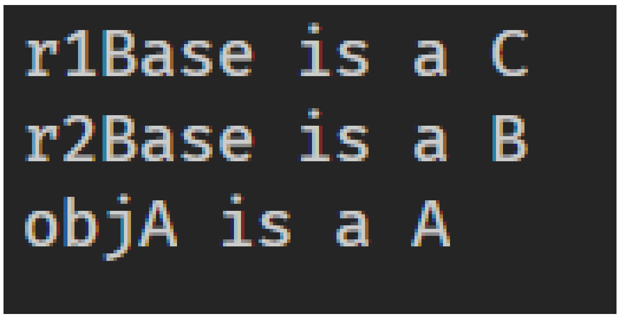


```
value in derived class = 3
```

Program - 49

Runtime polymorphism in case of multilevel inheritance.

```cpp
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class A
{
public:
    virtual char getName() const { return 'A'; }
};

class B : public A
{
public:
    virtual char getName() const { return 'B'; }
};

class C : public B
{
public:
    virtual char getName() const { return 'C'; }
};

int main()
{
    C c;
    B b;

    A &r1Base = c;
    cout << "r1Base is a " << r1Base.getName() << '\n';

    A &r2Base = b;
    cout << "r2Base is a " << r2Base.getName() << '\n';

    A objA;
    cout << "objA is a " << objA.getName() << '\n';

    return 0;
}
```

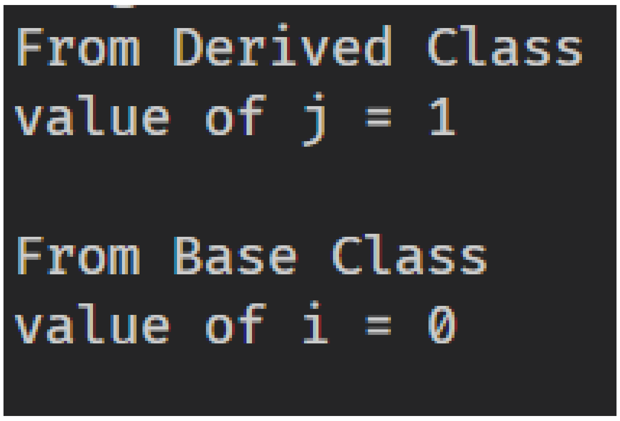


```
r1Base is a C
r2Base is a B
objA is a A
```

Program - 50

Explicit casting to enable a derived class pointer to point to base class object.

```
/**
* V. Sriram
* 10314902019
*/

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

class base
{
    int i;

public:
    base(int r = 0) : i(r) {}
    virtual void show()
    {
        cout << "From Base Class"
             << "\nvalue of i = " << i << endl;
    }
};

class derived : public base
{
    int j;

public:
    derived(int s = 1) : j(s) {}

    void show()
    {
        cout << "From Derived Class"
             << "\nvalue of j = " << j << endl;
    }
};
int main()
{
    base objBase;
    base *ptrBase;

    derived *ptrDerived;
    derived objDerived;

    // Explicit casting to enable a derived class pointer to point to base class
        object
    ptrDerived = (derived *)&objBase;
    ptrBase = &objDerived;

    ptrBase->show();
    cout << "\n";
    ptrDerived->show();
    return 0;
}
```

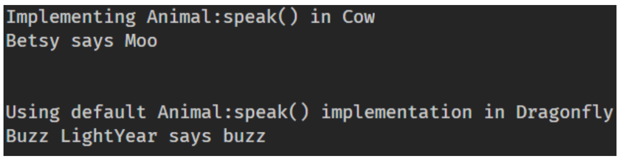


```
From Derived Class
value of j = 1

From Base Class
value of i = 0
```

Program - 51

Demonstrate the concept of abstract base class.

```
/**
* V. Sriram
* 10314902019
*/

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

class Animal // This Animal is an abstract base class
{
protected:
    const char *name;

public:
    Animal(const char *n) : name(n) {}

    const char *getName() const
    {
        return name;
    }
    virtual const char *speak() const = 0; // speak is now a pure virtual function
};

const char *Animal::speak() const // even though it has a body
{
    return "buzz";
}

class Cow : public Animal
{
public:
    Cow(const char *name) : Animal(name) {}

    // implementing absract base class's
    // pure virtual function
    const char *speak() const
    {
        return "Moo";
    }
};

class Dragonfly : public Animal
{

public:
    Dragonfly(const char *name) : Animal(name) {}

    const char *speak() const // this class is no longer abstract because we defined
        this function
    {
        return Animal::speak(); // use Animal's default implementation
    }
};

int main()
{
    Cow cow("Betsy");
    cout << "Implementing Animal:speak() in Cow\n";
    cout << cow.getName() << " says " << cow.speak() << '\n';

    Dragonfly dragonfly("Buzz LightYear");
    cout << "\n\nUsing default Animal:speak() implementation in Dragonfly\n";
    cout << dragonfly.getName() << " says " << dragonfly.speak() << "\n";

    return 0;
}
```



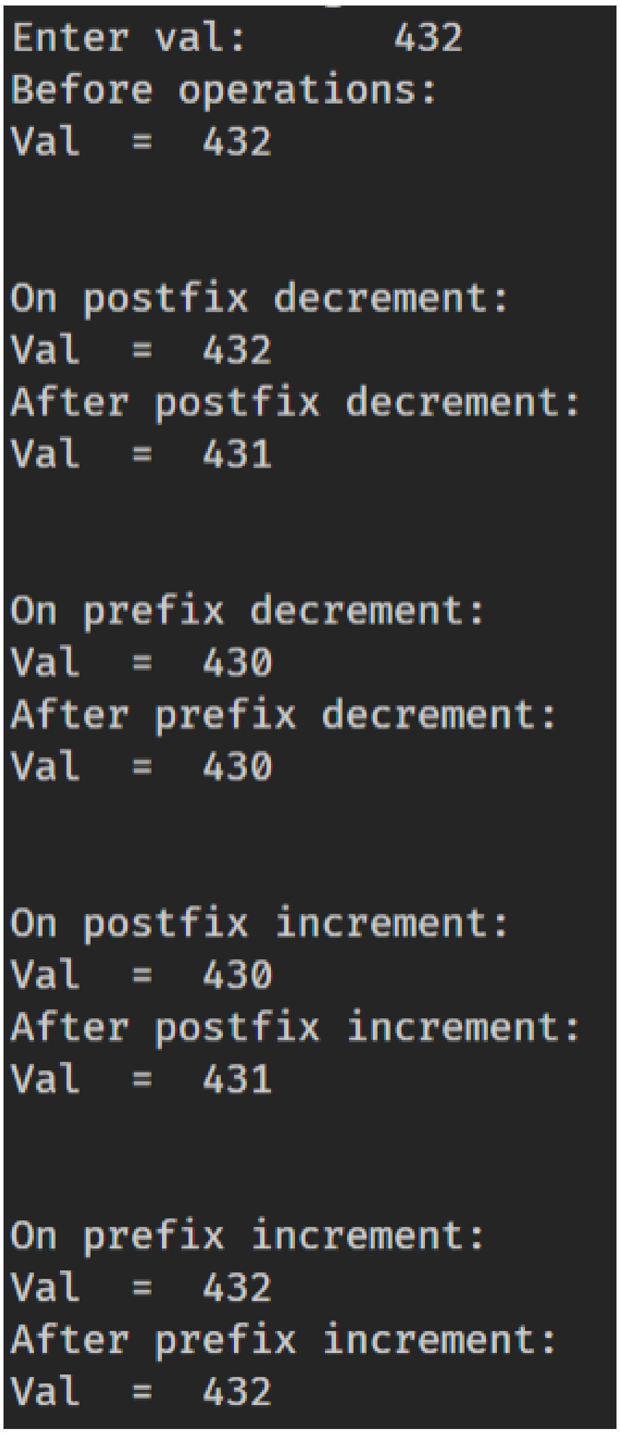
```
Implementing Animal:speak() in Cow
Betsy says Moo


Using default Animal:speak() implementation in Dragonfly
Buzz LightYear says buzz
```

Program - 52

WAP to overload prefix and postfix versions of increment and decrement operators for index class(++,–).

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Index
{
private:
    int val;

public:
    // convert int -> Index
    Index(int v = 0) : val(v) {}

    void input()
    {
        cout << "Enter val:\t";
        cin >> val;
    }

    void display()
    {
        cout << "Val  =  " << val << "\n";
    }

    // overloading prefix decrement operator
    friend Index operator--(Index &k)
    {
        return Index(--k.val);
    }

    // overloading postfix decrement operator
    friend Index operator--(Index &k, int)
    {
        return Index(k.val--);
    }

    // overloading prefix increment operator
    friend Index operator++(Index &k)
    {
        return Index(++k.val);
    }

    // overloading postfix increment operator
    friend Index operator++(Index &k, int)
    {
        return Index(k.val++);
    }
};

int main()
{
    Index i1;
```

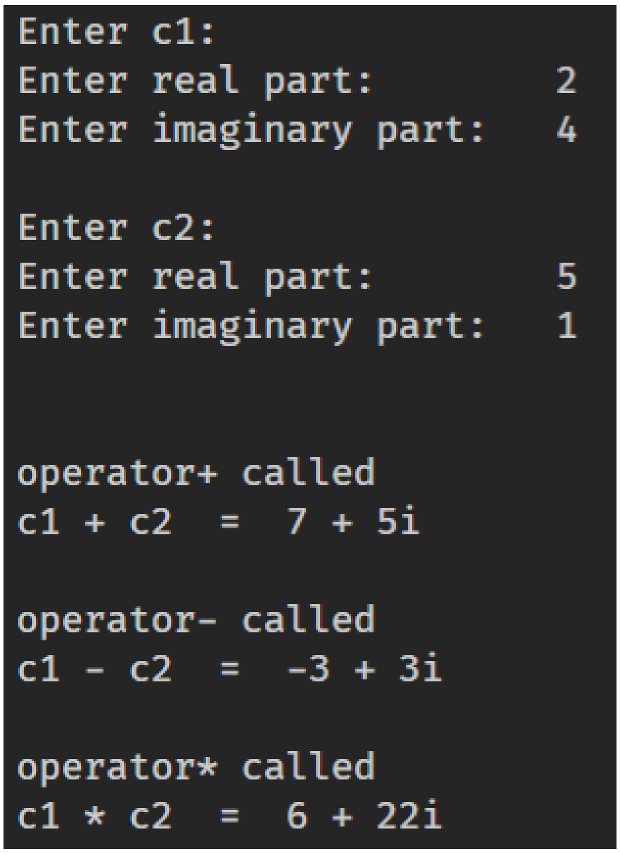
```cpp
    i1.input();

    cout << "Before operations:\n";
    i1.display();

    cout << "\n\n";

    // =============================================================
    cout << "On postfix decrement:\n";
    (i1--).display(); // using postfix decrement opertor
    cout << "After postfix decrement:\n";
    i1.display();

    cout << "\n\n";

    // =============================================================
    cout << "On prefix decrement:\n";
    (--i1).display(); // using prefix decrement opertor
    cout << "After prefix decrement:\n";
    i1.display();

    cout << "\n\n";

    // =============================================================
    cout << "On postfix increment:\n";
    (i1++).display(); // using postfix increment opertor
    cout << "After postfix increment:\n";
    i1.display();

    cout << "\n\n";

    // =============================================================
    cout << "On prefix increment:\n";
    (++i1).display(); // using prefix increment opertor
    cout << "After prefix increment:\n";
    i1.display();

    return 0;
}
```


```
Enter val:      432
Before operations:
Val  =  432


On postfix decrement:
Val  =  432
After postfix decrement:
Val  =  431


On prefix decrement:
Val  =  430
After prefix decrement:
Val  =  430


On postfix increment:
Val  =  430
After postfix increment:
Val  =  431


On prefix increment:
Val  =  432
After prefix increment:
Val  =  432
```

Program - 53

WAP to overload +,-,* for complex number class.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Complex
{
    int real, imag;

public:
    Complex(int r = 0, int i = 0) : real(r), imag(i) {}

    void show()
    {
        cout << real << " + " << imag << "i" << endl;
    }

    void input()
    {
        cout << "Enter real part:\t";
        cin >> real;

        cout << "Enter imaginary part:\t";
        cin >> imag;
    }

    // overloading the binary plus operator
    Complex operator+(Complex c2)
    {
        cout << "operator+ called\n";
        return Complex(real + c2.real, imag + c2.imag);
    }

    // overloading the binary minus operator
    Complex operator-(Complex c2)
    {
        cout << "operator- called\n";
        return Complex(real - c2.real, imag - c2.imag);
    }

    // overloading the binary multiply operator
    Complex operator*(Complex c2)
    {
        cout << "operator* called\n";
        return Complex(
            real * c2.real - imag * c2.imag,
            real * c2.imag + imag * c2.real);
    }
};

int main()
{
    Complex c1, c2;
    cout << "Enter c1:\n";
    c1.input();

    cout << "\nEnter c2:\n";
    c2.input();

    cout << "\n\n";

    Complex c3 = c1 + c2;
    cout << "c1 + c2  =  ";
    c3.show();

    cout << "\n";

    Complex c4 = c1 - c2;
    cout << "c1 - c2  =  ";
    c4.show();

    cout << "\n";

    Complex c5 = c1 * c2;
    cout << "c1 * c2  =  ";
    c5.show();

    return 0;
}
```

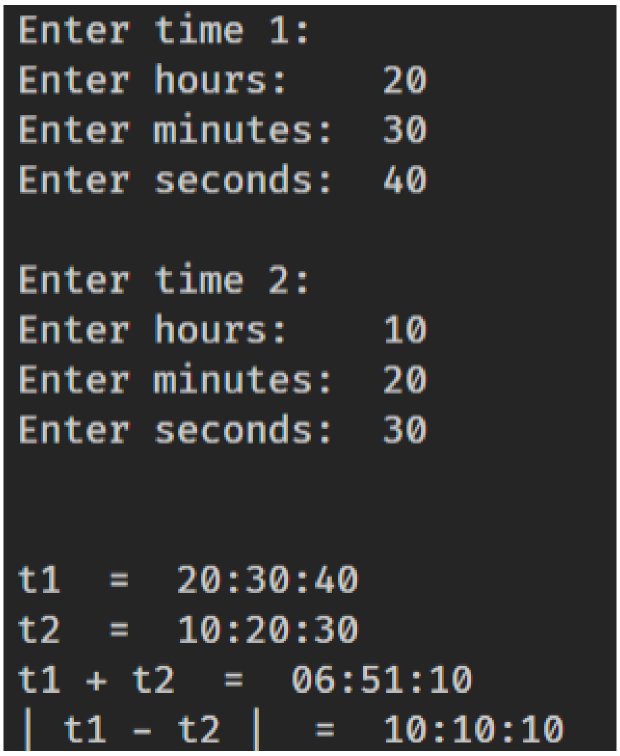


```
Enter c1:
Enter real part:        2
Enter imaginary part:   4

Enter c2:
Enter real part:        5
Enter imaginary part:   1


operator+ called
c1 + c2  =  7 + 5i

operator- called
c1 - c2  =  -3 + 3i

operator* called
c1 * c2  =  6 + 22i
```

Program - 54

WAP to overload +,-for time class.

```cpp
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Time
{
private:
    int h, m;
    long long s;

public:
    Time(int hours = 0, int minutes = 0, long seconds = 0)
    {
        if (hours < 0 || minutes < 0 || seconds < 0)
            throw "Invalid time. Values can't be negative";

        h = hours;
        m = minutes;
        s = seconds;
        normlize();
    }

    // returns total time converted to seconds
    long long getTotalSeconds() const
    {
        return s + m * 60 + h * 60 * 60;
    }

    // convert overflowed values to standard
    // seconds <= 60
    // minutes <= 60
    // hours <= 24
    void normlize()
    {
        if (s >= 60)
        {
            m += s / 60;
            s %= 60;
        }
        if (m >= 60)
        {
            h += m / 60;
            m %= 60;
        }
        if (h >= 24)
        {
            h %= 24;
        }
    }

    friend ostream &operator<<(ostream &out, Time &t);
    friend istream &operator>>(istream &in, Time &t);
    // overload + and - operators
    Time operator+(const Time &t2);
    Time operator-(const Time &t2);
};

Time Time::operator+(const Time &t2)
{
    // add time in seconds and normalize to correct format
    long long totalSeconds = this->getTotalSeconds() + t2.getTotalSeconds();
    return Time(0, 0, totalSeconds);
}

Time Time::operator-(const Time &t2)
{
    // subtract time in seconds and return absolute value
    // as -ve time doesn't make sense
    long long totalSeconds = this->getTotalSeconds() - t2.getTotalSeconds();
    return Time(0, 0, totalSeconds < 0 ? -totalSeconds : totalSeconds);
}

// get time from input stream
istream &operator>>(istream &in, Time &t)
{
    cout << "Enter hours:\t";
    in >> t.h;

    cout << "Enter minutes:\t";
    in >> t.m;

    cout << "Enter seconds:\t";
    in >> t.s;

    t.normlize();

    return in;
}

// display in format hh:mm:ss
ostream &operator<<(ostream &out, Time &t)
{
    printf("%02d:%02d:%02lld", t.h, t.m, t.s);
    return out;
}

int main()
{
    Time t1, t2; // objects of type Time

    cout << "Enter time 1:\n";
    cin >> t1;

    cout << "\nEnter time 2:\n";
    cin >> t2;

    cout << "\n\nt1  =  " << t1;
    cout << "\nt2  =  " << t2 << "\n";

    // add time obejcts
    Time t3 = t1 + t2;
    cout << "t1 + t2  =  " << t3;

    // subtract time objects
    Time t4 = t1 - t2;
    cout << "\n| t1 - t2 | =  " << t4;
    return 0;
}
```

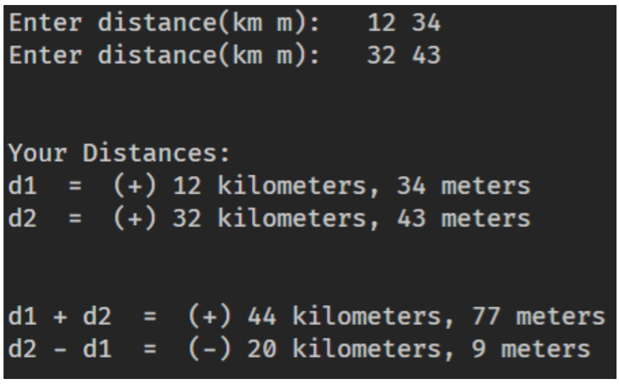


```
Enter time 1:
Enter hours:    20
Enter minutes:  30
Enter seconds:  40

Enter time 2:
Enter hours:    10
Enter minutes:  20
Enter seconds:  30


t1  =  20:30:40
t2  =  10:20:30
t1 + t2  =  06:51:10
| t1 - t2 |  =  10:10:10
```

Program - 55

WAP to overload +,-for distance class.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Distance
{
private:
    int km, m;
    int totalMeters;
    // isNegative: track if distance is < 0
    bool isNegative;

public:
    Distance(int kilometer = 0, int meter = 0)
    {
        km = kilometer;
        m = meter;

        normalize();
    }

    // convert overflowed values to standard
    // meters <= 1000
    void normalize()
    {
        totalMeters = km * 1000 + m;

        if (totalMeters < 0)
        {
            isNegative = true;
            km = -totalMeters / 1000;
            m = -totalMeters % 1000;
        }
        else
        {
            isNegative = false;
            km = totalMeters / 1000;
            m = totalMeters % 1000;
        }
    }

    void input()
    {
        cout << "Enter distance(km m):\t";
        cin >> km >> m;
        normalize();
    }

    // display in format km kilometers, m meters
    void display()
    {
        if (isNegative)
            cout << "(-) " << km << " kilometers, " << m << " meters\n";
        else
            cout << "(+) " << km << " kilometers, " << m << " meters\n";
    }

    // convert distance to meters
    int getTotalMeters() const
    {
        return totalMeters;
    }

    // get sum of 2 distance objects
    Distance operator+(const Distance &d) const
    {
        int totalMeters = this->getTotalMeters() + d.getTotalMeters();
        return Distance(0, totalMeters);
    }

    // get difference of 2 distance objects
    Distance operator-(const Distance &d) const
    {
        int totalMeters = this->getTotalMeters() - d.getTotalMeters();
        return Distance(0, totalMeters);
    }
};

int main()
{
    Distance d1, d2;

    d1.input();
    d2.input();

    cout << "\n\nYour Distances:\n";
    cout << "d1  =  ";
    d1.display();
    cout << "d2  =  ";
    d2.display();

    Distance d3 = d1 + d2;
    cout << "\n\nd1 + d2  =  ";
    d3.display();

    Distance d4 = d1 - d2;
    cout << "d2 - d1  =  ";
    d4.display();

    return 0;
}
```

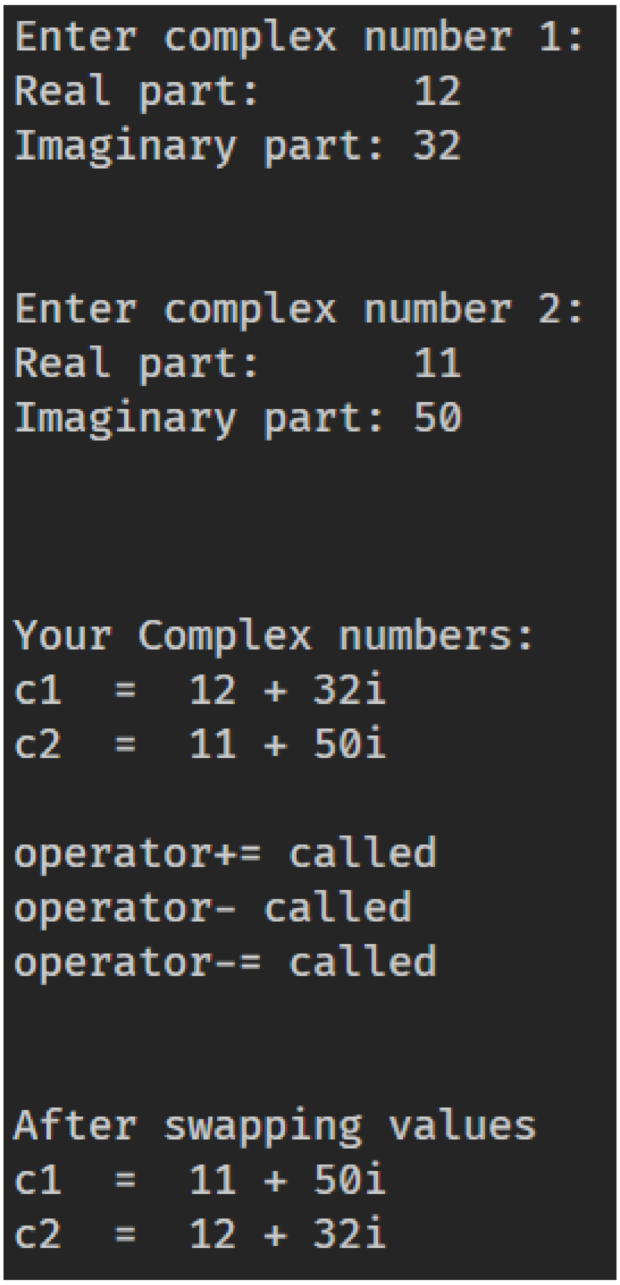


```
Enter distance(km m):   12 34
Enter distance(km m):   32 43


Your Distances:
d1  =  (+) 12 kilometers, 34 meters
d2  =  (+) 32 kilometers, 43 meters


d1 + d2  =  (+) 44 kilometers, 77 meters
d2 - d1  =  (-) 20 kilometers, 9 meters
```

Program - 56

WAP to overload short hand operators +=,-= for complex number class.

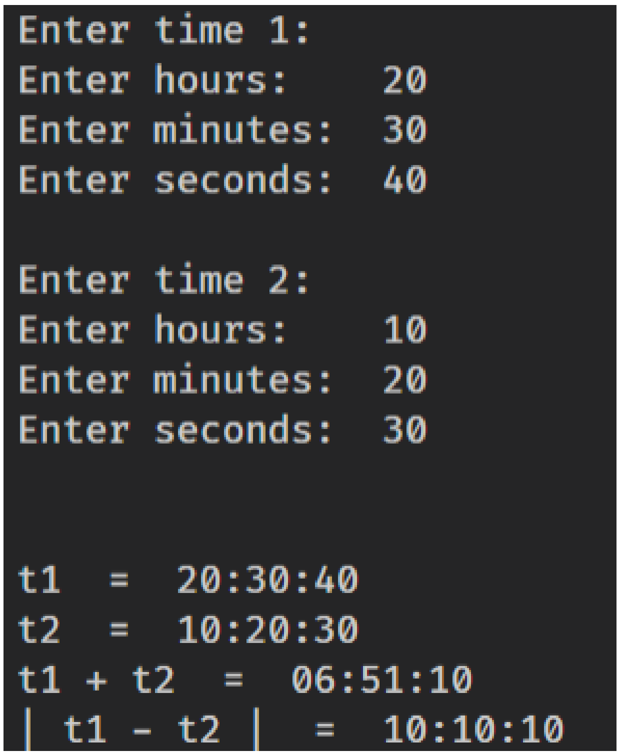
```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Complex
{
private:
    int real, imag;

public:
    Complex(int r = 0, int i = 0)
    {
        real = r;
        imag = i;
    }

    void input()
    {
        cout << "Real part:\t";
        cin >> real;

        cout << "Imaginary part:\t";
        cin >> imag;
        cout << '\n';
    }

    // outputs in format
    // 2 + 32i
    void output()
    {
        cout << real;
        if (imag < 0)
            cout << " - " << -imag;
        else
            cout << " + " << imag;

        cout << "i\n";
    }

    // overload += operator
    void operator+=(const Complex &c2)
    {
        cout << "operator+= called\n";
        real += c2.real;
        imag += c2.imag;
    }

    // overload -= operator
    void operator-=(const Complex &c2)
    {
        cout << "operator-= called\n";
        real -= c2.real;
        imag -= c2.imag;
    }

    Complex operator-(const Complex &c2)
    {
        cout << "operator- called\n";
        return Complex(real - c2.real, imag - c2.imag);
    }
};

// swap 2 obejects of Complex type
void swap(Complex &c1, Complex &c2)
{
    c1 += c2;
    c2 = c1 - c2;
    c1 -= c2;
}

int main()
{
    Complex c1, c2;

    cout << "Enter complex number 1:\n";
    c1.input();
    cout << "\nEnter complex number 2:\n";
    c2.input();

    cout << "\n\nYour Complex numbers:\n";
    cout << "c1  =  ";
    c1.output();
    cout << "c2  =  ";
    c2.output();

    cout << "\n";
    // swap values
    // this function called operator+=, operator-
    // operator-= internally
    swap(c1, c2);

    cout << "\n\nAfter swapping values\n";
    cout << "c1  =  ";
    c1.output();
    cout << "c2  =  ";
    c2.output();

    return 0;
}
```


```
Enter complex number 1:
Real part:      12
Imaginary part: 32


Enter complex number 2:
Real part:      11
Imaginary part: 50



Your Complex numbers:
c1  =  12 + 32i
c2  =  11 + 50i

operator+= called
operator- called
operator-= called


After swapping values
c1  =  11 + 50i
c2  =  12 + 32i
```

Program - 57

WAP to overload +,-using friend function for time class.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Time
{
private:
    int h, m;
    long long s;

public:
    Time(int hours = 0, int minutes = 0, long seconds = 0)
    {
        if (hours < 0 || minutes < 0 || seconds < 0)
            throw "Invalid time. Values can't be negative";

        h = hours;
        m = minutes;
        s = seconds;
        normlize();
    }

    // returns total time converted to seconds
    long long getTotalSeconds() const
    {
        return s + m * 60 + h * 60 * 60;
    }

    // convert overflowed values to standard
    // seconds <= 60
    // minutes <= 60
    // hours <= 24
    void normlize()
    {
        if (s >= 60)
        {
            m += s / 60;
            s %= 60;
        }
        if (m >= 60)
        {
            h += m / 60;
            m %= 60;
        }
        if (h >= 24)
        {
            h %= 24;
        }
    }

    friend ostream &operator<<(ostream &out, Time &t);
    friend istream &operator>>(istream &in, Time &t);
    // overload + and - operators
    friend Time operator+(const Time &t1, const Time &t2);
    friend Time operator-(const Time &t1, const Time &t2);
};

Time operator+(const Time &t1, const Time &t2)
{
    // add time in seconds and normalize to correct format
    long long totalSeconds = t1.getTotalSeconds() + t2.getTotalSeconds();
    return Time(0, 0, totalSeconds);
}

Time operator-(const Time &t1, const Time &t2)
{
    // subtract time in seconds and return absolute value
    // as -ve time doesn't make sense
    long long totalSeconds = t1.getTotalSeconds() - t2.getTotalSeconds();
    return Time(0, 0, totalSeconds < 0 ? -totalSeconds : totalSeconds);
}

// get time from input stream
istream &operator>>(istream &in, Time &t)
{
    cout << "Enter hours:\t";
    in >> t.h;

    cout << "Enter minutes:\t";
    in >> t.m;

    cout << "Enter seconds:\t";
    in >> t.s;

    t.normlize();

    return in;
}

// display in format hh:mm:ss
ostream &operator<<(ostream &out, Time &t)
{
    printf("%02d:%02d:%02lld", t.h, t.m, t.s);
    return out;
}

int main()
{
    Time t1, t2; // objects of type Time

    cout << "Enter time 1:\n";
    cin >> t1;

    cout << "\nEnter time 2:\n";
    cin >> t2;

    cout << "\n\nt1  =  " << t1;
    cout << "\nt2  =  " << t2 << "\n";

    // add time obejcts
    Time t3 = t1 + t2;
    cout << "t1 + t2  =  " << t3;

    // subtract time objects
    Time t4 = t1 - t2;
    cout << "\n| t1 - t2 | =  " << t4;
    return 0;
}
```

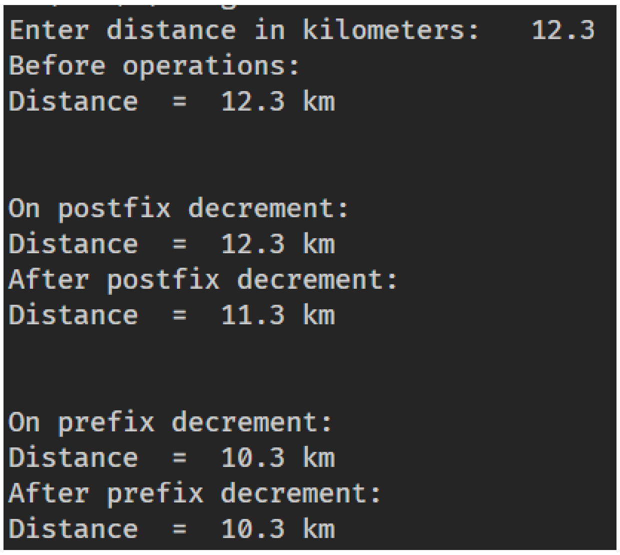


```
Enter time 1:
Enter hours:     20
Enter minutes:   30
Enter seconds:   40

Enter time 2:
Enter hours:     10
Enter minutes:   20
Enter seconds:   30


t1   =   20:30:40
t2   =   10:20:30
t1 + t2   =   06:51:10
| t1 - t2 |   =   10:10:10
```

Program - 58

WAP to overload prefix and postfix decrement operator for location class using friend function.

```cpp
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Kilometer
{
private:
    double km;

public:
    // convert int -> Kilometer
    Kilometer(double kilometer = 0) : km{kilometer} {}

    void input()
    {
        cout << "Enter distance in kilometers:\t";
        cin >> km;
    }

    void display()
    {
        cout << "Distance  =  " << km << " km\n";
    }

    // overloading prefix decrement operator
    friend Kilometer operator--(Kilometer &k)
    {
        return Kilometer(--k.km);
    }

    // overloading postfix decrement operator
    friend Kilometer operator--(Kilometer &k, int)
    {
        return Kilometer(k.km--);
    }
};

int main()
{
    Kilometer k;
    k.input();

    cout << "Before operations:\n";
    k.display();

    cout << "\n\n";

    cout << "On postfix decrement:\n";
    (k--).display(); // using postfix decrement opertor
    cout << "After postfix decrement:\n";
    k.display();

    cout << "\n\n";
    cout << "On prefix decrement:\n";
    (--k).display(); // using prefix decrement opertor
    cout << "After prefix decrement:\n";
    k.display();

    return 0;
}
```

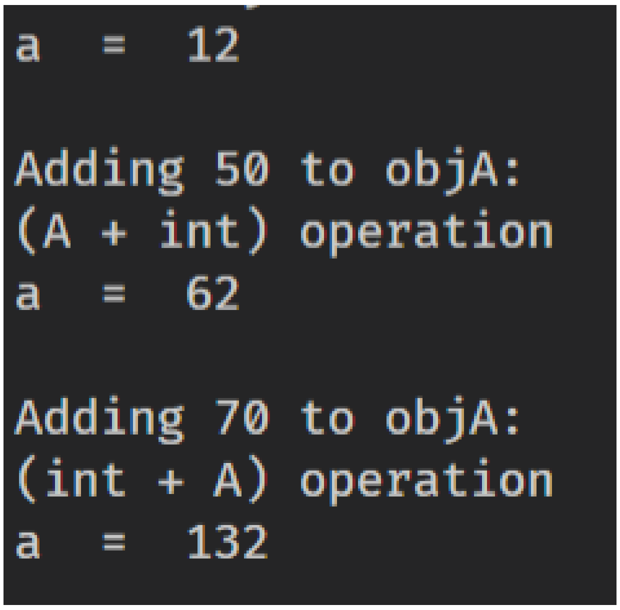


```
Enter distance in kilometers:   12.3
Before operations:
Distance  =  12.3 km


On postfix decrement:
Distance  =  12.3 km
After postfix decrement:
Distance  =  11.3 km


On prefix decrement:
Distance  =  10.3 km
After prefix decrement:
Distance  =  10.3 km
```

Program - 59

Friend function for operator overloading to add flexibility.(e.g. int+C1)

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class A
{
    int a;

public:
    A(int a = 0) : a(a) {}

    void show()
    {
        cout << "a  =  " << a << "\n";
    }

    // overloding + operator to add integer
    friend A operator+(int i, const A &objA);
    friend A operator+(const A &objA, int i);
};

A operator+(int i, const A &objA)
{
    cout << "(int + A) operation\n";
    return A(i + objA.a);
}

A operator+(const A &objA, int i)
{
    cout << "(A + int) operation\n";
    return A(objA.a + i);
}

int main()
{
    A objA(12);
    objA.show();

    cout << "\nAdding 50 to objA:\n";
    objA = objA + 50;
    objA.show();

    cout << "\nAdding 70 to objA:\n";
    objA = 70 + objA;
    objA.show();

    return 0;
}
```

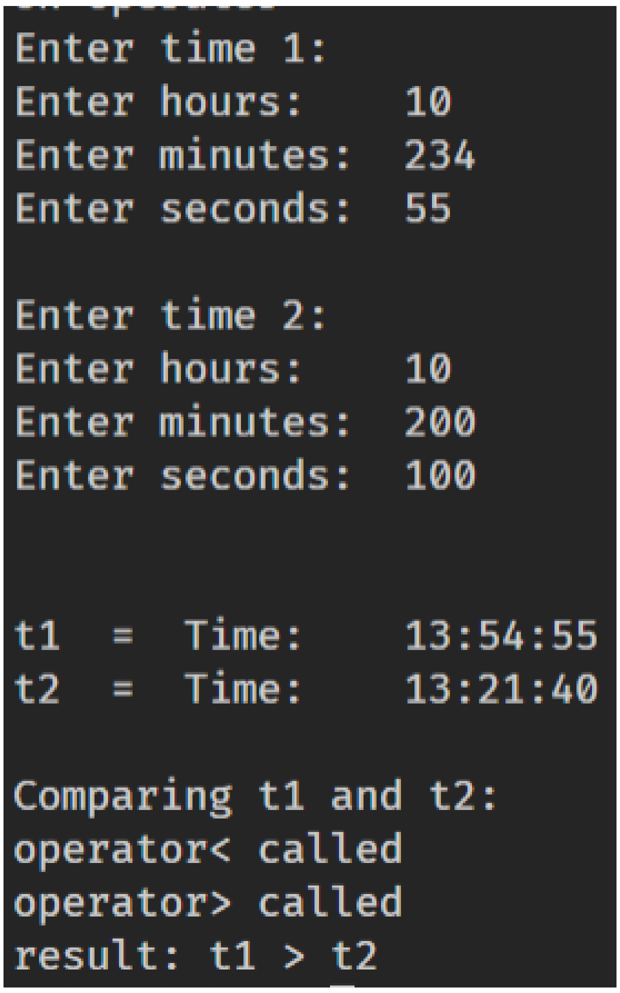


```
a  =  12

Adding 50 to objA:
(A + int) operation
a  =  62

Adding 70 to objA:
(int + A) operation
a  =  132
```

Program - 60

Overloading of comparison operators (< , >)

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Time
{
private:
    int h, m, s;

public:
    Time(int hours = 0, int minutes = 0, int seconds = 0)
    {
        if (hours < 0 || minutes < 0 || seconds < 0)
            throw "Invalid time. Values can't be negative";

        h = hours;
        m = minutes;
        s = seconds;
        normlize();
    }

    // convert overflowed values to standard
    // seconds <= 60
    // minutes <= 60
    // hours <= 24
    void normlize()
    {
        if (s >= 60)
        {
            m += s / 60;
            s %= 60;
        }
        if (m >= 60)
        {
            h += m / 60;
            m %= 60;
        }
        if (h >= 24)
        {
            h %= 24;
        }
    }

    friend ostream &operator<<(ostream &out, Time &t);
    friend istream &operator>>(istream &in, Time &t);

    // overloading comparison operators
    bool operator<(const Time &t)
    {
        cout << "operator< called\n";

        if (h < t.h)
            return true;
        else if (h > t.h)
            return false;
        else if (m < t.m)
            return true;
        else if (m > t.m)
            return false;
        else if (s < t.s)
            return true;
        else
            return false;
    }

    // overloading comparison operators
    bool operator>(const Time &t)
    {
        cout << "operator> called\n";

        if (h > t.h)
            return true;
        else if (h < t.h)
            return false;
        else if (m > t.m)
            return true;
        else if (m < t.m)
            return false;
        else if (s > t.s)
            return true;
        else
            return false;
    }
};

// get time from input stream
istream &operator>>(istream &in, Time &t)
{
    cout << "Enter hours:\t";
    in >> t.h;

    cout << "Enter minutes:\t";
    in >> t.m;

    cout << "Enter seconds:\t";
    in >> t.s;

    t.normlize();

    return in;
}

// display in format hh:mm:ss
ostream &operator<<(ostream &out, Time &t)
{
    printf("Time:\t%02d:%02d:%02d", t.h, t.m, t.s);
    return out;
}

int main()
{
    Time t1, t2;

    cout << "Enter time 1:\n";
    cin >> t1;
    cout << "\nEnter time 2:\n";
    cin >> t2;

    cout << "\n\nt1  =  " << t1;
    cout << "\nt2  =  " << t2 << "\n";

    cout << "\nComparing t1 and t2:\n";
    if (t1 < t2) // using opertor<
        cout << "result:\tt1 < t2";
    else if (t1 > t2) // using opertor>
        cout << "result:\tt1 > t2";
    else
        cout << "result:\tt1 == t2";

    return 0;
}
```

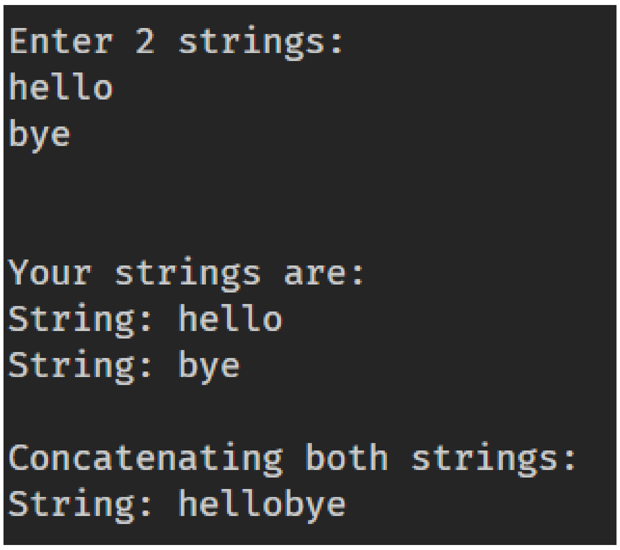


```
Enter time 1:
Enter hours:     10
Enter minutes:   234
Enter seconds:   55

Enter time 2:
Enter hours:     10
Enter minutes:   200
Enter seconds:   100


t1   =   Time:     13:54:55
t2   =   Time:     13:21:40

Comparing t1 and t2:
operator< called
operator> called
result: t1 > t2
```

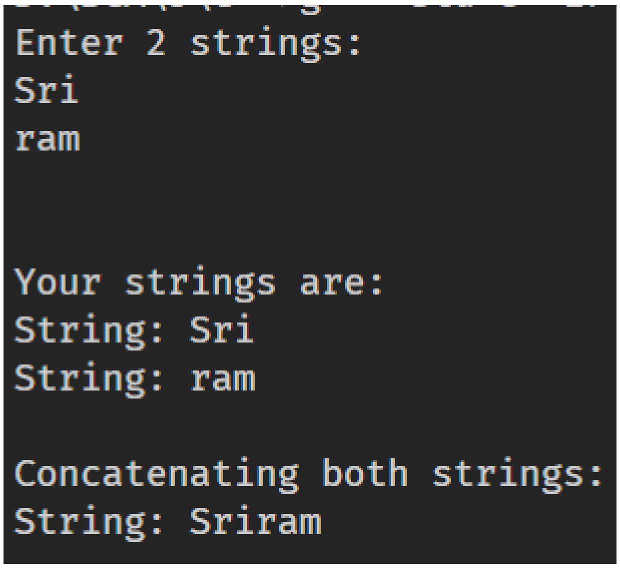
Program - 61

Overloading of + for string concatenation.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
#include <string.h>
using namespace std;

class String
{
    char *str;
    int len;

public:
    String(char *s) : str(s), len(strlen(s)) {}

    void display() const
    {
        cout << "String:\t" << str << "\n";
    }

    // overloading + operator
    // which concats the strings
    // and stores it in the first operand
    String operator+(String s2)
    {
        strcat(str, s2.str);
        len = strlen(str);
        return *this;
    }
};

int main()
{
    char t1[50], t2[50];
    cout << "Enter 2 strings:\n";

    cin >> t1 >> t2;

    String s1(t1), s2(t2);
    cout << "\n\nYour strings are:\n";
    s1.display();
    s2.display();

    cout << "\nConcatenating both strings:\n";
    // using overloaded + operator to concat strings
    s1 = s1 + s2;
    s1.display();

    return 0;
}
```



```
Enter 2 strings:
hello
bye


Your strings are:
String: hello
String: bye

Concatenating both strings:
String: hellobye
```

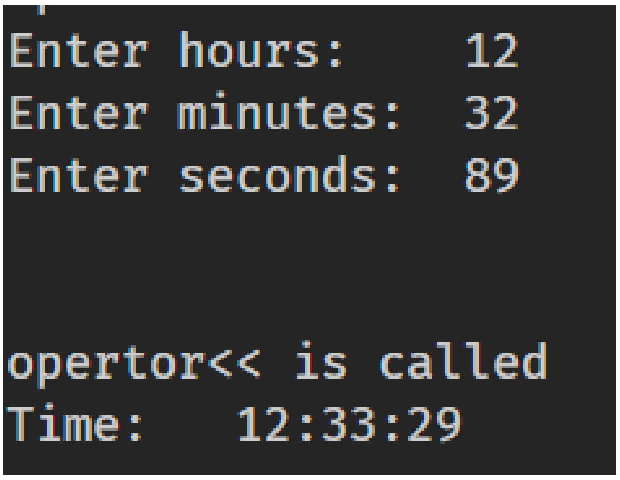


```
Enter 2 strings:
Sri
ram


Your strings are:
String: Sri
String: ram

Concatenating both strings:
String: Sriram
```

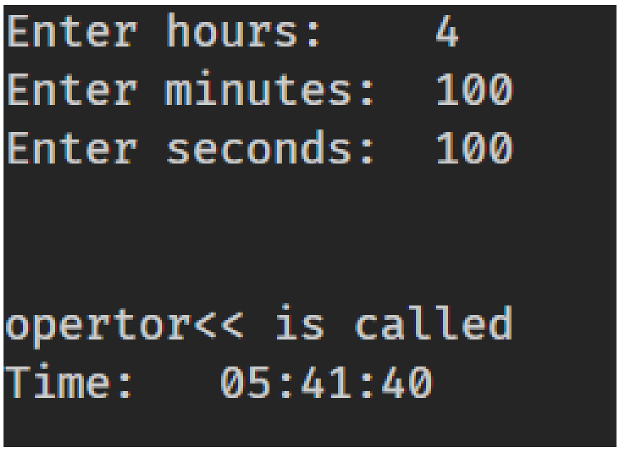
Program - 62

Overloading of << and >> operators.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Time
{
private:
    int h, m, s;

public:
    Time(int hours = 0, int minutes = 0, int seconds = 0)
    {
        if (hours < 0 || minutes < 0 || seconds < 0)
            throw "Invalid time. Values can't be negative";

        h = hours;
        m = minutes;
        s = seconds;
        normlize();
    }

    // convert overflowed values to standard
    // seconds <= 60
    // minutes <= 60
    // hours <= 24
    void normlize()
    {
        if (s >= 60)
        {
            m += s / 60;
            s %= 60;
        }
        if (m >= 60)
        {
            h += m / 60;
            m %= 60;
        }
        if (h >= 24)
        {
            h %= 24;
        }
    }

    friend ostream &operator<<(ostream &out, Time &t);
    friend istream &operator>>(istream &in, Time &t);
};

// get time from input stream
istream &operator>>(istream &in, Time &t)
{
    cout << "operator>> is called\n";

    cout << "Enter hours:\t";
    in >> t.h;

    cout << "Enter minutes:\t";
    in >> t.m;

    cout << "Enter seconds:\t";
    in >> t.s;

    t.normlize();

    return in;
}

// display in format hh:mm:ss
ostream &operator<<(ostream &out, Time &t)
{
    out << "opertor<< is called\n";
    printf("Time:\t%02d:%02d:%02d\n", t.h, t.m, t.s);
    return out;
}

int main()
{
    Time t1;
    cin >> t1;

    cout << "\n\n";

    cout << t1;

    return 0;
}
```



```
Enter hours:    12
Enter minutes:  32
Enter seconds:  89


opertor<< is called
Time:   12:33:29
```

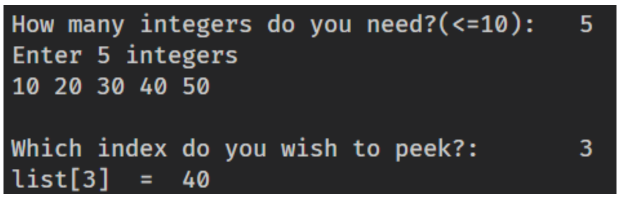


```
Enter hours:    4
Enter minutes:  100
Enter seconds:  100


opertor<< is called
Time:   05:41:40
```

Program - 63

Operator [] overloading.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class IntList
{
    int list[10];

public:
    IntList() : list({0, 0, 0, 0, 0, 0, 0, 0, 0, 0}) {}

    // overloading [] operator
    // reference is returned so that assignment is possible
    // at an index
    int &operator[](int index)
    {
        if (index < 0 || index >= 10)
        {
            cout << "Out of range index\n";
        }

        return list[index];
    }
};

int main()
{
    IntList il;

    int len;
    cout << "How many integers do you need?(<=10):\t";
    cin >> len;

    cout << "Enter " << len << " integers\n";
    for (int i = 0; i < len; ++i)
        cin >> il[i]; // using the overloaded [] operator to assign a value at index

    int index;
    cout << "\nWhich index do you wish to peek?:\t";
    cin >> index;
    cout << "list[" << index << "]  =  " << il[index]; // using the overloaded []
        operator to get the value at index

    return 0;
}
```

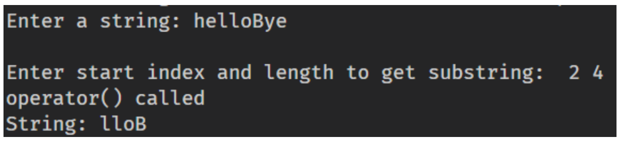


```
How many integers do you need?(<=10):   5
Enter 5 integers
10 20 30 40 50

Which index do you wish to peek?:       3
list[3]  =  40
```

Program - 64

Overloading of function call operator().

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
#include <string.h>
using namespace std;

class MyString
{
    const char *s;
    const int length;

public:
    MyString(const char *s) : s(s), length(strlen(s)) {}

    // overloading parentesis
    // to get substring of "s" member
    MyString operator()(int start, int len)
    {
        cout << "operator() called\n";

        // index cannot start before 0
        // index can't go beyond the length of string
        if (start < 0 || start + len > length)
        {
            cout << "Invalid range\n";
            return MyString("");
        }

        // store the substring in a temp string
        // return a MyString object using nameless objects concept
        char temp[length];
        for (int i = 0; i < len; ++i)
        {
            temp[i] = s[start + i];
        }
        temp[len] = '\0';

        return MyString(temp);
    }

    void display()
    {
        cout << "String:\t" << s;
    }
};

int main()
{
    char temp[100];
    cout << "Enter a string:\t";
    cin >> temp;

    int start, len;
    MyString str(temp);
    cout << "\nEnter start index and length to get substring:\t";
    cin >> start >> len;

    // using overloaded ()
    MyString substr = str(start, len);
    substr.display();

    return 0;
}
```

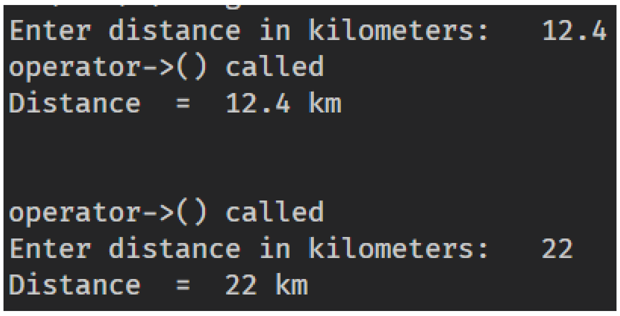


```
Enter a string: helloBye

Enter start index and length to get substring:  2 4
operator() called
String: lloB
```

Program - 65

Overloading of Arrow operator.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Kilometer
{
private:
    double km;

public:
    Kilometer(double kilometer = 0)
        : km{kilometer}
    {
    }

    void input()
    {
        cout << "Enter distance in kilometers:\t";
        cin >> km;
    }

    void display() const
    {
        cout << "Distance  =  " << km << " km\n";
    }

    // overload -> operator
    Kilometer *operator->()
    {
        cout << "operator->() called\n";
        return this;
    }
};

int main()
{
    Kilometer k;
    k.input();
    // since -> is overloaded
    // -> can be used similar to . operator
    // to access members
    k->display();

    cout << "\n\n";

    k->input();
    k.display();

    return 0;
}
```

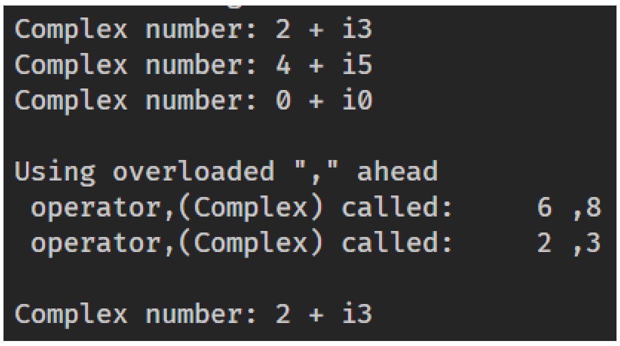


```
Enter distance in kilometers:   12.4
operator->() called
Distance  =  12.4 km


operator->() called
Enter distance in kilometers:   22
Distance  =  22 km
```

Program - 66

Overloading of , operator.

```cpp
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Complex
{
    int real, imag;

public:
    Complex(int r = 0, int i = 0) : real(r), imag(i) {}

    void show()
    {
        cout << "Complex number:\t" << real << " + i" << imag << endl;
    }

    // overloading the binary plus operator
    Complex operator+(Complex c2)
    {
        return Complex(real + c2.real, imag + c2.imag);
    }

    // overloading the comma operator
    Complex operator,(Complex c2)
    {
        Complex temp(c2.real, c2.imag);
        cout << " operator,(Complex) called:\t" << temp.real << " ," << temp.imag <<
        endl;

        return temp;
    }
};

int main()
{
    Complex obj1(2, 3), obj2(4, 5), obj3;

    obj1.show();
    obj2.show();
    obj3.show();

    cout << "\nUsing overloaded \",\" ahead\n";
    obj3 = (obj1, obj1 + obj2, obj1);

    cout << "\n";
    obj3.show();

    return 0;
}
```

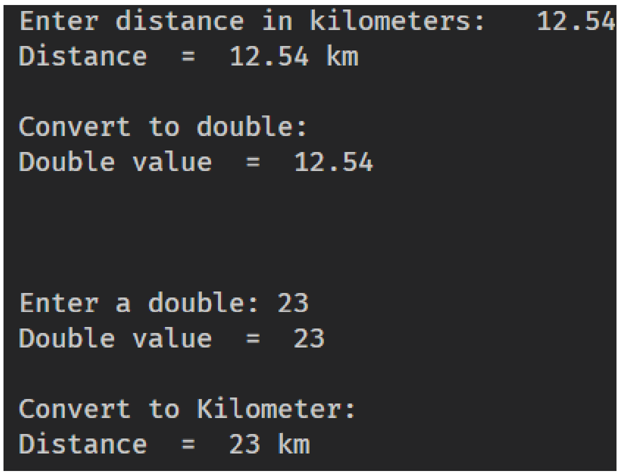


```
Complex number: 2 + i3
Complex number: 4 + i5
Complex number: 0 + i0

Using overloaded "," ahead
 operator,(Complex) called:     6 ,8
 operator,(Complex) called:     2 ,3

Complex number: 2 + i3
```

Program - 67

Demonstrate Data Conversion between basic and user-defined data types(both ways conversion)

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

class Kilometer
{
private:
    double km;

public:
    // convert int -> Kilometer
    Kilometer(double kilometer = 0) : km{kilometer} {}

    // Overloaded int cast
    // convert Kilometer -> int
    operator double() const { return km; }

    void input()
    {
        cout << "Enter distance in kilometers:\t";
        cin >> km;
    }

    void display()
    {
        cout << "Distance  =  " << km << " km\n";
    }
};

// print a double value
void printDouble(double d)
{
    cout << "Double value  =  " << d << "\n";
}

int main()
{
    Kilometer k;
    k.input();
    k.display();
    cout << "\nConvert to double:\n";
    printDouble(k);

    cout << "\n\n\n";

    double d;
    cout << "Enter a double:\t";
    cin >> d;
    printDouble(d);
    cout << "\nConvert to Kilometer:\n";
    k = d;
    k.display();
    return 0;
}
```

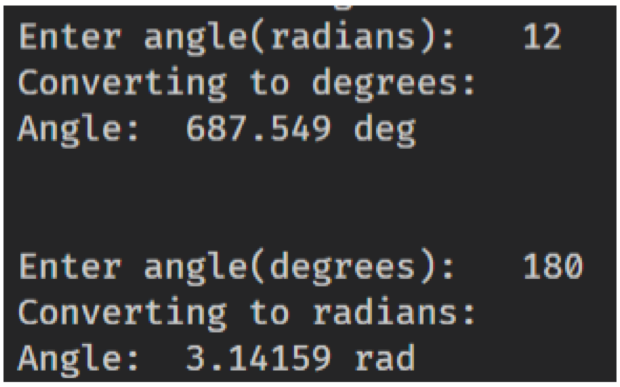


```
Enter distance in kilometers:   12.54
Distance  =  12.54 km

Convert to double:
Double value  =  12.54



Enter a double: 23
Double value  =  23

Convert to Kilometer:
Distance  =  23 km
```

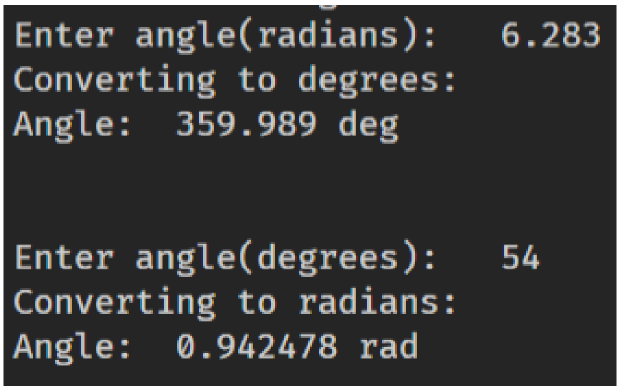
Program - 68

Data Conversion between objects of different classes(degree and radian—degree to radian and radian to degree) by defining Conversion routine in destination object.

```cpp
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
#define PI 3.14159265358979323846264
using namespace std;

class Degree
{
    double angle;

public:
    Degree(double a = 0.0) : angle(a) {}

    void input()
    {
        cout << "Enter angle(degrees):\t";
        cin >> angle;
    }

    void display()
    {
        cout << "Angle:\t" << angle << " deg\n";
    }

    double getAngle() const
    {
        return angle;
    }
};

class Radian
{
    double angle;

public:
    Radian(double a = 0.0) : angle(a) {}

    // to convert Radian -> Degree
    operator Degree() const
    {
        return Degree(angle * 180 / PI);
    }

    // to convert Degree -> Radian
    Radian(Degree d)
    {
        angle = d.getAngle() * PI / 180;
    }

    void input()
    {
        cout << "Enter angle(radians):\t";
        cin >> angle;
    }

    void display()
    {
        cout << "Angle:\t" << angle << " rad\n";
    }

    double getAngle() const
    {
        return angle;
    }
};

int main()
{
    Radian rad;
    Degree deg;

    // convert Radian -> Degree
    rad.input();
    cout << "Converting to degrees:\n";
    deg = rad;
    deg.display();

    cout << "\n\n";

    // convert Degree -> Radian
    deg.input();
    cout << "Converting to radians:\n";
    rad = deg;
    rad.display();

    return 0;
}
```



```
Enter angle(radians):   12
Converting to degrees:
Angle:  687.549 deg


Enter angle(degrees):   180
Converting to radians:
Angle:  3.14159 rad
```

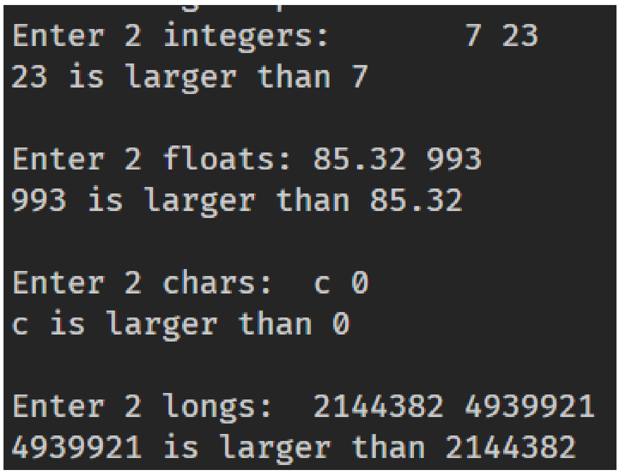


```
Enter angle(radians):   6.283
Converting to degrees:
Angle:  359.989 deg


Enter angle(degrees):   54
Converting to radians:
Angle:  0.942478 rad
```

Program - 69

Program to display largest among two numbers using function templates.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

// function prints largest of 2 values
// a generic type T
template <typename T>
void largest(T a, T b)
{
    // would throw error if instanes type T
    // are not comparable using relational operators
    if (a > b)
        cout << a << " is larger than " << b;
    else if (a < b)
        cout << b << " is larger than " << a;
    else
        cout << a << " = " << b;
}

int main()
{
    // COMPARING INT
    int i1, i2;
    cout << "Enter 2 integers:\t";
    cin >> i1 >> i2;
    largest(i1, i2);

    // COMPARING FLOAT
    float f1, f2;
    cout << "\n\nEnter 2 floats:\t";
    cin >> f1 >> f2;
    largest(f1, f2);

    // COMPARING CHAR
    char c1, c2;
    cout << "\n\nEnter 2 chars:\t";
    cin >> c1 >> c2;
    largest(c1, c2);

    // COMPARING LONG LONG
    long long l1, l2;
    cout << "\n\nEnter 2 longs:\t";
    cin >> l1 >> l2;
    largest(l1, l2);

    return 0;
}
```

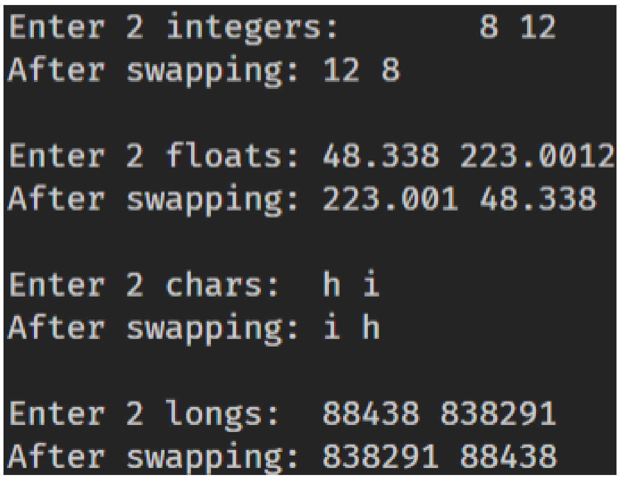


```
Enter 2 integers:       7 23
23 is larger than 7

Enter 2 floats: 85.32 993
993 is larger than 85.32

Enter 2 chars:  c 0
c is larger than 0

Enter 2 longs:  2144382 4939921
4939921 is larger than 2144382
```

Program - 70

Program to Swap two variables Using Function Templates (use it to swap char, float, long, int).

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using std::cin, std::cout;

// function swaps 2 references of any type
// a generic type T
template <typename T>
void swap(T &a, T &b)
{
    T temp = a;
    a = b;
    b = temp;
}

int main()
{
    // SWAP INTEGERS
    int i1, i2;
    cout << "Enter 2 integers:\t";
    cin >> i1 >> i2;
    swap(i1, i2);
    cout << "After swapping:\t" << i1 << " " << i2;

    // SWAP FLOATS
    float f1, f2;
    cout << "\n\nEnter 2 floats:\t";
    cin >> f1 >> f2;
    swap(f1, f2);
    cout << "After swapping:\t" << f1 << " " << f2;

    // SWAP CHARS
    char c1, c2;
    cout << "\n\nEnter 2 chars:\t";
    cin >> c1 >> c2;
    swap(c1, c2);
    cout << "After swapping:\t" << c1 << " " << c2;

    // SWAP LONG LONG
    long long l1, l2;
    cout << "\n\nEnter 2 longs:\t";
    cin >> l1 >> l2;
    swap(l1, l2);
    cout << "After swapping:\t" << l1 << " " << l2;

    return 0;
}
```

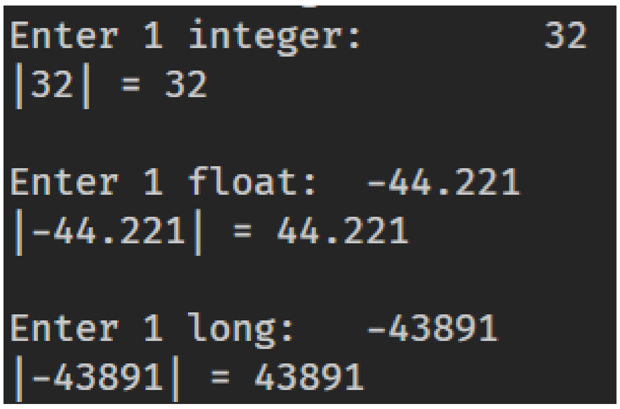


```
Enter 2 integers:       8 12
After swapping: 12 8

Enter 2 floats: 48.338 223.0012
After swapping: 223.001 48.338

Enter 2 chars:  h i
After swapping: i h

Enter 2 longs:  88438 838291
After swapping: 838291 88438
```

Program - 71

Program to define template function for abs() function. (use it to find absolute value of float, long, int).

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

// returns the absolute value of a
// a : |a|
// a generic type T
template <typename T>
T abs(T a)
{
    if (a < 0)
        return -a;
    else
        return a;
}

int main()
{
    // ABSOLUTE VALUE OF INT
    int i;
    std::cout << "Enter 1 integer:\t";
    std::cin >> i;
    std::cout << "|" << i << "|"
              << " = " << abs(i);

    // ABSOLUTE VALUE OF FLOAT
    float f;
    std::cout << "\n\nEnter 1 float:\t";
    std::cin >> f;
    std::cout << "|" << f << "|"
              << " = " << abs(f);

    // ABSOLUTE VALUE OF LONG LONG
    long long l;
    std::cout << "\n\nEnter 1 long:\t";
    std::cin >> l;
    std::cout << "|" << l << "|"
              << " = " << abs(l);

    return 0;
}
```



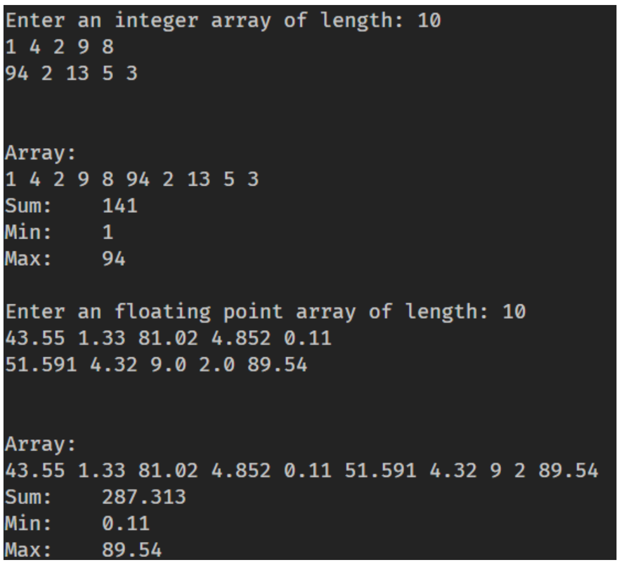
```
Enter 1 integer:        32
|32| = 32

Enter 1 float:  -44.221
|-44.221| = 44.221

Enter 1 long:   -43891
|-43891| = 43891
```

Program - 72

Write a program that inputs elements in integer array and floating point array using template. Find the sum of elements,find minimum, find maximum —for integer array and floating point array using template function. (so, two template functions)void input(T *array, const int n) // n is number of elements{}void sum_min_max(T* **array, const int n)

```
/**
* V. Sriram
* 10314902019
*/

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

// input array of generic type from user
template <typename T>
void input(T *arr, const int n)
{
    for (int i = 0; i < n; ++i)
    {
        cin >> arr[i];
    }
}

// print sum, min, max of array
template <typename T>
void sum_min_max(T *arr, const int n)
{
    T min = numeric_limits<T>::max();
    T max = numeric_limits<T>::min();
    T sum = 0;

    for (int i = 0; i < n; ++i)
    {
        if (arr[i] < min)
            min = arr[i];
        if (arr[i] > max)
            max = arr[i];

        sum += arr[i];
    }

    cout << "Sum:\t" << sum;
    cout << "\nMin:\t" << min;
    cout << "\nMax:\t" << max;
}

// print array of any type
template <typename T>
void printArray(T *arr, const int n)
{
    cout << "Array:\n";
    for (int i = 0; i < n; ++i)
    {
        cout << arr[i] << ' ';
    }
    cout << "\n";
}
```

```
int main()
{
    // INTEGER ARRAY
    const int length = 10;
    int arri[length];
    cout << "Enter an integer array of length: " << length << "\n";
    input(arri, length);
    cout << "\n\n";
    printArray(arri, length);
    sum_min_max(arri, length);

    cout << "\n\n";

    // FLOAT ARRAY
    float arrf[length];
    cout << "Enter an floating point array of length: " << length << "\n";
    input(arrf, length);
    cout << "\n\n";
    printArray(arrf, length);
    sum_min_max(arrf, length);

    return 0;
}
```



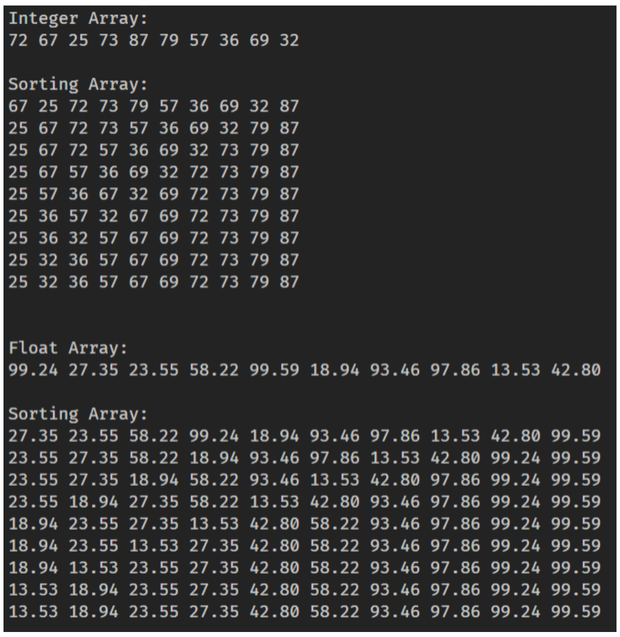
```
Enter an integer array of length: 10
1 4 2 9 8
94 2 13 5 3


Array:
1 4 2 9 8 94 2 13 5 3
Sum:    141
Min:    1
Max:    94

Enter an floating point array of length: 10
43.55 1.33 81.02 4.852 0.11
51.591 4.32 9.0 2.0 89.54


Array:
43.55 1.33 81.02 4.852 0.11 51.591 4.32 9 2 89.54
Sum:    287.313
Min:    0.11
Max:    89.54
```

Program - 73

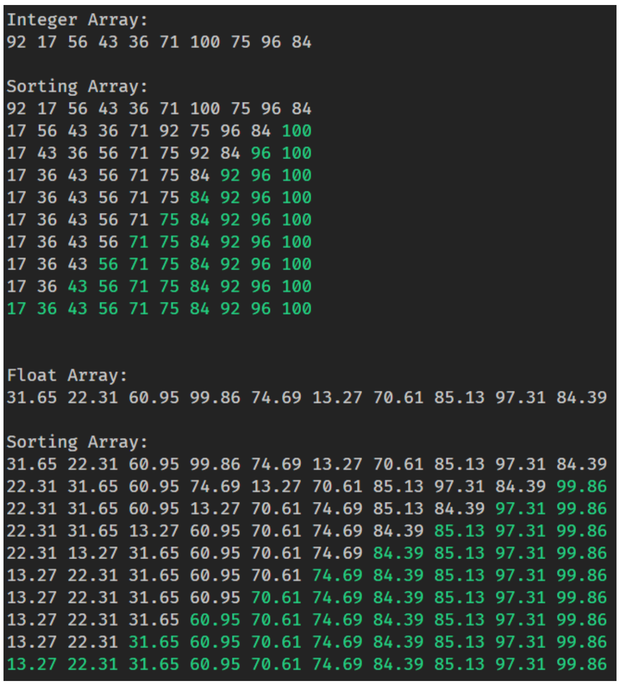
Demonstrate Bubblesort (for float and int) using template function

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
#include <cstdlib>
#include <ctime>
#include <windows.h>
using namespace std;

// swap references of any type
template <typename T>
void swapVals(T &a, T &b)
{
    T temp = b;
    b = a;
    a = temp;
}

// print array storing any type of data
template <typename T>
void printArray(T *arr, const int n, const int green = -1)
{
    HANDLE hConsole;
    hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    bool toColor = (green != -1);

    cout.setf(ios::fixed);
    cout.precision(2);

    for (int i = 0; i < n; ++i)
    {
        if (toColor)
        {
            if (i >= green)
                SetConsoleTextAttribute(hConsole, 0x0a);
            else
                SetConsoleTextAttribute(hConsole, 0x07);
        }
        cout << arr[i] << ' ';
    }

    SetConsoleTextAttribute(hConsole, 0x07);
    cout << "\n";
}

// bubble sort array storing comparable type of data
template <typename T>
void bubbleSort(T *arr, const int n)
{
    for (int i = 0; i < n - 1; ++i)
    {
        printArray(arr, n, n - i);
        for (int j = 0; j < n - i - 1; ++j)
        {
            if (arr[j] > arr[j + 1])
```

```cpp
                swapVals(arr[j], arr[j + 1]);
            }
        }
    printArray(arr, n, 0);
}

int main()
{
    const int length = 10;
    int arri[length];
    float arrf[length];

    // randomly generate array value
    srand(time(0));
    for (int i = 0; i < length; ++i)
    {
        // int in range 10-100
        arri[i] = rand() % 91 + 10;
        // float in range 10-100
        arrf[i] = 10 + (float)rand() / (float)(RAND_MAX / 90);
    }

    // INT ARRAY
    cout << "Integer Array:\n";
    printArray(arri, length);
    cout << "\nSorting Array:\n";
    bubbleSort(arri, length);

    cout << "\n\n";

    // FLOAT ARRAY
    cout << "Float Array:\n";
    printArray(arrf, length);
    cout << "\nSorting Array:\n";
    bubbleSort(arrf, length);

    return 0;
}
```



```
Integer Array:
72 67 25 73 87 79 57 36 69 32

Sorting Array:
67 25 72 73 79 57 36 69 32 87
25 67 72 73 57 36 69 32 79 87
25 67 72 57 36 69 32 73 79 87
25 67 57 36 69 32 72 73 79 87
25 57 36 67 32 69 72 73 79 87
25 36 57 32 67 69 72 73 79 87
25 36 32 57 67 69 72 73 79 87
25 32 36 57 67 69 72 73 79 87
25 32 36 57 67 69 72 73 79 87


Float Array:
99.24 27.35 23.55 58.22 99.59 18.94 93.46 97.86 13.53 42.80

Sorting Array:
27.35 23.55 58.22 99.24 18.94 93.46 97.86 13.53 42.80 99.59
23.55 27.35 58.22 18.94 93.46 97.86 13.53 42.80 99.24 99.59
23.55 27.35 18.94 58.22 93.46 13.53 42.80 97.86 99.24 99.59
23.55 18.94 27.35 58.22 13.53 42.80 93.46 97.86 99.24 99.59
18.94 23.55 27.35 13.53 42.80 58.22 93.46 97.86 99.24 99.59
18.94 23.55 13.53 27.35 42.80 58.22 93.46 97.86 99.24 99.59
18.94 13.53 23.55 27.35 42.80 58.22 93.46 97.86 99.24 99.59
13.53 18.94 23.55 27.35 42.80 58.22 93.46 97.86 99.24 99.59
13.53 18.94 23.55 27.35 42.80 58.22 93.46 97.86 99.24 99.59
```

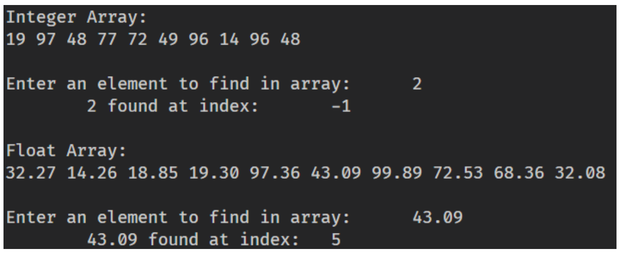


```
Integer Array:
92 17 56 43 36 71 100 75 96 84

Sorting Array:
92 17 56 43 36 71 100 75 96 84
17 56 43 36 71 92 75 96 84 100
17 43 36 56 71 75 92 84 96 100
17 36 43 56 71 75 84 92 96 100
17 36 43 56 71 75 84 92 96 100
17 36 43 56 71 75 84 92 96 100
17 36 43 56 71 75 84 92 96 100
17 36 43 56 71 75 84 92 96 100
17 36 43 56 71 75 84 92 96 100
17 36 43 56 71 75 84 92 96 100


Float Array:
31.65 22.31 60.95 99.86 74.69 13.27 70.61 85.13 97.31 84.39

Sorting Array:
31.65 22.31 60.95 99.86 74.69 13.27 70.61 85.13 97.31 84.39
22.31 31.65 60.95 74.69 13.27 70.61 85.13 97.31 84.39 99.86
22.31 31.65 60.95 13.27 70.61 74.69 85.13 84.39 97.31 99.86
22.31 31.65 13.27 60.95 70.61 74.69 84.39 85.13 97.31 99.86
22.31 13.27 31.65 60.95 70.61 74.69 84.39 85.13 97.31 99.86
13.27 22.31 31.65 60.95 70.61 74.69 84.39 85.13 97.31 99.86
13.27 22.31 31.65 60.95 70.61 74.69 84.39 85.13 97.31 99.86
13.27 22.31 31.65 60.95 70.61 74.69 84.39 85.13 97.31 99.86
13.27 22.31 31.65 60.95 70.61 74.69 84.39 85.13 97.31 99.86
13.27 22.31 31.65 60.95 70.61 74.69 84.39 85.13 97.31 99.86
```

Program - 74

Program to define template function for searching an element in arrays of different data types.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
#include <time.h>
#include <stdlib.h>
#include <cmath>
using namespace std;

// find element in array and return its index
template <typename T>
int findIndex(T *arr, const int n, T el)
{
    for (int i = 0; i < n; ++i)
        if (arr[i] == el)
            return i;

    return -1;
}

// print an array of size n
template <typename T>
void printArray(T *arr, const int n)
{
    cout.setf(ios::fixed);
    cout.precision(2);

    for (int i = 0; i < n; ++i)
        cout << arr[i] << ' ';

    cout << "\n";
}

int main()
{
    const int length = 10;
    int arri[length], eli;
    float arrf[length], elf;

    // generate random array elemens
    srand(time(0));
    for (int i = 0; i < length; ++i)
    {
        // int in range 10-100
        arri[i] = rand() % 91 + 10;
        // float in range 10-100
        arrf[i] = 10 + (float)rand() / (float)(RAND_MAX / 90);
        arrf[i] = round(arrf[i] * 100) / 100;
    }

    // INT ARRAY
    cout << "Integer Array:\n";
    printArray(arri, length);
    cout << "\nEnter an element to find in array:\t";
    cin >> eli;
    cout << "\t" << eli << " found at index:\t" << findIndex(arri, length, eli);

    cout << "\n\n";

    // FLOAT ARRAY
    cout << "Float Array:\n";
    printArray(arrf, length);
    cout << "\nEnter an element to find in array:\t";
    cin >> elf;
    cout << "\t" << elf << " found at index:\t" << findIndex(arrf, length, elf);

    return 0;
}
```

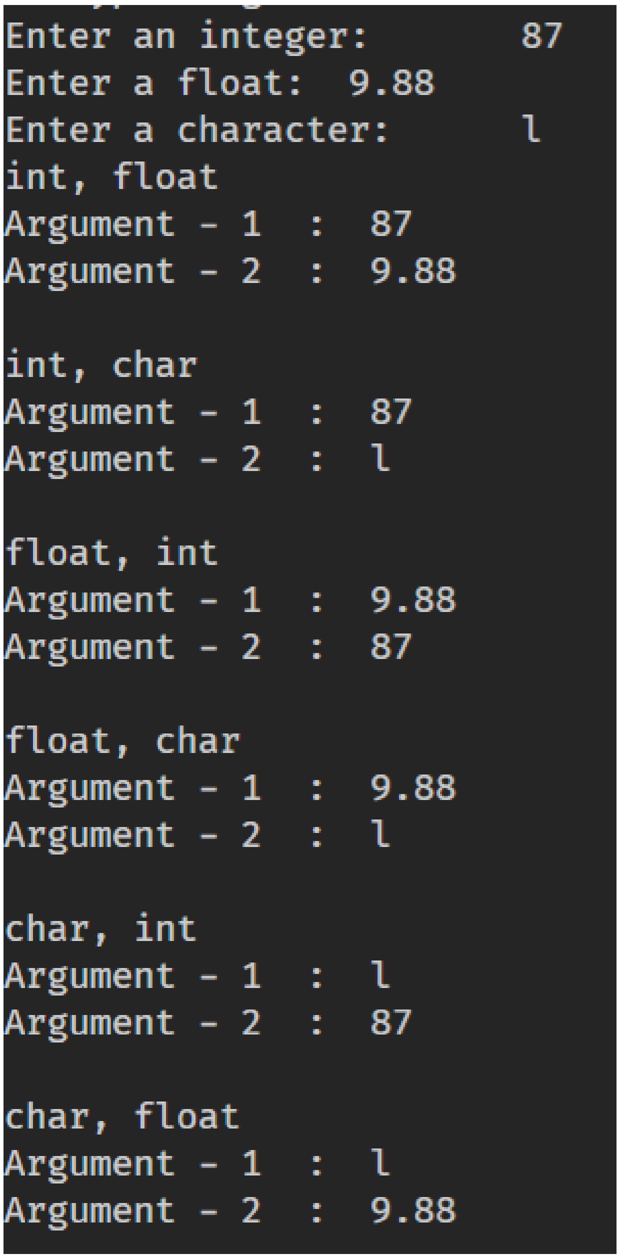


```
Integer Array:
19 97 48 77 72 49 96 14 96 48

Enter an element to find in array:      2
        2 found at index:       -1

Float Array:
32.27 14.26 18.85 19.30 97.36 43.09 99.89 72.53 68.36 32.08

Enter an element to find in array:      43.09
        43.09 found at index:   5
```

Program - 75

Program to demonstrate Function Template with Multiple Arguments of more than one generic type.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

// template taking multiple arguments
// with more than 1 type
template <typename T1, typename T2>
void display(T1 a, T2 b)
{
    // display the gicen arguments
    cout << "Argument - 1"
         << "  :  " << a << "\n";
    cout << "Argument - 2"
         << "  :  " << b << "\n\n";
}

int main()
{
    int i;
    float f;
    char c;

    cout << "Enter an integer:\t";
    cin >> i;

    cout << "Enter a float:\t";
    cin >> f;

    cout << "Enter a character:\t";
    cin >> c;

    // CALL FUNCTION WITH int AND floar
    cout << "int, float\n";
    display(i, f);

    // CALL FUNCTION WITH int AND char
    cout << "int, char\n";
    display(i, c);

    // CALL FUNCTION WITH float AND int
    cout << "float, int\n";
    display(f, i);

    // CALL FUNCTION WITH float AND char
    cout << "float, char\n";
    display(f, c);

    // CALL FUNCTION WITH char AND int
    cout << "char, int\n";
    display(c, i);

    // CALL FUNCTION WITH char AND float
    cout << "char, float\n";
    display(c, f);

    return 0;
}
```

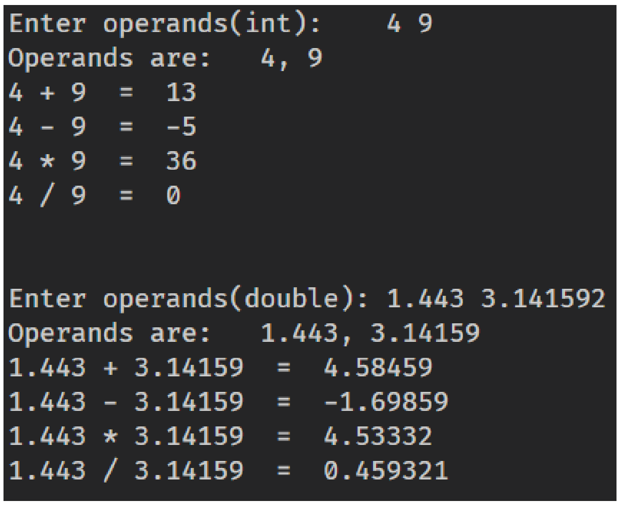


```
Enter an integer:       87
Enter a float:  9.88
Enter a character:      l
int, float
Argument - 1   :  87
Argument - 2   :  9.88

int, char
Argument - 1   :  87
Argument - 2   :  l

float, int
Argument - 1   :  9.88
Argument - 2   :  87

float, char
Argument - 1   :  9.88
Argument - 2   :  l

char, int
Argument - 1   :  l
Argument - 2   :  87

char, float
Argument - 1   :  l
Argument - 2   :  9.88
```

Program - 76

Basic calculator for addition, subtraction, multiplication and division.-Class template

```cpp
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

// calculator class
// operators on any type
template <typename T>
class Calculator
{
    T op1, op2;

public:
    Calculator(T op1, T op2) : op1(op1), op2(op2)
    {
        cout << "Operands are:\t" << op1 << ", " << op2;
    }

    // print results after operation
    void add()
    {
        cout << op1 << " + " << op2 << "  =  " << op1 + op2 << '\n';
    }

    void subtract()
    {
        cout << op1 << " - " << op2 << "  =  " << op1 - op2 << '\n';
    }

    void multiply()
    {
        cout << op1 << " * " << op2 << "  =  " << op1 * op2 << '\n';
    }

    void divide()
    {
        cout << op1 << " / " << op2 << "  =  " << op1 / op2 << '\n';
    }
};

int main()
{
    int op1, op2;
    cout << "Enter operands(int):\t";
    cin >> op1 >> op2;
    Calculator<int> calc1(op1, op2);

    cout << "\n";
    calc1.add();
    calc1.subtract();
    calc1.multiply();
    calc1.divide();

    cout << "\n\n";
    double op3, op4;
    cout << "Enter operands(double):\t";
    cin >> op3 >> op4;
    Calculator<double> calc2(op3, op4);

    cout << "\n";
    calc2.add();
    calc2.subtract();
    calc2.multiply();
    calc2.divide();

    return 0;
}
```

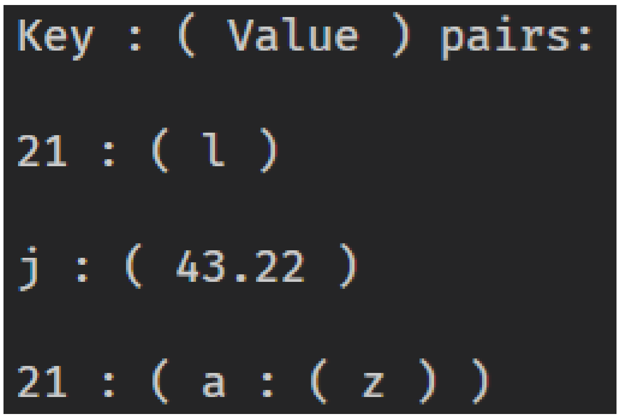


```
Enter operands(int):    4 9
Operands are:   4, 9
4 + 9  =  13
4 - 9  =  -5
4 * 9  =  36
4 / 9  =  0


Enter operands(double): 1.443 3.141592
Operands are:   1.443, 3.14159
1.443 + 3.14159  =  4.58459
1.443 - 3.14159  =  -1.69859
1.443 * 3.14159  =  4.53332
1.443 / 3.14159  =  0.459321
```

Program - 77

To demonstrate class template with Multiple parameters of more than one generic type.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

// class template with >1 generic type
// KeyValue stores a pair of data together
template <typename T1, typename T2>
class KeyValue
{
    T1 key;
    T2 value;

public:
    KeyValue(T1 k, T2 v) : key(k), value(v) {}

    // overload << opertor to print object of this class
    friend ostream &operator<<(ostream &out, const KeyValue &kv)
    {
        out << kv.key << " : ( " << kv.value << " )";
        return out;
    }
};

int main()
{
    // INT AND CHAR
    KeyValue<int, char> a(21, 'l');
    // CHAR AND FLOAT
    KeyValue<char, float> b('j', 43.22);
    // INT AND KeyValue which further stores CHAR AND CHAR
    KeyValue<int, KeyValue<char, char>> c(21, KeyValue<char, char>('a', 'z'));

    cout << "Key : ( Value ) pairs:\n\n";
    cout << a << "\n\n"
         << b << "\n\n"
         << c;

    return 0;
}
```

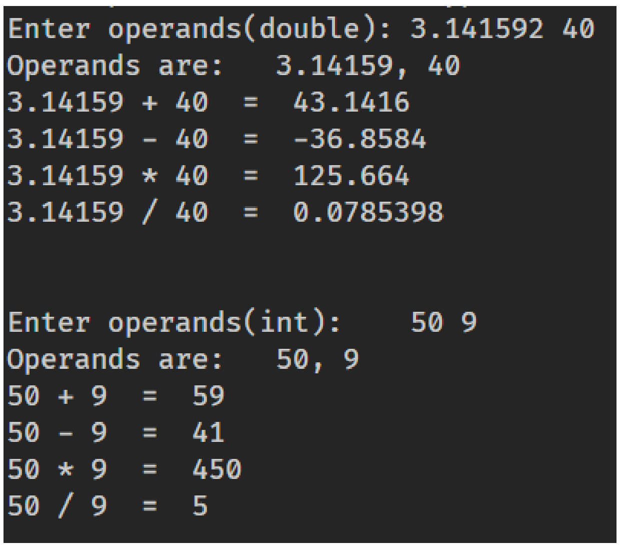


```
Key : ( Value ) pairs:

21 : ( l )

j : ( 43.22 )

21 : ( a : ( z ) )
```

Program - 78

To demonstrate Class Templates with default data type specified for the template argument.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

// template with default type
template <typename T = double>
class Calculator
{
    T op1, op2;

public:
    Calculator(T op1, T op2) : op1(op1), op2(op2)
    {
        cout << "Operands are:\t" << op1 << ", " << op2;
    }

    void add()
    {
        cout << op1 << " + " << op2 << "  =  " << op1 + op2 << '\n';
    }

    void subtract()
    {
        cout << op1 << " - " << op2 << "  =  " << op1 - op2 << '\n';
    }

    void multiply()
    {
        cout << op1 << " * " << op2 << "  =  " << op1 * op2 << '\n';
    }

    void divide()
    {
        cout << op1 << " / " << op2 << "  =  " << op1 / op2 << '\n';
    }
};

int main()
{
    // DOUBLE
    double op1, op2;
    cout << "Enter operands(double):\t";
    cin >> op1 >> op2;
    // using default explicit type = double
    Calculator calc1(op1, op2);

    cout << "\n";
    calc1.add();
    calc1.subtract();
    calc1.multiply();
    calc1.divide();

    cout << "\n\n";
    // INT
    int op3, op4;
    cout << "Enter operands(int):\t";
    cin >> op3 >> op4;
    // using implicit type
    Calculator<int> calc2(op3, op4);

    cout << "\n";
    calc2.add();
    calc2.subtract();
    calc2.multiply();
    calc2.divide();

    return 0;
}
```

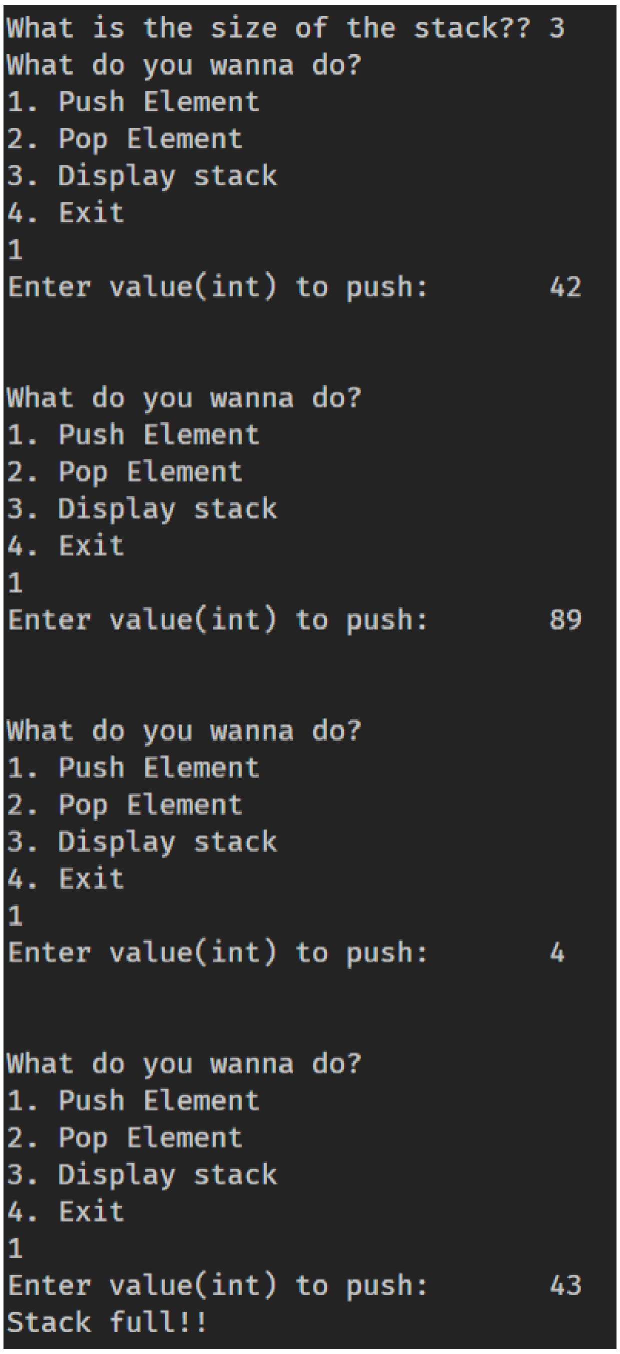


```
Enter operands(double): 3.141592 40
Operands are:   3.14159, 40
3.14159 + 40  =  43.1416
3.14159 - 40  =  -36.8584
3.14159 * 40  =  125.664
3.14159 / 40  =  0.0785398


Enter operands(int):    50 9
Operands are:   50, 9
50 + 9  =  59
50 - 9  =  41
50 * 9  =  450
50 / 9  =  5
```

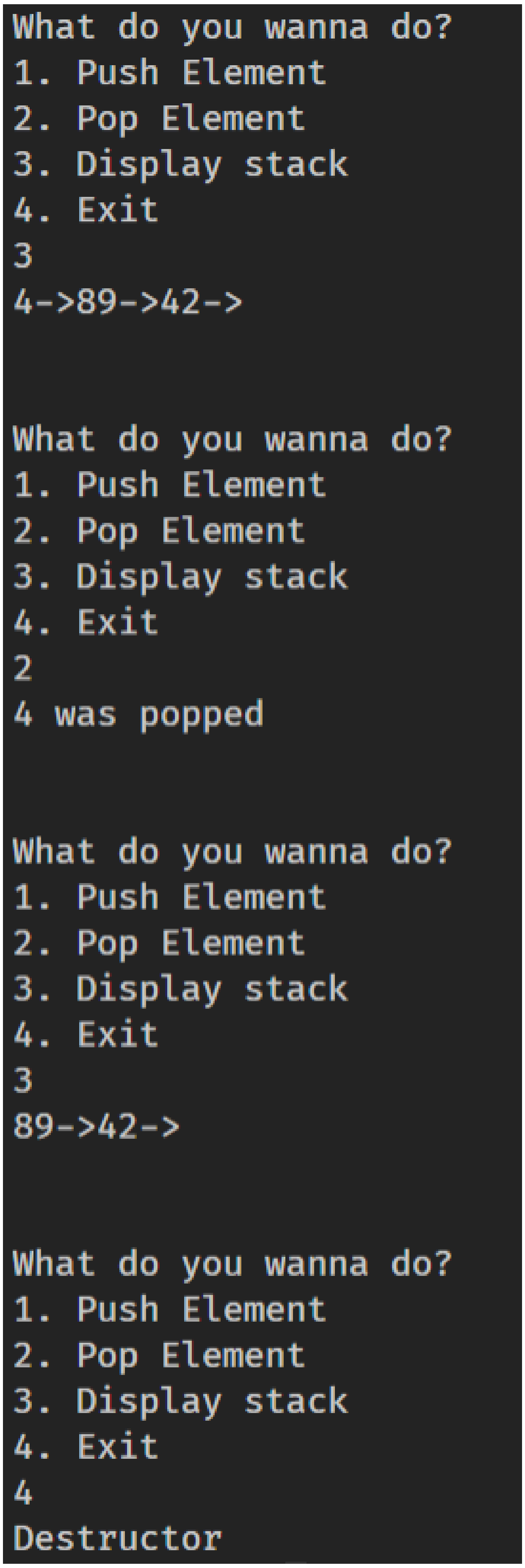
Program - 79

Implement stack for int and double using class template.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

// Stack class that can store data of any type
template <typename T>
class Stack
{
private:
    /*
    * arr is the stack storage
    * top points to last pushed element
    * stackSize is the max-size of stack
    */
    T *arr;
    int top, stackSize;

public:
    Stack(int size)
    {
        // use constructor to initialize max-size of stack
        // using dynamic allocation, deallocating memory
        // in destructor
        stackSize = size;
        arr = new T[size];
        top = -1;
    }

    // insert element in stack
    void push(int el)
    {
        if (top >= stackSize - 1)
        {
            cout << "Stack full!!\n";
            return;
        }
        arr[++top] = el;
    }

    // remove last pushed element from stack
    void pop()
    {
        if (top < 0)
        {
            cout << "Stack empty!!\n";
            return;
        }
        cout << arr[top--] << " was popped\n";
    }

    // print all elements in stack
    // int the direction they'll be popped
    void display()
    {
        if (top < 0)
        {
            cout << "Stack empty!!\n";
            return;
        }

        int counter = top;
        while (counter >= 0)
        {
            cout << arr[counter--] << "->";
        }
        cout << '\n';
    }

    ~Stack()
    {
        // deallocate arr memory which was allocated
        // in constructor
        cout << "Destructor";
        delete[] arr;
    }
};

int main()
{
    int size;
    cout << "What is the size of the stack??\t";
    cin >> size;

    // initialize stack size as entered by user
    // storing data of type int
    Stack<int> s(size);

    do
    {
        cout << "What do you wanna do?\n";
        cout << "1. Push Element\n";
        cout << "2. Pop Element\n";
        cout << "3. Display stack\n";
        cout << "4. Exit\n";

        // choice is the index of above printed options
        int choice;
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "Enter value(int) to push:\t";
            int el;
            cin >> el;
            s.push(el);
            break;

        case 2:
            s.pop();
            break;

        case 3:
            s.display();
            break;
        case 4:
            return 0;

        default:
            cout << "Invalid input!!";
            break;
        }

        cout << "\n\n";

    } while (true);

    return 0;
}
```



```
What is the size of the stack?? 3
What do you wanna do?
1. Push Element
2. Pop Element
3. Display stack
4. Exit
1
Enter value(int) to push:       42


What do you wanna do?
1. Push Element
2. Pop Element
3. Display stack
4. Exit
1
Enter value(int) to push:       89


What do you wanna do?
1. Push Element
2. Pop Element
3. Display stack
4. Exit
1
Enter value(int) to push:       4


What do you wanna do?
1. Push Element
2. Pop Element
3. Display stack
4. Exit
1
Enter value(int) to push:       43
Stack full!!
```

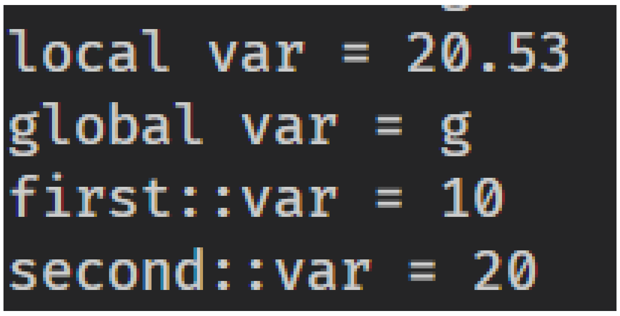


```
What do you wanna do?
1. Push Element
2. Pop Element
3. Display stack
4. Exit
3
4->89->42->


What do you wanna do?
1. Push Element
2. Pop Element
3. Display stack
4. Exit
2
4 was popped


What do you wanna do?
1. Push Element
2. Pop Element
3. Display stack
4. Exit
3
89->42->


What do you wanna do?
1. Push Element
2. Pop Element
3. Display stack
4. Exit
4
Destructor
```

Program - 80

To demonstrate nesting of namespace.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>

// nesting second inside first

namespace first
{
    int var = 10;
    namespace second
    {
        int secVar = 20;
    } // namespace second
} // namespace first
using namespace std;

char var = 'g';

int main()
{
    double var = 20.53;
    cout << "local var = " << var << endl;
    cout << "global var = " << ::var << endl;
    cout << "first::var = " << first::var << endl;     // access var defined inside
        namespace first
    cout << "second::var = " << first::second::secVar; // access secVar defined
        inside namespace second

    return 0;
}
```

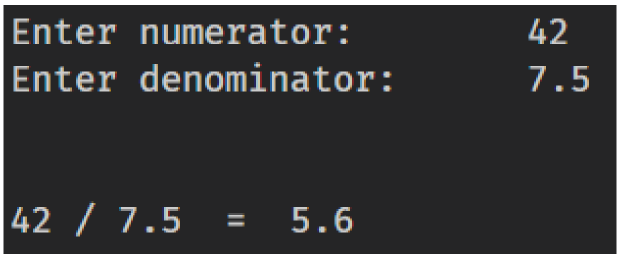


```
local var = 20.53
global var = g
first::var = 10
second::var = 20
```

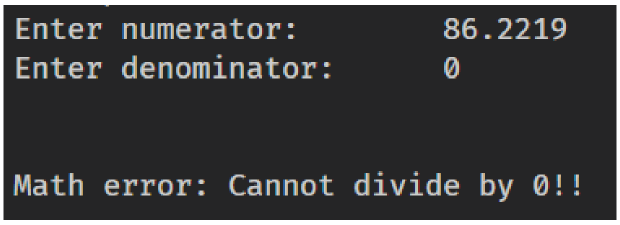
Program - 81

To demonstrate working of exception handling mechanism for divide by zero exception.

```
/**
* V. Sriram
* 10314902019
*/

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

// returns result of num divided by den
double divide(double num, double den)
{
    if (den == 0)
    {
        // numbers can't be divided by 0
        throw runtime_error("Math error: Cannot divide by 0!!");
    }

    return (num / den);
}

int main()
{
    double num, den, res;

    cout << "Enter numerator:\t";
    cin >> num;

    cout << "Enter denominator:\t";
    cin >> den;

    cout << "\n\n";

    // handle any exception thrown on calling divide
    try
    {
        res = divide(num, den);
        cout << num << " / " << den << "  =  " << res << "\n";
    }
    catch (runtime_error &e)
    {
        cout << e.what() << "\n";
    }

    return 0;
}
```



```
Enter numerator:        42
Enter denominator:      7.5


42 / 7.5  =  5.6
```

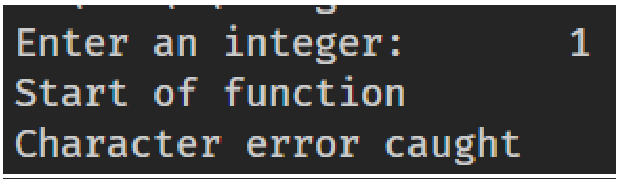


```
Enter numerator:        86.2219
Enter denominator:      0


Math error: Cannot divide by 0!!
```

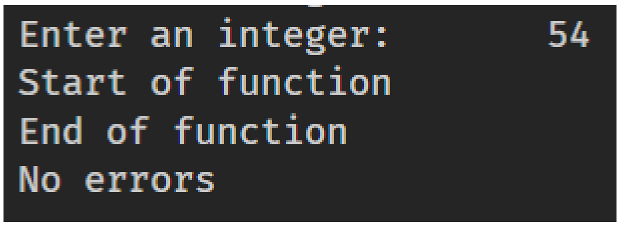
Program - 82

To demonstrate the exception handling mechanism for two types of exceptions.

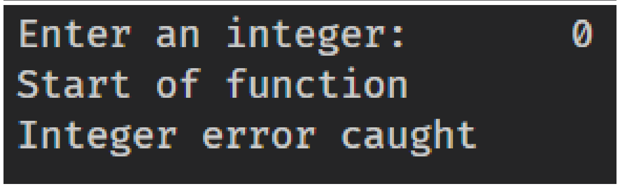
```cpp
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

// a dummy function that throws errors if n is 0 or 1
void check(int n)
{
    cout << "Start of function\n";

    if (n == 0)
        throw 1;
    else if (n == 1)
        throw 'c';

    cout << "End of function\n";
}

int main()
{
    int i;
    cout << "Enter an integer:\t";
    cin >> i;

    try
    {
        check(i);
        cout << "No errors\n";
    }
    catch (int e) // catch an error of type int
    {
        cout << "Integer error caught\n";
    }
    catch (char e) // catch an error of type char
    {
        cout << "Character error caught\n";
    }

    return 0;
}
```



```
Enter an integer:       1
Start of function
Character error caught
```



```
Enter an integer:       54
Start of function
End of function
No errors
```

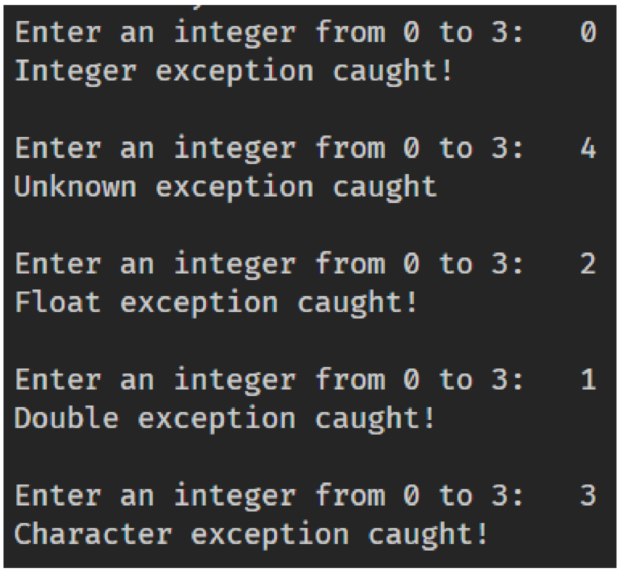


```
Enter an integer:       0
Start of function
Integer error caught
```

Program - 83

To demonstrate exceptions thrown by functions invoked from within try block.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

enum foodType
{
    BURGER,
    PIZZA,
    CHOLE_BHATURE,
    MOMOS
};

// dummy function that throws errors
void throwError(foodType t)
{
    if (t == BURGER)
        throw 1;
    else if (t == PIZZA)
        throw double(1.3);
    else if (t == CHOLE_BHATURE)
        throw float(1.3);
    else if (t == MOMOS)
        throw 'c';
    else
        throw t;
}

int main()
{
    for (int i = 0; i < 5; ++i)
    {
        try
        {
            int n;
            cout << "Enter an integer from " << BURGER << " to " << MOMOS << ":\t";
            cin >> n;

            throwError(foodType(n));
        }
        catch (int e) // catch an error of type int
        {
            cout << "Integer exception caught!";
        }
        catch (double e) // catch an error of type double
        {
            cout << "Double exception caught!";
        }
        catch (float e) // catch an error of type float
        {
            cout << "Float exception caught!";
        }
        catch (char e) // catch an error of type char
        {
            cout << "Character exception caught!";
        }
        catch (...) // catch error of any other type
        {
            cout << "Unknown exception caught";
        }

        cout << "\n\n";
    }

    return 0;
}
```

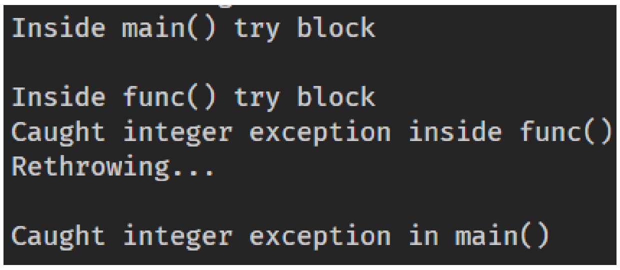


```
Enter an integer from 0 to 3:   0
Integer exception caught!

Enter an integer from 0 to 3:   4
Unknown exception caught

Enter an integer from 0 to 3:   2
Float exception caught!

Enter an integer from 0 to 3:   1
Double exception caught!

Enter an integer from 0 to 3:   3
Character exception caught!
```

Program - 84

To demonstrate rethrowing of an exception.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

// function that throws error
void func()
{
    try
    {
        cout << "Inside func() try block\n";
        throw 1;
    }
    catch (int e)
    {
        cout << "Caught integer exception inside func()\nRethrowing...\n";
        throw; // rethrow the error thrown in try block
    }
}

int main()
{
    try
    {
        cout << "Inside main() try block\n\n";
        func();
    }
    catch (int e) // catch any integer errors thrown
    {
        cout << "\nCaught integer exception in main()\n";
    }

    return 0;
}
```

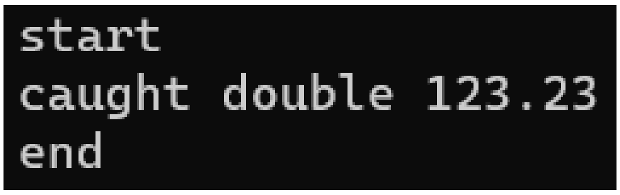


```
Inside main() try block

Inside func() try block
Caught integer exception inside func()
Rethrowing...

Caught integer exception in main()
```

Program - 85

To restrict a function to throw only a few specified types of exceptions.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
using namespace std;

void func(int t) throw(char, double)
{
    /** error as function can't throw int types
    if (t == 0)
        throw t;
    */
    if (t == 1)
        throw 'a';
    else if (t == 2)
        throw 123.23;
}

int main()
{
    cout << "start\n";

    try
    {
        func(2);
    }
    catch (int i) /*this catch statement must be ignored then,but it is running*/
    {
        cout << "caught an integer " << i; /*this is the output on the screen*/
    }
    catch (char c)
    {
        cout << "caught character " << c;
    }
    catch (double a)
    {
        cout << "caught double " << a;
    }

    cout << "\nend\n\n";

    return 0;
}
```

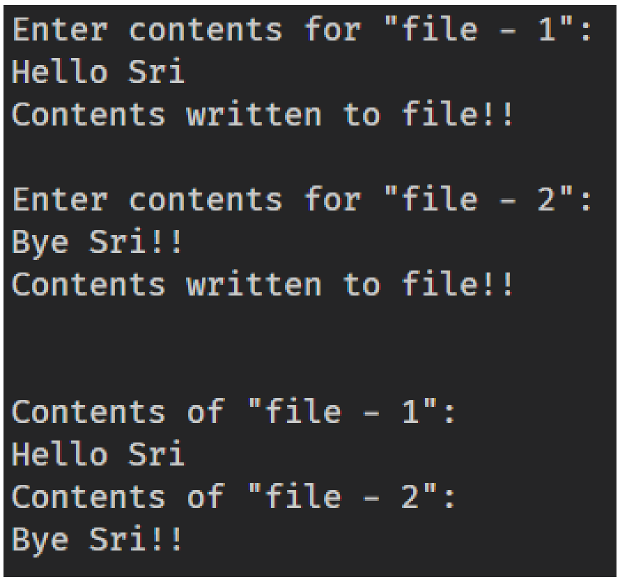


```
start
caught double 123.23
end
```

Program - 86

Create 2 files using open method of ofstream. Read the contents of each file one by one.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
    const int size = 2;
    // object to write to file
    ofstream filesWrite;

    for (int i = 0; i < size; ++i)
    {
        char str[50];

        cout << "Enter contents for \"file - " << i + 1 << "\":\n";
        cin.getline(str, 50, '\n');

        // open file and write user input to it
        filesWrite.open("file - " + to_string(i + 1) + ".txt");
        filesWrite << str << "\n";
        filesWrite.close();

        cout << "Contents written to file!!\n\n";
    }

    cout << "\n";

    // object to read from file
    ifstream filesRead;
    for (int i = 0; i < size; ++i)
    {
        char str[50];

        cout << "Contents of \"file - " << i + 1 << "\":\n";

        // open file, read and print contents inside
        filesRead.open("file - " + to_string(i + 1) + ".txt");
        filesRead.getline(str, 50, '\n');
        filesRead.close();

        cout << str << '\n';
    }

    return 0;
}
```

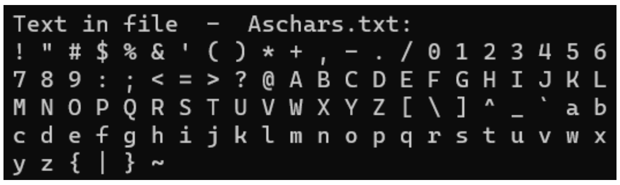


```
Enter contents for "file - 1":
Hello Sri
Contents written to file!!

Enter contents for "file - 2":
Bye Sri!!
Contents written to file!!


Contents of "file - 1":
Hello Sri
Contents of "file - 2":
Bye Sri!!
```

Program - 87

Write a program that uses the put() function to write all characters with ASCII values 33 to 127 to a file called Aschars.txt and then displays all

```
/**
* V. Sriram
* 10314902019
*/

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

int main()
{
    // object to read and right files
    fstream file;

    // open file in output mode
    file.open("Aschars.txt", ios::out);

    // store ascii characters from 33-127 into file
    for (int i = 33; i <= 127; ++i)
    {
        file.put(char(i));
        file.put(' ');
    }

    file.close();

    cout << "Text in file  -  Aschars.txt:\n";

    // open file in input mode to read
    file.open("Aschars.txt", ios::in);

    // read each character and print to console
    char c;
    while (file.get(c))
    {
        cout << c;
    }

    return 0;
}
```

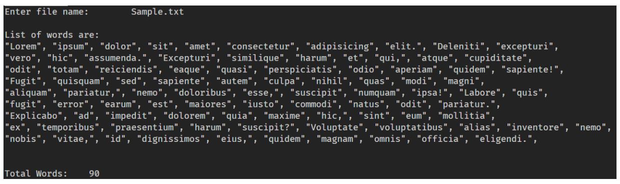


```
Text in file  -  Aschars.txt:
! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6
7 8 9 : ; < = > ? @ A B C D E F G H I J K L
M N O P Q R S T U V W X Y Z [ \ ] ^ _ ` a b
c d e f g h i j k l m n o p q r s t u v w x
y z { | } ~
```

Program - 88

Create a file with name “Sample.txt”, Write a program that opens “sample.txt” file in read mode and counts number of words in it.

```cpp
/**
* V. Sriram
* 10314902019
*/

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

// count number of words in file given by filename
void countWordsInFile(const char *fileName)
{
    // object to read files
    ifstream file(fileName);

    char str[128];
    int count = 0;

    cout << "\nList of words are:\n";
    // read file word by word
    while (file >> str)
    {
        count++;
        cout << "\"" << str << "\"," << (count % 10 == 0 ? "\n" : " ");
    }

    cout << "\n\nTotal Words:\t" << count;
}

int main()
{
    char fileName[FILENAME_MAX];

    // get filename as input
    cout << "Enter file name:\t";
    cin.getline(fileName, FILENAME_MAX, '\n');

    countWordsInFile(fileName);

    return 0;
}
```

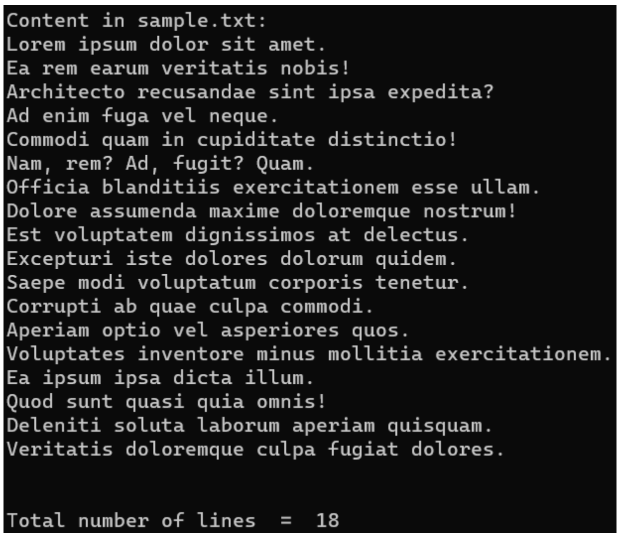


```
Enter file name:        Sample.txt

List of words are:
"Lorem", "ipsum", "dolor", "sit", "amet", "consectetur", "adipisicing", "elit.", "Deleniti", "excepturi",
"vero", "hic", "assumenda.", "Excepturi", "similique", "harum", "et", "qui,", "atque", "cupiditate",
"odit", "totam", "reiciendis", "eaque", "quasi", "perspiciatis", "odio", "aperiam", "quidem", "sapiente!",
"Fugit", "quisquam", "sed", "sapiente", "autem", "culpa", "nihil", "quas", "modi", "magni",
"aliquam", "pariatur,", "nemo", "doloribus", "esse,", "suscipit", "numquam", "ipsa!", "Labore", "quis",
"fugit", "error", "earum", "est", "maiores", "iusto", "commodi", "natus", "odit", "pariatur.",
"Explicabo", "ad", "impedit", "dolorem", "quia", "maxime", "hic,", "sint", "eum", "mollitia",
"ex", "temporibus", "praesentium", "harum", "suscipit?", "Voluptate", "voluptatibus", "alias", "inventore", "nemo",
"nobis", "vitae,", "id", "dignissimos", "eius,", "quidem", "magnam", "omnis", "officia", "eligendi.",


Total Words:    90
```

Program - 89

Write a program that opens “sample.txt” file in read mode and counts number of lines in it.

```cpp
/**
* V. Sriram
* 10314902019
*/

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

int main()
{
    // object to read from file
    ifstream file;
    file.open("sample.txt");

    cout << "Content in sample.txt:\n";
    int count = 0;
    char str[256];
    // count and print lines from file
    while (file.getline(str, 256, '\n'))
    {
        cout << str << "\n";
        count++;
    }

    cout << "\n\nTotal number of lines  =  " << count;

    return 0;
}
```

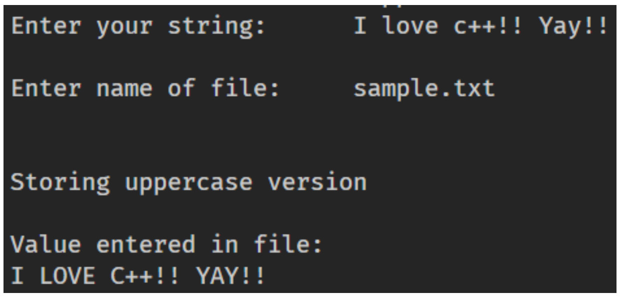


```
Content in sample.txt:
Lorem ipsum dolor sit amet.
Ea rem earum veritatis nobis!
Architecto recusandae sint ipsa expedita?
Ad enim fuga vel neque.
Commodi quam in cupiditate distinctio!
Nam, rem? Ad, fugit? Quam.
Officia blanditiis exercitationem esse ullam.
Dolore assumenda maxime doloremque nostrum!
Est voluptatem dignissimos at delectus.
Excepturi iste dolores dolorum quidem.
Saepe modi voluptatum corporis tenetur.
Corrupti ab quae culpa commodi.
Aperiam optio vel asperiores quos.
Voluptates inventore minus mollitia exercitationem.
Ea ipsum ipsa dicta illum.
Quod sunt quasi quia omnis!
Deleniti soluta laborum aperiam quisquam.
Veritatis doloremque culpa fugiat dolores.


Total number of lines  =  18
```

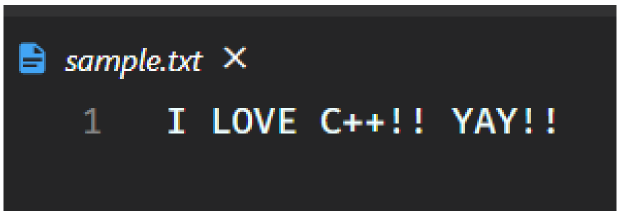
Program - 90

Write a Program that inputs alphabets in lowercase, coverts them into uppercase and writes to a file. (file name has been entered by user). Then it displays the contents of file.

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
#include <fstream>
#include <ctype.h>
using namespace std;

int main()
{
    // getting string from user
    const int stringMaxSize = 256;
    char str[stringMaxSize];
    cout << "Enter your string:\t";
    cin.getline(str, 256, '\n');

    // getting filename from user
    char filename[FILENAME_MAX];
    cout << "Enter name of file:\t";
    cin.getline(filename, FILENAME_MAX, '\n');

    // fstream object to work with files
    fstream file;

    // transforming user input to uppercase
    // storing in file specified by user
    cout << "\nStoring uppercase version";
    file.open(filename, ios::out);
    for (int i = 0; str[i] != '\0'; ++i)
    {
        file.put(toupper(str[i]));
    }
    file.close();

    // reading from file the contents entered before
    cout << "\n\nValue entered in file:\n";
    file.open(filename, ios::in);
    file.getline(str, stringMaxSize, '\n');
    cout << str << '\n';
    file.close();

    return 0;
}
```



```
Enter your string:      I love c++!! Yay!!

Enter name of file:     sample.txt


Storing uppercase version

Value entered in file:
I LOVE C++!! YAY!!
```

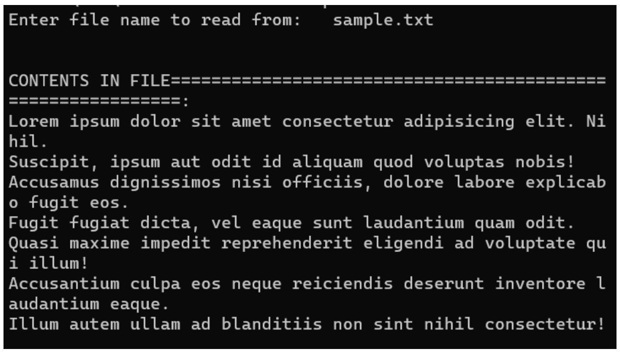


```
sample.txt ×
I LOVE C++!! YAY!!
```

Program - 91

Write a program that inputs name of the file from which characters are to be read from. Read each character and display the same till end of file is reached.Use sequential input and output operators.

```cpp
/**
* V. Sriram
* 10314902019
*/

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

int main()
{
    char filename[FILENAME_MAX];
    // get filename as input from user
    cout << "Enter file name to read from:\t";
    cin.getline(filename, FILENAME_MAX, '\n');

    // read character by character and print
    // contents of file to console
    cout << "\nCONTENTS IN
        FILE============================================================:\n";
    char c;
    ifstream file(filename);
    while (file.get(c))
    {
        cout << c;
    }

    return 0;
}
```

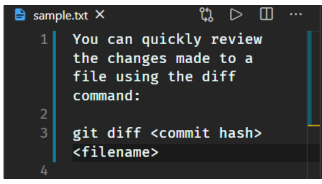


```
Enter file name to read from:   sample.txt


CONTENTS IN FILE=========================================
=================:
Lorem ipsum dolor sit amet consectetur adipisicing elit. Ni
hil.
Suscipit, ipsum aut odit id aliquam quod voluptas nobis!
Accusamus dignissimos nisi officiis, dolore labore explicab
o fugit eos.
Fugit fugiat dicta, vel eaque sunt laudantium quam odit.
Quasi maxime impedit reprehenderit eligendi ad voluptate qu
i illum!
Accusantium culpa eos neque reiciendis deserunt inventore l
audantium eaque.
Illum autem ullam ad blanditiis non sint nihil consectetur!
```

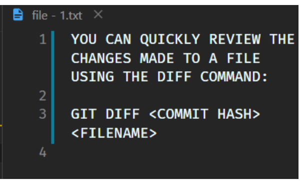
Program - 92

Write a C++ program to copy one file to another file after converting the lower-casecharacters to upper case characters.

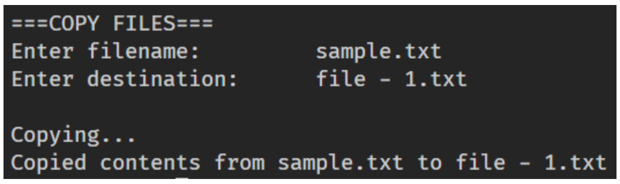
```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
#include <fstream>
#include <ctype.h>
using namespace std;

int main()
{
    char inputFileName[FILENAME_MAX], outputFileName[FILENAME_MAX];

    cout << "===COPY FILES===\n";

    // get file names from user
    cout << "Enter filename:\t\t";
    cin.getline(inputFileName, FILENAME_MAX, '\n');
    cout << "Enter destination:\t";
    cin.getline(outputFileName, FILENAME_MAX, '\n');

    // object to read from file
    ifstream inputFile(inputFileName);
    // object to right ot file
    ofstream outputFile(outputFileName);

    // copy all content, character by character
    // from inputFile and store in outputFile
    cout << "\nCopying...";
    char c;
    while (inputFile.get(c))
        outputFile.put(toupper(c));

    inputFile.close();
    outputFile.close();

    cout << "\nCopied contents from " << inputFileName << " to " << outputFileName;

    return 0;
}
```



```
You can quickly review
the changes made to a
file using the diff
command:

git diff <commit hash>
<filename>
```



```
YOU CAN QUICKLY REVIEW THE
CHANGES MADE TO A FILE
USING THE DIFF COMMAND:

GIT DIFF <COMMIT HASH>
<FILENAME>
```

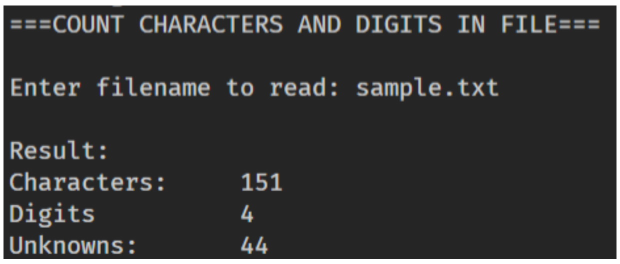


```
===COPY FILES===
Enter filename:         sample.txt
Enter destination:      file - 1.txt

Copying...
Copied contents from sample.txt to file - 1.txt
```

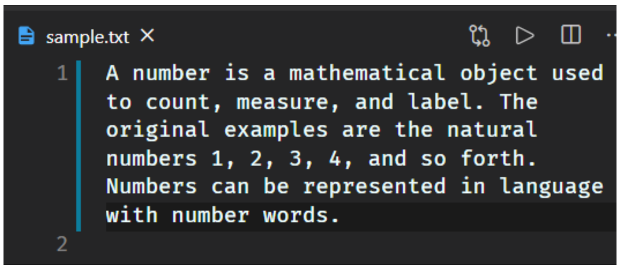
Program - 93

Write a C++ program to count the number of characters and digits in a file. This file has first to be created by the user only.

```cpp
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
#include <fstream>
#include <cctype>
using namespace std;

int main()
{
    cout << "===COUNT CHARACTERS AND DIGITS IN FILE===\n";

    // get filename from user
    char filename[FILENAME_MAX];
    cout << "\nEnter filename to read:\t";
    cin.getline(filename, FILENAME_MAX, '\n');

    // object to read from file
    ifstream file(filename);
    int charCount, digCount, unknownCount;
    charCount = digCount = unknownCount = 0;

    // read character by character and check
    // if its a letter or digit or unknown value
    char c;
    while (file.get(c))
    {
        if (isalpha(c))
            charCount++;
        else if (isdigit(c))
            digCount++;
        else
            unknownCount++;
    }

    // don't forget to close the file to avoid corruption
    file.close();

    cout << "\nResult:";
    cout << "\nCharacters:\t" << charCount;
    cout << "\nDigits\t\t" << digCount;
    cout << "\nUnknowns:\t" << unknownCount;

    return 0;
}
```



```
===COUNT CHARACTERS AND DIGITS IN FILE===

Enter filename to read: sample.txt

Result:
Characters:     151
Digits          4
Unknowns:       44
```

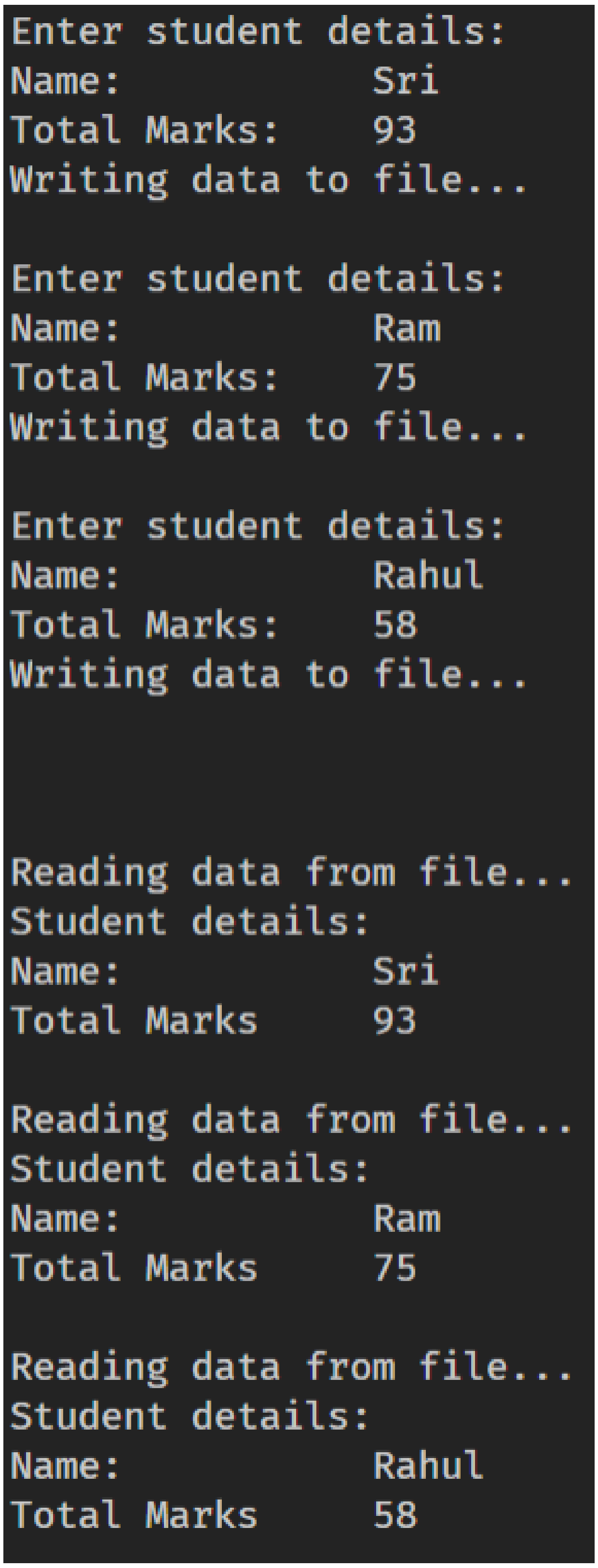


```
A number is a mathematical object used
to count, measure, and label. The
original examples are the natural
numbers 1, 2, 3, 4, and so forth.
Numbers can be represented in language
with number words.
```

Program - 94

Demonstrate Reading & writing object of the classin a file..

```
/**
* V. Sriram
* 10314902019
*/

#include <iostream>
#include <cstring>
#include <fstream>
using namespace std;

// student class
class Student
{
    char name[50];
    double totalMarks;

public:
    Student() {}

    Student(char name[], double totalMarks)
    {
        strcpy(this->name, name);
        this->totalMarks = totalMarks;
    }

    void input()
    {
        cout << "Enter student details:\n";

        cout << "Name:\t\t";
        cin >> name;

        cout << "Total Marks:\t";
        cin >> totalMarks;
    }

    void display()
    {
        cout << "Student details:\n";
        cout << "Name:\t\t" << name << "\n";
        cout << "Total Marks\t" << totalMarks << "\n";
    }
};

int main()
{
    char filename[] = "studentDetails.dat";
    // open binary file in output/write mode
    fstream file(filename, ios::binary | ios::out);

    // get student details and
    // write to files as binary data
    Student arr[3];
    for (int i = 0; i < 3; ++i)
    {
        arr[i].input();
        cout << "Writing data to file...";
        file.write((char *)(&arr[i]), sizeof(Student));
        cout << "\n\n";
    }

    file.close();

    cout << "\n\n";

    // open binary file in input/read mode
    file.open(filename, ios::binary | ios::in);

    // read each student data stored and print to console
    Student obj;
    for (int i = 0; i < 3; ++i)
    {
        cout << "Reading data from file...\n";
        file.read((char *)(&obj), sizeof(Student));
        obj.display();
        cout << "\n";
    }

    file.close();

    return 0;
}
```



```
Enter student details:
Name:               Sri
Total Marks:        93
Writing data to file...

Enter student details:
Name:               Ram
Total Marks:        75
Writing data to file...

Enter student details:
Name:               Rahul
Total Marks:        58
Writing data to file...



Reading data from file...
Student details:
Name:               Sri
Total Marks         93

Reading data from file...
Student details:
Name:               Ram
Total Marks         75

Reading data from file...
Student details:
Name:               Rahul
Total Marks         58
```