Object Oriented Programming With C++ Programs !!

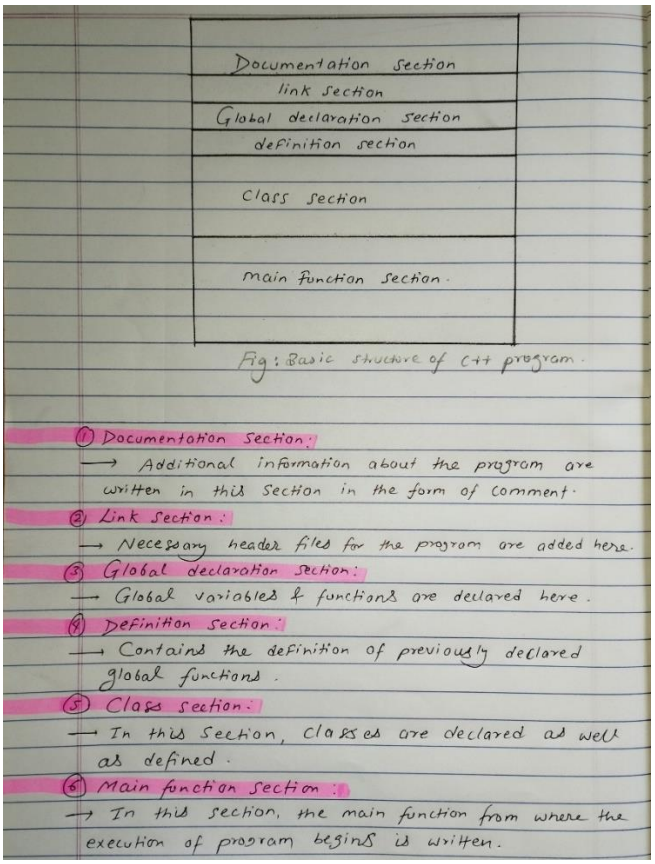
Name :
Contact :
College :
Address :

S.N	Title	Pg no.
1.	C++ Program Structure.	4
2.	Data input and output in C++.	4
3.	User defined function : taking argument and return value	4
4.	User defined function : taking argument and return no value	5
5.	User defined function : taking no argument and return value	5
6.	User defined function : taking no argument and return no value	5
7.	Function call : Call by value (program to swap values of 2 variables)	6
8.	Function call : Call by Reference (program to swap values of 2 variables)	6
9.	Default arguments demonstration program.	7
10.	Default argument program for calculating SI.	7
11.	Inline function Program for displaying square and cube of a number.	8
12.	Function overloading and ways to overload a function	8
13.	Function overloading by different data type parameters.	8
14.	Function overloading by different parameter numbers.	9
15.	Function overloading by different parameters sequence.	9
16.	Recursion function for Factorial of a number.	10
17.	Class demonstration Program.	10
18.	Defining member functions inside class body.	11
19.	Defining member functions outside class body.	11
20.	Scope Resolution Operator demonstration.	12
21.	Passing objects as arguments (program to add 2 complex numbers).	12
22.	Returning objects from Function (program to add 2 complex numbers).	13
23.	Friend function to swap values of 2 private members (variables).	14
24.	Friend function to calculate the area and perimeter of a circle.	14
25.	Static data Members and Static Member function.	15
26.	Member function of one class having friend function of another class.	16
27.	Returning no object from function : program to add 2 Heights (feet, inches)	17
28.	Returning object from function : program to add 2 Heights (feet, inches)	18
29.	Using friend function : program to add 2 Heights (feet, inches)	18
30.	Returning no object from function : Program to add 2 Times (Hrs, mins, secs).	19
31.	Returning object from function : Program to add 2 Times (Hrs, mins, secs).	20
32.	Using friend function : Program to add 2 Times (Hrs, mins, secs).	21
33.	Returning no object from function : Program to add 2 Distances (Km, meter, cm).	22
34.	Returning object from function : Program to add 2 Distances (Km, meter, cm).	23
35.	Using friend function : Program to add 2 Distances (Km, meter, cm).	24
36.	‘‘this” pointer first program.	24
37.	‘‘this” pointer second program.	25
38.	‘‘this” pointer third program.	25
39.	Friend class demonstration program.	26
40.	Program to show Default constructor	27
41.	Program to show Parameterized constructor	27
42.	Program to show Copy constructor	27
43.	Program to show all three constructors.	28
44.	Constructor overloading demonstration program.	29
45.	Destruction demonstration program.	30
46.	Enumeration : “enum” keyword demonstration first program.	30
47.	Enumeration : “enum” keyword demonstration Second program.	31
48.	Dynamic memory allocation for single integer value.	32
49.	Dynamic memory allocation for an array integer values.	33
50.	Dynamic memory allocation program from note.	34

S.N	Title	Pg no.
51.	Dynamic memory allocation of class object array using arrow (->) operator.	34
52.	Uniary Operator (-) overloading using class function.	35
53.	Uniary Operator (-) overloading using friend function.	36
54.	Binary Operator (+) overloading using class function.	37
55.	Binary Operator (+) overloading using friend function.	38
56.	Binary operator (>) overloading using class function.	39
57.	Basic to Basic data type conversion first progarm.	39
58.	Basic to Basic data type conversion second progarm.	40
59.	Basic to Class type conversion using constructor. (for time)	41
60.	Basic to Class type conversion by overloading " = " operator. (for time)	41
61.	Basic to Class type conversion using constructor. (for distance)	42
62.	Basic to Class type conversion by overloading " = " operator. (for distance)	43
63.	Class to Basic type conversion. (for time)	44
64.	Class to Basic type conversion. (for distance)	44
65.	Class to Class type conversion using conversion function in destination class.	45
66.	Class to Class type conversion using conversion function in source class.	47
67.	Access modes in Inheritance.	48
68.	Private access mode demonstration.	48
69.	Protected access mode demostration.	49
70.	Public access mode demonstration.	50
71.	Single inheritance.	51
72.	Multi-level Inheritance first program.	51
73.	Multi-level Inheritance second program.	52
74.	Multiple Inheritance.	53
75.	Hierarchical Inheritance.	54
76.	Hybrid Inheritance.	55
77.	Constructor in derived class first program.	56
78.	Constructor in derived class second program.	57
79.	Virtual Base class demonstration program.	59
80.	Ambiguity in inheritance program.	60
81.	Single inheritance same function name in both base and derived class.	61
82.	Function overriding program.	61
83.	Containership demonstration program.	62
84.	Virtual function : Run time polymorphism	63
85.	Pure virtual function demonstration program.	65
86.	Abstract Class demonstration program.	66
87.	Exception Handeling for division with single catch block	67
88.	Exception Handeling for division with multiple catch block	67
89.	Namespace demonstration Program.	68
90.	" using " keyword demonstration program.	69
91.	Template description.	70
92.	Function template for displaying any 2 values.	70
93.	Function template for determining the larger value between any 2 values.	71
94.	Class template for calculating the area of rectangle.	71
95.	Stream classes description.	72
96.	Writing on a file using ofstream class.	72
97.	Writing on a file using fstream class.	73
98.	Writing (name, marks[5], roll) on a file using ofstream class.	73
99.	Reading data of student (name, marks[5], roll) from file using ifsream.	74
100.	Writing on file using open() member function.	75

101.	Writing on a file using terminal in real time.	76
102.	Previous year Exam Qn.	77

1. C++ Program Structure.



2. Data input and output in C++.

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

int main()
{
    int x;
    char name[25];
    cout<<"Enter your full name : ";
    cin.get(name,sizeof(name));

    cout<<"Enter any number : ";
    cin>>x;

    cout<<"Name : "<<name<<endl;
    cout<<"The number you entered is "<<x<<endl;
    return 0;
}
```

3. User defined function: taking argument and return value.

```
#include <iostream>
using namespace std;
int sum(int, int, int);
int main()
{
    int a, b, c, total;
    cout << "Enter any 3 numbers : " << endl;
```

```
    cin >> a >> b >> c;
    total = sum(a, b, c);
    cout << "The Sum of " << a << ", " << b << " and " << c << " is " << total << endl;
    return 0;
}
int sum(int n1, int n2, int n3)
{
    return (n1 + n2 + n3);
}
```

4. User defined function: taking argument and return no value.

```
#include <iostream>
using namespace std;
void sum(int, int, int);
int main()
{
    int a, b, c;
    cout << "Enter any 3 numbers." << endl;
    cin >> a >> b >> c;
    sum(a, b, c);
    return 0;
}

void sum(int n1, int n2, int n3)
{
    int total;
    total = n1 + n2 + n3;
    cout << "The sum of " << n1 << ", " << n2 << " and " << n3 << " is " << total << endl;
}
```

5. User defined function: taking no argument and return value.

```
#include <iostream>
using namespace std;
int sum();
int main()
{
    int total;
    total = sum();
    cout << "The sum of 3 numbers is " << total << endl;
    return 0;
}
int sum()
{
    int a, b, c;
    cout << "Enter any 3 numbers." << endl;
    cin >> a >> b >> c;
    return (a + b + c);
}
```

6. User defined function: taking no argument and return no value.

```
#include <iostream>
using namespace std;
void sum();
int main()
```

```
{
    sum();
    return 0;
}

void sum()
{
    int a, b, c, total;
    cout << "Enter any 3 numbers." << endl;
    cin >> a >> b >> c;
    total = a + b + c;
    cout << "The sum of " << a << ", " << b << ", "
         << "and " << c << " is " << total << endl;
}
```

7. Function call: Call by value (program to swap values of 2 variables).

```
#include <iostream>
using namespace std;
void swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
    cout << "Displaying values from inside swap() after calling swap():" << endl;
    cout << "a : " << x << "\t b : " << y << endl;
}
int main()
{
    int a = 10, b = 15;
    cout << "Values Before Swapping:" << endl;
    cout << "a : " << a << "\t b : " << b << endl;
    swap(a, b);
    cout << "Displaying values from inside main() after calling swap():" << endl;
    cout << "a : " << a << "\t b : " << b << endl;
    return 0;
}
```

Output:

```
Values Before Swapping:
a : 10   b : 15
Displaying values from inside swap() after calling swap():
a : 15   b : 10
Displaying values from inside main() after calling swap():
a : 10   b : 15
```

8. Function call: Call by Reference (program to swap values of 2 variables).

```
#include <iostream>
using namespace std;
void swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
```

```
    cout << "Displaying values from inside swap() after calling swap():" << endl;
    cout << "a : " << *x << "\t b : " << *y << endl;
}
int main()
{
    int a = 10, b = 15;
    cout << "Values Before Swapping:" << endl;
    cout << "a : " << a << "\t b : " << b << endl;
    swap(&a, &b);
    cout << "Displaying values from inside main() after calling swap():" << endl;
    cout << "a : " << a << "\t b : " << b << endl;
    return 0;
}
```

Output:

```
Values Before Swapping:
a : 10   b : 15
Displaying values from inside swap() after calling swap():
a : 15   b : 10
Displaying values from inside main() after calling swap():
a : 15   b : 10
```

9. Default arguments demonstration program.

```
#include <iostream>
using namespace std;
void total(int m1 = 10, int m2 = 20, int m3 = 30);
int main()
{
    cout << "Program to demonstrate use of default arguments." << endl;
    total();        // 10+20+30 --> sum = 60
    total(7);       // 7+20+30  --> sum = 57
    total(6, 5);    // 6+5+30   --> sum = 41
    total(4, 5, 6); // 4+5+6    --> sum = 15
    return 0;
}
void total(int a, int b, int c)
{
    cout << "sum = " << a + b + c << endl;
}
```

10. Default argument program for calculating SI.

```
#include <iostream>
using namespace std;
int simple_interest(int p, int t, int r = 8);
int main()
{
    int principle, time, interest;
    cout << "Enter principle amonut : ";
    cin >> principle;
    cout << "Enter time period in yrs : ";
    cin >> time;
    interest = simple_interest(principle, time);
```

```
    cout << "The Simple Interest amount is " << interest << endl;
    return 0;
}
int simple_interest(int p, int t, int r)
{
    int i;
    i = (p * t * r) / 100;
    return i;
}
```

11. Inline function Program for displaying square and cube of a number.

```
#include <iostream>
using namespace std;
inline int square(int x)
{
    return (x * x);
}
inline int cube(int x)
{
    return (x * x * x);
}
int main()
{
    int x, sq, cb;
    cout << "Enter any number : ";
    cin >> x;
    sq = square(x);
    cb = cube(x);
    cout << "The square of " << x << " is " << sq << endl;
    cout << "The cube of " << x << " is " << cb << endl;

    return 0;
}
```

12. Function overloading and ways to overload a function.

A feature of OOP where 2 or more functions can have same name but different parameters is called function overloading or function polymorphism.To overload a function with same name, the function must obey any one of the following conditions.

1. Parameters should have different type.
2. parameters numbers should be different.
3. Parameters should have different sequence.

13. Function overloading by different data type parameters.

```
#include <iostream>
using namespace std;
void add(float a, float b);
void add(int a, int b);
// --> Here, both add() have Same number of parameters but different parameter type.

void add(float a, float b)
{
    cout << "Sum of 2 floating pointer numbers : " << (a + b) << endl;
}
```

```
void add(int a, int b)
{
    cout << "Sum of 2 integer numbers : " << (a + b) << endl;
}
int main()
{
    add(10, 20);
    add(40.5f, 60.5f);
    return 0;
}
```

14. Function overloading by different parameter numbers.

```
#include <iostream>
using namespace std;
void add(int a, int b);
void add(int a, int b, int c);
// both add() have same type of parameters but parameters number is different.

void add(int a, int b)
{
    cout << "The sum of 2 integer numbers is : " << (a + b) << endl;
}
void add(int a, int b, int c)
{
    cout << "The sum of 3 integer numbers is : " << (a + b + c) << endl;
}

void add(float a, float b)
{
    cout << "The sum of 2 floating numbers is : " << (a + b)  << endl;
}
void add(float a, int b, float c)
{
    cout << "The sum of 3 floating numbers is : " << (a + b + c) << endl;
}

int main()
{
    add(55, 65);
    add(40, 67, 89);
    cout << endl;
    add(44.7f, 33.3f);
    add(23.4, 56.6f, 8.5f);
    return 0;
}
```

15. Function overloading by different parameters sequence.

```
#include <iostream>
using namespace std;
void add(int a, float b);
void add(float a, int b);
// --> Here, the function have same number of parameter but different sequence of
// parameters.

void add(int a, float b)
{
    cout << " --> Function of sequence (int, float) called!!" << endl;
    cout << "The sum of 2 integer numbers is : " << (a + b) << endl;
```

```
}

void add(float a, int b)
{
    cout << " --> Function of sequence (float, int) called !!" << endl;
    cout << "The sum of 2 floating numbers is : " << (a + b) << endl;
}
int main()
{
    add(44, 4.6f);
    cout << endl;
    add(4.6f, 44);
    return 0;
}
```

16. Recursion function for Factorial of a number.

```
#include <iostream>
using namespace std;
long fact(int n)
{
    if (n == 0 || n == 1) // Base condition
        return 1;
    return (n * fact(n - 1)); // Recursive function call
}
int main()
{
    int num;
    cout << "Enter a positive number." << endl;
    cin >> num;
    cout << "Factorial of " << num << " is " << fact(num);
    return 0;
}
```

17. Class demonstration Program.

```
#include <iostream>
using namespace std;
class student
{
private:
    char name[25];
    int roll;

public:
    void getdata()
    {
        cout << "Enter name : ";
        cin.get(name, sizeof(name));
        cout << "Enter roll number : ";
        cin >> roll;
    }
    void showdata()
    {
        cout << "Name : " << name << endl;
        cout << "Roll no : " << roll << endl;
    }
};

int main()
{
    student s; // s object created of student class.
```

```
    s.getdata();
    s.showdata();
    return 0;
}
```

18. Defining member functions inside class body.

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

public:
    void getx()
    {
        cout << "Enter value of x : " << endl;
        cin >> x;
    }
    void showx()
    {
        cout << "The value of x : " << x << endl;
    }
};
int main()
{
    A a1;
    a1.getx();
    a1.showx();
    return 0;
}
```

19. Defining member functions outside class body.

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

public:
    void getx();
    void showx();
};

// Here, Member functions are defined oustide the class body.
void A ::getx()
{
    cout << "Enter value of x : " << endl;
    cin >> x;
}


void A ::showx()
{
    cout << "The value of x : " << x << endl;
}
```

```
int main()
{
    A a1;
    a1.getx();
    a1.showx();
    return 0;
}
```

20. Scope Resolution Operator demonstration.

```
#include <iostream>
using namespace std;
int x = 30;
int main()
{
    int x = 70;
    cout << "The value of x inside main section is " << x << endl;    // ... is 70.
    cout << "The value of x outside main section is " << ::x << endl; // ... is 30.
    return 0;
}
```

21. Passing objects as arguments (program to add 2 complex numbers).

```
#include <iostream>
using namespace std;
class complex
{
private:
    float real, img;

public:
    void getcomplex();
    void addcomplex(complex, complex);
    void display();
};

// Defining all the member functions outside the class because it
// looks messy when we define function inside the class body.

void complex ::getcomplex()
{
    cout << "Enter real part : ";
    cin >> real;
    cout << "Enter img part : ";
    cin >> img;
}
void complex ::display()
{
    cout << "The sum is ";
    if (img > 0)
        cout << real << " + i" << img << endl;
    else
        cout << real << " - i" << (-1) * img << endl;
}
void complex::addcomplex(complex c1, complex c2)
{
    real = c1.real + c2.real;
    img = c1.img + c2.img;
}
int main()
```

```
{
    complex first, second, result;
    cout << "--> Enter first complex number :" << endl;
    first.getcomplex();
    cout << "--> Enter second complex number :" << endl;
    second.getcomplex();

    result.addcomplex(first, second);
    result.display();

    return 0;
}
```

22. Returning objects from Function (program to add 2 complex numbers).

```
#include <iostream>
using namespace std;
class complex
{
private:
    float real, img;

public:
    void getcomplex();
    complex addcomplex(complex);
    void display();
};
void complex ::getcomplex()
{
    cout << "Enter real part : ";
    cin >> real;
    cout << "Enter img part : ";
    cin >> img;
}
void complex ::display()
{
    cout << "The sum is ";
    if (img > 0)
        cout << real << " + i" << img << endl;
    else
        cout << real << " - i" << (-1) * img << endl;
}
complex complex::addcomplex(complex c)
{
    complex co;
    co.real = real + c.real;
    co.img = img + c.img;
    return co; // returning object 'co'.
}
int main()
{
    complex first, second, result;
    cout << "--> Enter first complex number :" << endl;
    first.getcomplex();
    cout << "--> Enter second complex number :" << endl;
    second.getcomplex();

    result = first.addcomplex(second);
    result.display();

    return 0;
}
```

23. Friend function to swap values of 2 private members (variables).

```
#include<iostream>
using namespace std;
class test
{
    private:
    int x = 50, y = 30;
    public:
    void displaydata()
    {
        cout<<"Value of x : "<<x<<endl
            <<"Value of y : "<<y<<endl;
    }
    friend void swap(test &);
};
void swap(test &a)
{
    int temp = a.x;
    a.x = a.y;
    a.y = temp;
    a.displaydata();
}
int main()
{
    test t;
    t.displaydata();
    cout<<"after swapping"<<endl;
    swap(t);
    cout<<"Inside main after swapping"<<endl;
    t.displaydata();
    return 0;
}
```

Output:

```
Value of x : 50
Value of y : 30
after swapping
Value of x : 30
Value of y : 50
Inside main after swapping
Value of x : 30
Value of y : 50
```

24. Friend function to calculate the area and perimeter of a circle.

```
#include<iostream>
#define pi 3.14
using namespace std;
class circle
{
    private:
    float radius;
    public:
    void getradius();
    friend void Area(circle);
    friend void perimeter(circle);
};
```

```
void circle :: getradius()
{
    cout<<"Enter the radius of circle : ";
    cin>>radius;
}

void Area(circle c)
{
    float area;
    area = pi*(c.radius)*(c.radius);
    cout<<"The area of circle is "<<area<<endl;
}
void perimeter(circle c)
{
    float perimeter;
    perimeter = 2*pi*c.radius;
    cout<<"The perimeter of circle is "<<perimeter<<endl;
}
int main()
{
    circle c;
    c.getradius();
    Area(c);
    perimeter(c);
    return 0;
}
```

25. Static data Members and Static Member function.

```
#include <iostream>
using namespace std;
class Employee
{
private:
    int id;
    static int count;
public:
    void displaydata()
    {
        count++;
        cout << "Hello, i am object no. " << count << endl;
    }
    static void displaycount()
    {
        cout << "The value of static data member count in this object is " << count;
       cout << endl;
    }
};

int Employee ::count; // Here default value is 0.
// int Employee :: count = 1000;  // <-- this is also
// valid.. we can assign any starting value here.

int main()
{
    int i;
    Employee e[5];
    for (i = 0; i < 5; i++)
    {
        e[i].displaydata();
        Employee::displaycount();
        cout<<endl;
```

```
    }
    return 0;
}
```

Output:

```
Hello, i am object no. 1
The value of static data member count in this object is 1

Hello, i am object no. 2
The value of static data member count in this object is 2

Hello, i am object no. 3
The value of static data member count in this object is 3

Hello, i am object no. 4
The value of static data member count in this object is 4

Hello, i am object no. 5
The value of static data member count in this object is 5
```

26. Member function of one class having friend function of another class.

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

class classA;
class classB
{
private:
    int y = 5;

public:
    void display()
    {
        cout << "y : " << y << endl;
    }
    void sum(classA &);
};
class classA
{
private:
    int x = 4;
public:
    void display()
    {
        cout << "x : " << x << endl;
    }
    friend void classB::sum(classA &);
};

void classB ::sum(classA &a)
{
    int sum;
    sum = a.x + y;
    cout << "Displaying the sum : " << sum << endl;
}
```

```
int main()
{
    classA a;
    classB b;
    a.display();
    b.display();
    b.sum(a);
    return 0;
}
```

27. Returning no object from function: program to add 2 Heights (feet, inches).

```
#include <iostream>
using namespace std;
class height
{
private:
    int feet, inches;

public:
    void getheight();
    void displayheight();
    void addheight(height, height);
    // height addheight(height);
};
void height ::getheight()
{
    cout << "Feet : ";
    cin >> feet;
    cout << "Inches : ";
    cin >> inches;
}
void height::displayheight()
{
    cout << feet << " feet " << inches << " Inches."<< endl;
}
void height::addheight(height h1, height h2)
{
    inches = h1.inches + h2.inches;
    feet = inches / 12;
    inches = inches % 12;
    feet = feet + h1.feet + h2.feet;
}
int main()
{
    height h1, h2,result;
    cout<<"Enter first height:"<<endl;
    h1.getheight();
    cout<<"Enter second height:"<<endl;
    h2.getheight();
    cout<<"The first entered height is "<<endl;
    h1.displayheight();
    cout<<"The second entered height is "<<endl;
    h2.displayheight();
    result.addheight(h1,h2);
    cout<<"The sum of these 2 heigts is ";
    result.displayheight();

    return 0;
}
```

28. Returning object from function: program to add 2 Heights (feet, inches).

```
#include <iostream>
using namespace std;
class height
{
private:
    int feet, inches;

public:
    void getheight();
    void displayheight();
    height addheight(height);
};
void height ::getheight()
{
    cout << "Feet : ";
    cin >> feet;
    cout << "Inches : ";
    cin >> inches;
}
void height::displayheight()
{
    cout << feet << " feet " << inches << " Inches." << endl;
}

height height ::addheight(height h)
{
    height temp;
    temp.inches = inches + h.inches;
    temp.feet = temp.inches / 12;
    temp.inches = temp.inches % 12;
    temp.feet = temp.feet + feet + h.feet;
    return temp;
}

int main()
{
    height h1, h2, result;
    cout << "Enter first height:" << endl;
    h1.getheight();
    cout << "Enter second height:" << endl;
    h2.getheight();
    cout << "The first entered height is " << endl;
    h1.displayheight();
    cout << "The second entered height is " << endl;
    h2.displayheight();

    result = h1.addheight(h2);
    cout << "The sum of these 2 heigts is ";
    result.displayheight();
    return 0;
}
```

29. Using friend function: program to add 2 Heights (feet, inches).

```
#include <iostream>
using namespace std;
class height
{
private:
```

```
    int feet, inches;

public:
    void getheight();
    void displayheight();
    friend height addheight(height &, height &);
};
void height ::getheight()
{
    cout << "Feet : ";
    cin >> feet;
    cout << "Inches : ";
    cin >> inches;
}
void height::displayheight()
{
    cout << feet << " feet " << inches << " Inches." << endl;
}

height addheight(height &h1, height &h2)
{
    height temp;
    temp.inches = h1.inches + h2.inches;
    temp.feet = temp.inches / 12;
    temp.inches = temp.inches % 12;
    temp.feet = temp.feet + h1.feet + h2.feet;
    return temp;
}

int main()
{
    height h1, h2, result;
    cout << "Enter first height:" << endl;
    h1.getheight();
    cout << "Enter second height:" << endl;
    h2.getheight();
    cout << "The first entered height is " << endl;
    h1.displayheight();
    cout << "The second entered height is " << endl;
    h2.displayheight();
    result = addheight(h1, h2);
    cout << "The sum of these 2 heigts is ";
    result.displayheight();

    return 0;
}
```

30. Returning no object from function: Program to add 2 Times (Hrs, mins, secs).

```
#include <iostream>
using namespace std;
class Time
{
private:
    int hour, min, sec;

public:
    void getTime();
    void displayTime();
    void addTime(Time, Time);
};
```

```
void Time ::getTime()
{
    cout << "Hours : ";
    cin >> hour;
    cout << "Minutes : ";
    cin >> min;
    cout << "Seconds : ";
    cin >> sec;
}
void Time ::displayTime()
{
    cout << hour << " Hour " << min << " Minutes " << sec << " Seconds." << endl;
}
void Time::addTime(Time t1, Time t2)
{
    sec = t1.sec + t2.sec;
    min = sec / 60;
    sec = sec % 60;
    min = min + t1.min + t2.min;
    hour = min / 60;
    min = min % 60;
    hour = hour + t1.hour + t2.hour;
}
int main()
{
    Time t1, t2, result;
    cout << " --> Enter first time:" << endl;
    t1.getTime();
    cout << " --> Enter second time:" << endl;
    t2.getTime();
    result.addTime(t1, t2);
    cout << "The sum of these 2 times is ";
    result.displayTime();
    return 0;
}
```

31. Returning object from function: Program to add 2 Times (Hrs, mins, secs).

```
#include <iostream>
using namespace std;
class Time
{
private:
    int hour, min, sec;

public:
    void getTime();
    void displayTime();
    Time addTime(Time);
};

void Time ::getTime()
{
    cout << "Hours : ";
    cin >> hour;
    cout << "Minutes : ";
    cin >> min;
    cout << "Seconds : ";
    cin >> sec;
}
void Time ::displayTime()
```

```
{
    cout << hour << " Hour " << min << " Minutes " << sec << " Seconds." << endl;
}
Time Time::addTime(Time t)
{
    Time temp;
    temp.sec = sec + t.sec;
    temp.min = temp.sec / 60;
    temp.sec = temp.sec % 60;
    temp.min = temp.min + min + t.min;
    temp.hour = temp.min / 60;
    temp.min = temp.min % 60;
    temp.hour = temp.hour + hour + t.hour;
    return temp;
}
int main()
{
    Time t1, t2, result;
    cout << " --> Enter first time:" << endl;
    t1.getTime();
    cout << " --> Enter second time:" << endl;
    t2.getTime();
    result = t1.addTime(t2);
    cout << "The sum of these 2 times is ";
    result.displayTime();
    return 0;
}
```

32. Using friend function: Program to add 2 Times (Hrs, mins, secs).

```
#include <iostream>
using namespace std;
class Time
{
private:
    int hour, min, sec;

public:
    void getTime();
    void displayTime();
    friend Time addTime(Time &, Time &);
};

void Time ::getTime()
{
    cout << "Hours : ";
    cin >> hour;
    cout << "Minutes : ";
    cin >> min;
    cout << "Seconds : ";
    cin >> sec;
}
void Time ::displayTime()
{
    cout << hour << " Hour " << min << " Minutes " << sec << " Seconds." << endl;
}

 Time addTime(Time &t1, Time &t2)
{
    Time temp;
    temp.sec = t1.sec + t2.sec;
    temp.min = temp.sec / 60;
```

```
    temp.sec = temp.sec % 60;
    temp.min = temp.min + t1.min + t2.min;
    temp.hour = temp.min / 60;
    temp.min = temp.min % 60;
    temp.hour = temp.hour + t1.hour + t2.hour;
    return temp;
}

int main()
{
    Time t1, t2, result;
    cout << " --> Enter first time:" << endl;
    t1.getTime();
    cout << " --> Enter second time:" << endl;
    t2.getTime();
    result = addTime(t1, t2);
    cout << "The sum of these 2 times is ";
    result.displayTime();
    return 0;
}
```

33. Returning no object from function: Program to add 2 Distances (Km, meter, cm).

```
#include<iostream>
using namespace std;
class Distance
{
    private:
    int km, meter, cm;
    public:
    void getdistance();
    void showdistance();
    void adddistance(Distance, Distance);
};
void Distance::getdistance()
{
    cout<<"Km : ";
    cin>>km;
    cout<<"Meter : ";
    cin>>meter;
    cout<<"cm : ";
    cin>>cm;
}
void Distance :: showdistance()
{
    cout<<km<<" km "<<meter<<" meter "<<cm<<" cm."<<endl;
}
void Distance::adddistance(Distance d1, Distance d2)
{
    cm = d1.cm + d2.cm;
    meter = cm/100;
    cm = cm % 100;
    meter = meter + d1.meter + d2.meter;
    km = meter / 1000;
    meter = meter % 1000;
    km = km + d1.km + d2.km;
}
int main()
{
    Distance d1, d2, result;
    cout<<" --> Enter first distance:"<<endl;
    d1.getdistance();
```

```
    cout<<" --> Enter second distance:"<<endl;
    d2.getdistance();
    result.adddistance(d1,d2);
    cout<<"The sum of these distances is ";
    result.showdistance();
    return 0;
}
```

34. Returning object from function: Program to add 2 Distances (Km, meter, cm).

```
#include<iostream>
using namespace std;
class Distance
{
    private:
    int km, meter, cm;
    public:
    void getdistance();
    void showdistance();
    Distance adddistance(Distance);
};
void Distance::getdistance()
{
    cout<<"Km : ";
    cin>>km;
    cout<<"Meter : ";
    cin>>meter;
    cout<<"cm : ";
    cin>>cm;
}
void Distance :: showdistance()
{
    cout<<km<<" km "<<meter<<" meter "<<cm<<" cm."<<endl;
}
Distance Distance::adddistance(Distance d)
{
    Distance temp;
    temp.cm = cm + d.cm;
    temp.meter = temp.cm/100;
    temp.cm = temp.cm % 100;
    temp.meter = temp.meter + meter + d.meter;
    temp.km = temp.meter / 1000;
    temp.meter = temp.meter % 1000;
    temp.km = temp.km + km + d.km;
    return temp;
}
int main()
{
    Distance d1, d2, result;
    cout<<" --> Enter first distance:"<<endl;
    d1.getdistance();
    cout<<" --> Enter second distance:"<<endl;
    d2.getdistance();

    result = d1.adddistance(d2);
    cout<<"The sum of these distances is ";
    result.showdistance();
    return 0;
}
```

35. Using friend function: Program to add 2 Distances (Km, meter, cm).

```
#include<iostream>
using namespace std;
class Distance
{
    private:
    int km, meter, cm;
    public:
    void getdistance();
    void showdistance();
    friend Distance adddistance(Distance &, Distance &);
};
void Distance::getdistance()
{
    cout<<"Km : ";
    cin>>km;
    cout<<"Meter : ";
    cin>>meter;
    cout<<"cm : ";
    cin>>cm;
}
void Distance :: showdistance()
{
    cout<<km<<" km "<<meter<<" meter "<<cm<<" cm."<<endl;
}
Distance adddistance (Distance &d1, Distance &d2)
{
    Distance temp;
    temp.cm = d1.cm + d2.cm;
    temp.meter = temp.cm/100;
    temp.cm = temp.cm % 100;
    temp.meter = temp.meter + d1.meter + d2.meter;
    temp.km = temp.meter / 1000;
    temp.meter = temp.meter % 1000;
    temp.km = temp.km + d1.km + d2.km;
    return temp;
}
int main()
{
    Distance d1, d2, result;
    cout<<" --> Enter first distance:"<<endl;
    d1.getdistance();
    cout<<" --> Enter second distance:"<<endl;
    d2.getdistance();
    result = adddistance(d1,d2);
    cout<<"The sum of these distances is ";
    result.showdistance();
    return 0;
}
```

36. "this" pointer first program.

```
#include<iostream>
using namespace std;
class Box
{
    private:
    int l,b;
    public:
    void displayaddress()
    {
```

```
        cout<<"The address of b uing this keyword : "<<this<<endl;
    }

};
int main()
{
    Box b;
    b.displayaddress();
    cout<<"The address of b using & operator : "<<&b<<endl;
/*  Both above 2 line print the same address.so we can say that 'this' is a pointer that
    stores the address of object that invokes the member function in which this keyword is
    present.
*/
    return 0;
}
```

37. "this" pointer second program.

```
#include<iostream>
using namespace std;
class Salary
{
 private:
    int basic;
 public:
    void getsalary(int);
    void displaysalary();
};
void Salary::getsalary(int b)
{
    // basic = b;
    this -> basic = b;
   // both the above 2 statements are same.

}
void Salary::displaysalary()
{
    cout<<"salary without using this keyword : "<<basic<<endl;
    cout<<"salary using this keyword : "<<this->basic<<endl;
}
int main()
{
    Salary s;
    s.getsalary(20000);
    s.displaysalary();
    return 0;
}
```

38. "this" pointer third program.

```
#include <iostream>
using namespace std;
class Experiment
{
private:
    int x;

public:
```

```
    void test(int x)
// intensionally created same parameter name as the private members.
    {
        this->x = x;
        this->x++;
        cout << "The value of x outside test function is : " << this->x << endl;
        cout << "The value of x inside test function is : " << x << endl;
    }
};

int main()
{
    Experiment e;
    e.test(50);
    return 0;
}
```

39. Friend class demonstration program.

```
#include <iostream>
using namespace std;
class ABC
{
private:
    int a, b;

public:
    void getdata()
    {
        cout << "Enter value of 2 numbers:" << endl;
        cout << "a : ";
        cin >> a;
        cout << "b : ";
        cin >> b;
    }
    friend class XYZ;
};


class XYZ
{
public:
    void display(ABC c)
    {
        cout << "Displaying value from friend class:" << endl;
        cout << "a : " << c.a << endl;
        cout << "b : " << c.b << endl;
    }
};
int main()
{
    ABC a;
    a.getdata();
    XYZ x;
    x.display(a);
    return 0;
}
```

40. Program to show Default constructor.

```
#include <iostream>
using namespace std;
class test
{
public:
    test()
    {
        cout << "Hello, i am default constructor" << endl;
    }
};
int main()
{
    test t;
    return 0;
}
```

41. Program to show Parameterized constructor.

```
#include <iostream>
using namespace std;
class test
{
public:
    test(int x)
    {
        cout << "Hello, i am parameterized constructor" << endl;
        cout << "The value of x is " << x << endl;
    }
};
int main()
{
    test t = test(10);
    // parameterized constructor called explicitly.
    cout << endl;
    test d(60);
    // parameterized constructor called implicitely.
    return 0;
}
```

42. Program to show Copy constructor.

```
#include <iostream>
using namespace std;
class test
{
    int x, y;

public:
```

```
    test(int a)
    {
        cout << "Parameterized constructor is called !" << endl;
        x = a;
        cout << "The value of x is " << x << endl;
    }
    test(test &t)
    {
        cout << "copy constructor is called, copying values..." << endl;
        y = t.x;
        cout << "The value of y is " << y << endl;
    }
};
int main()
{
    cout << " --> this output is of callling copy constructor using assignment operator:";
    cout << endl;
    test t1(5);
    test t2 = t1;
// using assignment operator for calling copy constructor.
    // or,
    cout << endl;
// for space between the outputs, output looks good to!
    cout << " --> this output is of calling copy constructor using object as argunent:";
    cout << endl;
    test t3(10);
    test t4(t3);
// using object as argument for calling copy constructor.

    return 0;
}
```

43. Program to show all three constructors.

```
#include <iostream>
using namespace std;
class code
{
private:
    int id;

public:
    code() {}
    code(int a)
    {
        id = a;
    }
    code(code &x)
    {
        id = x.id;
    }
    int display()
    {
        return id;
    }
};
int main()
{
    code p1(50);  // call to parameterized constructor.
    code p2(p1);  // call to copy constructor using object as argument.
    code p3 = p2; // call to copy constructor using '='  operator.
    cout << "Displaying the result:" << endl;
```

```
    cout << "Code p1 id = " << p1.display() << endl;
    cout << "Code p2 id = " << p2.display() << endl;
    cout << "Code p3 id = " << p3.display() << endl;
    return 0;
}
```

Output:

```
Displaying the result:
Code p1 id = 50
Code p2 id = 50
Code p3 id = 50
```

44. Constructor overloading demonstration program.

```
#include <iostream>
using namespace std;
class complex
{
private:
    float x, y;

public:
    complex() {}
    complex(float a)
    {
        x = y = a;
    }
    complex(float real, float img)
    {
        x = real;
        y = img;
    }
    friend complex sum(complex &, complex &);
    friend void show(complex &);
};
complex sum(complex &c1, complex &c2)
{
    complex temp;
    temp.x = c1.x + c2.x;
    temp.y = c1.y + c2.y;
    return temp;
}
void show(complex &c)
{
    if (c.y < 0)
        cout << c.x << " - " << (-1) * c.y << "i" << endl;
    else
        cout << c.x << " + " << c.y << "i" << endl;
}
int main()
{
    complex A(2.7, -3.5); // <-- change the numbers inside // brackets if you want to do
                                                // experiment with code.
    complex B(1.6);       //  <-- change the numbers inside // brackets if you want to do
                                                // experiment with code.
    complex result;
    result = sum(A, B);

    cout << " A = ";
    show(A);
```

```
    cout << " B = ";
    show(B);
    cout << " Sum of these complex numbers = ";
    show(result);
    return 0;
}
```

Output:

```
A = 2.7 - 3.5i
B = 1.6 + 1.6i
Sum of these complex numbers = 4.3 - 1.9i
```

45. Destruction demonstration program.

```
#include <iostream>
using namespace std;
class Box
{
public:
    Box()
    {
        cout << "Hello, i am default constructor !!" << endl;
    }
    ~Box()
    {
        cout << "Hello, i am Destructor !!" << endl;
    }
};
int main()
{
    Box b1;
    return 0;
}
```

Output:

```
Hello, i am default constructor !!
Hello, i am Destructor !!
```

46. Enumeration: "enum" keyword demonstration first program.

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

/* Here the values of variables start from 0 and then increases by 1 for next variable. */
enum levels
{
    low,    // low value is set to 0;
    medium, // medium value is set to 1;
    high    // high value is set to 1;
};

/* here the values start with 1(because 1 is given), and then increases by 1 for next
variable. */
enum days
{
    sunday = 1,
    monday,
    tuesday,
    wednesday,
```

```
    thursday,
    friday,
    saturday
};

int main()
{
    enum levels l = low;
    cout << "The low value is: " << l << endl;
    l = medium;
    cout << "The medium value is: " << l << endl;
    l = high;
    cout << "The high value is: " << l << endl << endl;

    enum days d;
     d = sunday;
    cout << "the day number of sunday is: " << d << endl;
    d = monday;
    cout << "the day number of monday is: " << d << endl;
    d = wednesday;
    cout << "the day number of wednesday is: " << d << endl;
    d = saturday;
    cout << "the day number of saturday is: " << d << endl << endl;
    return 0;
}
```

Output:

```
The low value is: 0
The medium value is: 1
The high value is: 2

the day number of sunday is: 1
the day number of monday is: 2
the day number of wednesday is: 4
the day number of saturday is: 7
```

47. Enumeration: "enum" keyword demonstration Second program.

```
#include <iostream>
using namespace std;
int main()
{
    enum color
    {
        Red,
        Green,
        Blue,
        White = 5,
        Orange,
        Purple = 9,
        Pink,
        Gray

/*  we can assign values if we want, if values not assigned then values are assigned from 0.
But, once a value is assigned the next variable is assigned (value + 1) value in it and
process continues if no any other values are assigned. But if again new value is assignd to
other variables, conting starts from that assigned value for later varibles. you can check
output of this program to understand more about this.  */

    };
```

```
    color c;
    c = Red;
    cout << "The value of Red is :" << Red << endl;
    c = Green;
    cout << "The value of Green is :" << Green << endl;
    c = Blue;
    cout << "The value of Blue is :" << Blue << endl;
    c = White;
    cout << "The value of White is :" << White << endl;
    c = Orange;
    cout << "The value of Orange is :" << Orange << endl;
    c = Purple;
    cout << "The value of Purple is :" << Purple << endl;
    c = Pink;
    cout << "The value of Pink is :" << Pink << endl;
    c = Gray;
    cout << "The value of Gray is :" << Gray << endl;

  return 0;
}
```

Output:

```
The value of Red is :0
The value of Green is :1
The value of Blue is :2
The value of White is :5
The value of Orange is :6
The value of Purple is :9
The value of Pink is :10
The value of Gray is :11
```

48. Dynamic memory allocation for single integer value.

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

int main()
{
    int *ptr;
    ptr = new int;
    cout << "Enter any number" << endl;
    cin >> *ptr;
    cout << "The value is " << *ptr << endl;
    cout << "Size of ptr is = " << sizeof(ptr) << " bytes" << endl;        // 8 bytes
    delete ptr;
    return 0;
}
```

Output:

```
Enter any number
89
The value is 89
Size of ptr is = 8 bytes
```

49. Dynamic memory allocation for an array integer values.

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

int main()
{
    int *ptr;
    ptr = new int[3];
    cout << "The size of pointer is " << sizeof(ptr) << endl;

    cout << "Enter 3 values for the array" << endl;
    for (int i = 0; i < 3; i++)
    {
        cin >> *ptr;
        ptr = ptr + 1; // ptr ++ ;
    }
    ptr = ptr - 3;
/*
-3 subtracted because, in above loop ptr pointer is incremented 3 times and for accessing
the values from first it is necessary to reset the pointer varible ptr.
*/
    for (int i = 0; i < 3; i++)
    {
        cout << "The value at " << i + 1 << " position is " << *ptr << endl;
        ptr++;
    }
    delete[] ptr;
    return 0;
}
```

Output:

```
The size of pointer is 8
Enter 3 values for the array
43 54 78
The value at 1 position is 43
The value at 2 position is 54
The value at 3 position is 78
```

50. Dynamic memory allocation program from note.

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

int main()
{
    int n, i;
    float *p, sum = 0, avg;
    cout << "How many students are there ?" << endl;
    cin >> n;
    cout << "Enter the total marks of each students" << endl;
    p = new float[n];
    for (i = 0; i < n; i++)
    {
        cin >> *(p + i);
        sum = sum + *(p + i);
    }
    avg = sum / n;
    cout << "The marks you entered are:" << endl;
    for (i = 0; i < n; i++)
    {
```

```
        cout << setw(4) << *(p + i);
    }
    cout << endl;
    cout << "The average marks of students are : ";
    cout << endl << avg;
    delete[] p;
    return 0;
}
```

Output:

```
How many students are there ?
5
Enter the total marks of each students
560 540 589 530 589
The marks you entered are:
 560 540 589 530 589
The average marks of students are :
561.6
```

51. Dynamic memory allocation of class object array using arrow (->) operator.

```
#include <iostream>
using namespace std;
class shop
{
private:
    int id;
    float price;

public:
    void setdata(int x, float y)
    {
        id = x;
        price = y;
    }
    void displaydata()
    {
        cout << "ID Number of this item is " << id << endl;
        cout << "Price of this item is " << price << endl;
    }
};
int main()
{
    int size;
    cout << "Enter size of items to create" << endl;
    cin >> size;
    shop *ptr = new shop[size];
    shop *temp_ptr = ptr;
    int id, i;
    float price;
    for (i = 0; i < size; i++)
    {
        cout << "Enter the Id number of " << i + 1 << " item :";
        cin >> id;
        cout << "Enter the price of " << i + 1 << " item :";
        cin >> price;
        ptr->setdata(id, price);
```

```
        ptr++;
    }
    ptr = temp_ptr;
    for (i = 0; i < size; i++)
    {
        ptr->displaydata();
        ptr++;
    }
    return 0;
}
```

Output:

```
Enter size of items to create
2
Enter the Id number of 1 item :1001
Enter the price of 1 item :500
Enter the Id number of 2 item :1002
Enter the price of 2 item :600
ID Number of this item is 1001
Price of this item is 500
ID Number of this item is 1002
Price of this item is 600
```

52. Unary Operator (-) overloading using class function.

```
#include <iostream>
using namespace std;
class space
{
private:
    int x, y, z;

public:
    space(int a, int b, int c)
    {
        x = a;
        y = b;
        z = c;
    }
    void display()
    {
        cout << "X : " << x << endl;
        cout << "Y : " << y << endl;
        cout << "Z : " << z << endl;
    }
    void operator-()
    {
        x = -x;
        y = -y;
        z = -z;
    }
};

int main()
{
    space x(5, 6, 7);
    cout << "Before overloading the values are:" << endl;
    x.display();
    cout << endl;
```

```
    -x;           // call to operator - () function.
    cout << "After overloading the values are:" << endl;
    x.display();
    return 0;
}
```

Output:

```
Before overloading the values are:
X : 5
Y : 6
Z : 7

After overloading the values are:
X : -5
Y : -6
Z : -7
```

53. Unary Operator (-) overloading using friend function.

```
#include <iostream>
using namespace std;
class space
{
private:
    int x, y, z;

public:
    space(int a, int b, int c)
    {
        x = a;
        y = b;
        z = c;
    }
    void display()
    {
        cout << "X : " << x << endl;
        cout << "Y : " << y << endl;
        cout << "Z : " << z << endl;
    }

    friend void operator-(space &s);
};
void operator-(space &s)
{
    s.x = -s.x;
    s.y = -s.y;
    s.z = -s.z;
}

int main()
{
    space x(5, 6, 7);
    cout << "Before overloading the values are:" << endl;
    x.display();
    cout << endl;
```

```
    -x;
    cout << "After overloading the values are:" << endl;
    x.display();
    return 0;
}
```

54. Binary Operator (+) overloading using class function.

```
#include <iostream>
using namespace std;
class complex
{
private:
    int real, img;

public:
    complex() // default constructor
    {
        real = 0;
        img = 0;
    }
    complex(int a, int b) // constructor with parameter
    {
        real = a;
        img = b;
    }
    void display()
    {
        if (img < 0)
            cout << real << " - " << (-1) * img << "i" << endl;
        else
            cout << real << " + " << img << "i" << endl;
    }
    complex operator+(complex);
};
complex complex::operator+(complex c)
{
    complex temp;
    temp.real = real + c.real;
    temp.img = img + c.img;
    return temp;
}
int main()
{
    complex c1(4, -8);
    complex c2(5, 7);
    cout << "First complex number is : ";
    c1.display();
    cout << "Second complex number is : ";
    c2.display();
    cout << endl;
    complex result;
    result = c1 + c2;
    cout << "The sum of above complex numbers is: ";
    result.display();
    return 0;
}
```

Output:

```
First complex number is : 4 - 8i
Second somplex number is : 5 + 7i

The sum of above complex numbers is: 9 - 1i
```

55. Binary Operator (+) overloading using friend function.

```
#include <iostream>
using namespace std;
class complex
{
private:
    int real, img;

public:
    complex()               // default constructor
    {
        real = 0;
        img = 0;
    }
    complex(int a, int b)             // constructor with parameter
    {
        real = a;
        img = b;
    }
    void display()
    {
        if (img < 0)
            cout << real << " - " << (-1) * img << "i" << endl;
        else
            cout << real << " + " << img << "i" << endl;
    }
    friend complex operator+(complex &, complex &);       // friend function declared
};

complex operator+(complex &c1, complex &c2)              // friend funciton defined.
{
    complex temp;
    temp.real = c1.real + c2.real;
    temp.img = c1.img + c2.img;
    return temp;
}

int main()
{
    complex c1(4, -8);                    // call to parameterized constructor.
    complex c2(5, 7);                     // call to parameterized constructor.
    c1.display();
    c2.display();
    cout << endl;

    complex result;                       // call to default constructor.
    result = c1 + c2;
    cout << "The sum of above complex numbers is: ";
    result.display();
    return 0;
}
```

56. Binary operator (>) overloading using class function.

```cpp
#include <iostream>
using namespace std;
class MyClass
{
private:
    int value;

public:
    MyClass()
    {
        cout << "Enter the value : " << endl;
        cin >> value;
    }
    // Overloading the ">" operator
    bool operator>(MyClass other)
    {
        return (value > other.value);
    }
};

int main()
{

    MyClass obj1;
    MyClass obj2;

    if (obj1 > obj2)
        cout << "obj1  is greater than obj2" << endl;
    else
        cout << "obj1 is not greater than obj2" << endl;

    return 0;
}
```

Output:

```
Enter the value :
745
Enter the value :
345
obj1  is greater than obj2
```

57. Basic to Basic data type conversion first program.

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

int main()
{
    int a = 89;
    char c = 'A';
    float b = 90.6f;
    /*
    By default c++ assumes float variable as double variable
    so if we want to use float variable then need to use
    'F' or 'f' letter after tht value
    */
```

```
    cout << "value of 'A' in integer = " << int(c) << endl;              // value = 65
    /* converts the char data into int data*/
    cout << "c = " << c << endl;

    cout << "value of '90.6f' in character = " << char(b) << endl;      // value = Z
    /* first converts the float into int type and then gives the
       equivalent character value to the integer */

    cout << "value of '90.6f' in integer = " << int(b) << endl;          // value = 90
    /* this converts the float type data into int type */

    cout << "value of '89' in character = " << char(a) << endl;          // value = Y

    return 0;
}
```

Output:

```
value of 'A' in integer = 65
c = A
value of '90.6f' in character = Z
value of '90.6f' in integer = 90
value of '89' in character = Y
```

58. Basic to Basic data type conversion second program.

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

int main()
{
    int a = 5, x, y;
    float b = 7.98f, w;
    char m = 'B', z;
    x = b;
    y = m;
    z = a;
    w = m;

    cout << "Displaying the result of conversion !!" << endl;
    cout << "w = " << w << endl;                        // W = 66
    cout << "x = " << x << endl;                        // x = 7
    cout << "y = " << y << endl;                        // y = 66
    cout << "z = " << z << endl;                        // z = ♣

    return 0;
}
```

Output:

```
Displaying the result of conversion !!
w = 66
x = 7
y = 66
z = ♣
```

59. Basic to Class type conversion using constructor. (for time)

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

class Time
{
    int hours, minutes;

public:
    Time() {}

    Time(int t)
    {
        hours = t / 60;
        minutes = t % 60;
    }
    void display()
    {
        cout << "The time is " << hours << " Hours and " <<
       minutes << " Minutes" << endl;
    }
};
int main()
{
    int duration;
    cout << "This is the example of Basic to Class type conversion" << endl;
    cout << "Enter Time in duration of minutes :";
    cin >> duration;

    Time t;
    t = Time(duration);
    t.display();
    return 0;
}
```

Output:

```
This is the example of Basic to Class type conversion
Enter Time in duration of minutes :129
The time is 2 Hours and 9 Minutes
```

60. Basic to Class type conversion by overloading "=" operator. (for time)

```
#include <iostream>
using namespace std;
class Time
{
    int hours, minutes;

public:
    void operator=(int t)
    {
        hours = t / 60;
        minutes = t % 60;
    }
    void display()
    {
        cout << "The time is " << hours << " Hours " << minutes << " minutes" << endl;
    }
};
```

```
int main()
{
    int time;
    cout << "Basic to class type conversion" << endl;
    cout << "Enter time in minutes : ";
    cin >> time;
    Time t;
    t = time;
    // or,
    // t.operator=(time);
    t.display();

    return 0;
}
```

Output:

```
Basic to class type conversion
Enter time in minutes : 135
The time is 2 Hours 15 minutes
```

61. Basic to Class type conversion using constructor. (for distance)

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

class Distance
{
    int cm, meter;
    float km;

public:
    Distance() {}
    Distance(int d)
    {
        meter = d;
        cm = d * 100;
        km = d / 1000.0f;

/* dividing by float number, forces '/' operator to give result in
  floating point. if we divide by '1000' then we will get integer value in result and it
  will not be precise...  */

    }
    void display()
    {
        cout << "The distance is:" << endl;
        cout << "In Km : " << km << " km" << endl;
        cout << "In Meter : " << meter << " meters" << endl;
        cout << "In cm : " << cm << " cm" << endl;
    }
};

int main()
{
    int length;
    cout << "This is the Basic to class conversion" << endl;
    cout << "Enter length in meters : ";
    cin >> length;

    Distance d;
```

```
    d = Distance(length); // constructor called.
    d.display();
    return 0;
}
```

Output:

```
This is the Basic to class conversion
Enter length in meters : 1439
The distance is:
In Km : 1.439 km
In Meter : 1439 meters
In cm : 143900 cm
```

62. Basic to Class type conversion by overloading "=" operator. (for distance)

```
#include <iostream>
using namespace std;
class Distance
{
    int cm, meter;
    float km;

public:
    void operator=(int d)
    {
        km = d / 1000.0f;
        meter = d;
        cm = d * 100;
    }
    void display()
    {
        cout << "The distance in KM is : " << km << endl;
        cout << "The distance in Meter is : " << meter << endl;
        cout << "The distance in CM is : " << cm << endl;
    }
};
int main()
{
    int length;
    cout << "This is basic to class type conversion using operator overloading." << endl;
    cout << "Enter Length in meters : ";
    cin >> length;

    Distance d;
    d = length; // operator '=' is called.
    d.display();
    return 0;
}
```

Output:

```
This is basic to class type conversion using operator overloading.
Enter Length in meters : 4527
The distance in KM is : 4.527
The distance in Meter is : 4527
The distance in CM is : 452700
```

63. Class to Basic type conversion. (for time)

```
#include <iostream>
using namespace std;
class Time
{
    int hours, minutes;

public:
    Time(int h, int m)          // parameterized constructor...
    {
        cout << "Constructor with 2 parameter is called" << endl;
        hours = h;
        minutes = m;
    }
    operator int()              // conversion function...
    {
        cout << "Class to basic type conversion in progress..." << endl;
        return ((hours) * 60 + minutes);
    }
    ~Time()
    {
        cout << "Destructor is called !" << endl;
    }
};
int main()
{
    int duration, hr, min;
    cout << "Enter Time:" << endl;
    cout << "Hours : ";
    cin >> hr;
    cout << "Minutes : ";
    cin >> min;
    Time t(hr, min);
    duration = t;
    cout << "The duration is " << duration << " in minutes" << endl;
    cout << "Class to basic type conversion completed !" << endl;
    return 0;
}
```

Output:

```
Enter Time:
Hours : 1
Minutes : 29
Constructor with 2 parameter is called
Class to basic type conversion in progress...
The duration is 89 in minutes
Class to basic type conversion completed !
Destructor is called !
```

64. Class to Basic type conversion. (for distance)

```
#include <iostream>
using namespace std;
class Length
{
    int meter, cm;
```

```
public:
    Length(int m)                   // parameterized constructor
    {
        cout << "Parameterized constructor is called !" << endl;
        meter = m;
    }
    operator int()                 // conversion function
    {
        cout << "class to basic type conversion in progress ..." << endl;
/* this is just a message i want to print....you can write anything here.. */
        return (meter * 100);
    }
    ~Length()                      // destructor
    {
        cout << "Destructor is called !" << endl;
    }
};
int main()
{
    int meter, cm;
    cout << "enter length in meter : ";
    cin >> meter;
    Length l(meter);         // call to parameterized constructor
    cm = l;                     // call to conversion function...
    cout << "The length in meter is " << meter << " meter and in cm is " << cm << " cm";
    cout << endl;
    cout << "Class to basic type conversion complete !" << endl;
    return 0;
}
```

Output:

```
enter length in meter : 269
Parameterized constructor is called !
class to basic type conversion in progress ...
The length in meter is 269 meter and in cm is 26900 cm
Class to basic type conversion complete !
Destructor is called !
```

65. Class to Class type conversion using conversion function in destination class.

```
#include <iostream>
#include <cmath>              // headerfile for using cos(), sin() math functions
using namespace std;
class Polar                   // source class
{
private:
    float radius, angle;

public:
    Polar()
    {
        radius = 0;
        angle = 0;
    }
    Polar(float r, float a)
    {
        radius = r;
        angle = a;
    }
```

```
    void display()
    {
        cout << "(" << radius << ", " << angle << ")" << endl;
    }
    float getr()
    {
        return radius;
    }
    float geta()
    {
        return angle;
    }
};

class Rectangular                    // destination class
{
private:
    float xco, yco;

public:
    Rectangular()
    {
        xco = 0;
        yco = 0;
    }
    Rectangular(float x, float y)
    {
        xco = x;
        yco = y;
    }
    Rectangular(Polar p)                // conversion function
    {
        float r, a;
        r = p.getr();
        a = p.geta();
        xco = r * cos(a);
        yco = r * sin(a);
    }
    void display()
    {
        cout << "(" << xco << ", " << yco << ")" << endl;
    }
};
int main()
{
    Rectangular r;
    Polar p(10.0, 0.7853);
    r = p;
    cout << "The polar coordinates are ";
    p.display();
    cout << "The rectangular coordinates are ";
    r.display();
    return 0;
}
```

Output:

```
The polar coordinates are (10, 0.7853)
The rectangular coordinates are (7.07176, 7.07037)
```

66. Class to Class type conversion using conversion function in source class.

```
#include <iostream>
#include <cmath>              // headerfile for using cos(), sin() math functions
using namespace std;
class Rectangular                     // Destination Class
{
private:
    float xco, yco;

public:
    Rectangular()
    {
        xco = 0;
        yco = 0;
    }
    Rectangular(float x, float y)
    {
        xco = x;
        yco = y;
    }

    void display()
    {
        cout << "The rectangular coordinates are (" << xco << ", " << yco << ")" << endl;
    }
};

class Polar                                 // Source Class
{
private:
    float radius, angle;

public:
    Polar()
    {
        radius = 0;
        angle = 0;
    }
    Polar(float r, float a)
    {
        radius = r;
        angle = a;
    }
    void display()
    {
        cout << "The polar coordinates are (" << radius << ", " << angle << ")" << endl;
    }
    operator Rectangular()
    {
        float x, y;
        x = radius * cos(angle);
        y = radius * sin(angle);
        return Rectangular(x, y);
    }
};
int main()
{
    Rectangular r;
    Polar p(70.0f, 0.7853f);
    r = p;                                  // call to operator function.
```

```
    p.display();
    r.display();
    return 0;
}
```

Output:

```
The polar coordinates are (70, 0.7853)
The rectangular coordinates are (49.5023, 49.4926)
```

67. Access modes in Inheritance.

```
/*
 Inheritance.
 Access specifier are the access restictions imposed during
 the derivation of differnt subclasses from the base class.
 --> There are 3 types of access specifiers. They are:
*/

/*
    1) private mode:
        Every thing from base class except the private section's
        data/functions are derived into the private section of
        derived class.
    2) protected mode:
        Everything from base class except the private section's
        data/functions are derived into the protected section of
        derived class.
    3) public mode:
        Everything from base class except the private section's
        data/functions are derived into the same section as they
        are in the base class. meaning, the protected data member
        of base class will be inside the protected section of
        derived class, and public members of base class will be
        in public section of derived class.
*/
```

68. Private access mode demonstration.

```
#include <iostream>
using namespace std;
class Base
{
private:
    int x;

protected:
    int y;

public:
    int z;
    Base()
    {
        x = 10;
        y = 20;
        z = 30;
    }
};
class derived1 : private Base
{
```

```
public:
    void showdata()
    {
        cout << "Here, x is not accessible here in derived class1 !" << endl;
        cout << "y = " << y << endl;
        cout << "z = " << z << endl;
    }
};
class derived2 : private Base
{
public:
    void showdata()
    {
        cout << "x, y, & z are not accessible here in derived 2 class !" << endl;
    }
};

int main()
{
    derived1 d1;
    derived2 d2;
    cout << "Data of derived1 class:" << endl;
    d1.showdata();
    cout << "Data of derived class:" << endl;
    d2.showdata();
    return 0;
}
```

69. Protected access mode demonstration.

```
#include <iostream>
using namespace std;
class Base
{
private:
    int x;

protected:
    int y;

public:
    int z;
    Base()
    {
        x = 10;
        y = 20;
        z = 30;
    }
};
class derived1 : protected Base
{
public:
    void showdata()
    {
        cout << "Here, x is not accessible here in derived class1 !" << endl;
        cout << "y = " << y << endl;
        cout << "z = " << z << endl;
// y & z are derived into the protected section of derived1 class
    }
};
```

```
class derived2 : protected Base
{
public:
    void showdata()
    {
        cout << "Here, x is not accessible here in derived class1 !" << endl;
        cout << "y = " << y << endl;
        cout << "z = " << z << endl;
// y & z are derived into the protected section of derived1 class.. same output as above
    };
};
int main()
{
    derived1 d1;
    derived2 d2;
    cout << "Data of derived1 class:" << endl;
    d1.showdata();
    cout << "Data of derived class:" << endl;
    d2.showdata();
    return 0;
}
```

70. Public access mode demonstration.

```
#include <iostream>
using namespace std;
class Base
{
private:
    int x;

protected:
    int y;

public:
    int z;
    Base()
    {
        x = 10;
        y = 20;
        z = 30;
    }
};
class derived1 : public Base
{
public:
    void showdata()
    {
        cout << "Here, x is not accessible here in derived1 class !" << endl;
        cout << "y = " << y << endl;
        cout << "z = " << z << endl;
// y is derived into protected & z is derived into public section of derived1 class.
    }
};
class derived2 : public Base
{
public:
    void showdata()
    {
        cout << "Here, x is not accessible here in derived2 class !" << endl;
        cout << "y = " << y << endl;
```

```
        cout << "z = " << z << endl;
// y is derived into protected & z is derived into public section of derived1 class.
        // same output as above.
    };
};
int main()
{
    derived1 d1;
    derived2 d2;
    cout << "Data of derived1 class:" << endl;
    d1.showdata();
    cout << "Data of derived2 class:" << endl;
    d2.showdata();
    return 0;
}
```

71. Single inheritance.

```
#include <iostream>
using namespace std;
class Base                    // Base class
{
protected:
    int x = 10;

public:
    void displayBase()
    {
        cout << "Hello, I am Base class and value of x = " << x << endl;
    }
};
class Derived : public Base            // Derived class
{
public:
    void displayDerived()
    {
        cout << "Hello, I am Derived class and value of x = " << x << endl;
    }
};
int main()
{
    Derived d;
    d.displayBase();
    d.displayDerived();
    return 0;
}
```

72. Multi-level Inheritance first program.

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

class A                                // Base class
{
protected:
    int x = 10;

public:
    void displayx()
    {
```

```
        cout << "Hello, I am Base class A and value of x = " << x << endl;
    }
};

class B : public A                    // Intermediate derived class
{
public:
    void displayy()
    {
        cout << "Hello, I am Intermediate Derived class B and value of x = " << x << endl;
    }
};
class C : public B                          // Derived class
{
public:
    void displayz()
    {
        cout << "Hello, I am Derived class C and value of x = " << x << endl;
    }
};
int main()
{
    C c;
    c.displayx();
    c.displayy();
    c.displayz();
    return 0;
}
```

73. Multi-level Inheritance second program.

```
#include <iostream>
using namespace std;
class A                                     // Base class
{
protected:
    int x;

public:
    void getx()
    {
        cout << "Enter value of x : ";
        cin >> x;
    }
};
class B : public A                          // Intermediate derived class
{
protected:
    int y;

public:
    void gety()
    {
        cout << "Enter the value of y : ";
        cin >> y;
    }
};
class C : public B                          // derived class
{
private:
    int z;
```

```
public:
    void getz()
    {
        cout << "Enter the balue of z : ";
        cin >> z;
    }
    void displaysum()
    {
        cout << "The sum of x, y, z is " << (x + y + z) << endl;
    }
};
int main()
{
    C c1;
    c1.getx();
    c1.gety();
    c1.getz();
    c1.displaysum();

    return 0;
}
```

74. Multiple Inheritance.

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

class A                                    // Base class
{
protected:
    int x;

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

class B                                   // Base class
{
protected:
    int y;

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

class C : public A, public B             // derived class
{
public:
    void displaysum()
    {
        cout << endl;
        cout << " x : " << x << endl;
```

```
        cout << " y : " << y << endl;
        cout << "Sum of x and y is : " << (x + y) << endl;
    }
};

int main()
{
    C c1;
    c1.getx();
    c1.gety();
    c1.displaysum();
    return 0;
}
```

75. Hierarchical Inheritance.

```
#include <iostream>
using namespace std;
class A                                    // Base class
{
protected:
    int x, y;

public:
    void getdata()
    {
        cout << "Enter x : ";
        cin >> x;
        cout << "Enter y : ";
        cin >> y;
    }
};
class B : public A                         // Derived class
{
public:
    void displaysum()
    {
        cout << "Sum of x and y is " << (x + y) << endl;
    }
};
class C : public A                         // Derived class
{
public:
    void displayproduct()
    {
        cout << "Product of x and y is " << (x * y) << endl;
    }
};
int main()
{

    B b;
    cout << "For class B:" << endl;
    b.getdata();
    b.displaysum();
    cout << endl;
    C c;
    cout << "For class C:" << endl;
    c.getdata();
    c.displayproduct();
```

```
    return 0;
}
```

76. Hybrid Inheritance.

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

class Student               // Base class
{
protected:
    char name[25];
    int roll;

public:
    void getintro()
    {
        cout << "Enter name : ";
        cin.get(name, sizeof(name));
        cout << "Enter roll no : ";
        cin >> roll;
    }
    void showintro()
    {
        cout << "Name : " << name << endl;
        cout << "Roll no : " << roll << endl;
    }
};
class Marks : public Student           // Intermediate derived class
{
protected:
    int marks;

public:
    void getmarks()
    {
        cout << "Enter total marks : ";
        cin >> marks;
    }
    void showmarks()
    {
        cout << "Total marks : " << marks << endl;
    }
};
class Score                            // Base class
{
protected:
    int score;

public:
    void getscore()
    {
        cout << "Enter score : ";
        cin >> score;
    }
    void showscore()
    {
        cout << "Score = " << score << endl;
    }
};
```

```
class Result : public Marks, public Score          // Derived class
{
private:
    int total_score;

public:
    void display()
    {
        total_score = marks + score;
        showintro();
        showmarks();
        showscore();
        cout << "The total score is " << total_score << endl;
    }
};
int main()
{
    Result r;
    r.getintro();
    r.getmarks();
    r.getscore();
    cout << endl;
    cout << "Now, Displaying the information as." << endl;
    r.display();
    return 0;
}
```

77. Constructor in derived class first program.

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

class base1
{
protected:
    int x;

public:
    base1(int m)
    {
        x = m;
        cout << "Base1 constructor is invoked !!" << endl;
        cout << "The value of x is " << x << endl;
        cout << endl;
    }
};
class base2
{
protected:
    int y;

public:
    base2(int n)
    {
        y = n;
        cout << "Base2 constructor is invoked !!" << endl;
        cout << "The value of y is " << y << endl;
        cout << endl;
    }
};
```

```
class derived : public base1, public base2
{
private:
    int i, j;

public:
    derived(int a, int b, int c, int d) : base1(a), base2(b)
    {
        i = c;
        j = d;
        cout << "Derived class constructor is invoked !!" << endl;
        cout << "The value of i is " << i << endl;
        cout << "The value of j is " << j << endl;
        cout << endl;
    }
};
int main()
{
    derived d(1, 2, 3, 4);

    return 0;
}
```

Output:

```
Base1 constructor is invoked !!
The value of x is 1

Base2 constructor is invoked !!
The value of y is 2

Derived class constructor is invoked !!
The value of i is 3
The value of j is 4
```

78. Constructor in derived class second program.

```
#include <iostream>
using namespace std;
class Alpha
{
    int x;

public:
    Alpha(int i)
    {
        x = i;
        cout << "Hello, I'm alpha." << endl;
    }
    void show_x()
    {
        cout << "x = " << x << endl;
    }
};
class Beta
{
    float y, z;

public:
```

```
    Beta(float j, float k)
    {
        y = j;
        z = k;
        cout << "Hello, I'm beta." << endl;
    }
    void show_yz()
    {
        cout << "y = " << y << endl;
        cout << "z = " << z << endl;
    }
};

class Gamma : public Alpha, public Beta
{
    int m, n;
public:
  Gamma(int a, float b, float c, int d, int e) : Alpha(a),
    Beta(b, c)
      {
          m = d;
          n = e;
          cout << "Hello, I'm Gamma." << endl;
      }
      void show_mn()
      {
          cout << "m = " << m << endl;
          cout << "n = " << n << endl;
      }
};

int main()
{
    Gamma g(5, 10.6, 2.6, 6, 9);
    cout << endl;
    g.show_x();
    cout << endl;
    g.show_yz();
    cout << endl;
    g.show_mn();

    return 0;
}
```

Output:

```
Hello, I'm alpha.
Hello, I'm beta.
Hello, I'm Gamma.

x = 5

y = 10.6
z = 2.6

m = 6
n = 9
```

79. Virtual Base class demonstration program.

```
#include <iostream>
using namespace std;
class Student
{
protected:
    char name[25];
    int roll;

public:
    void getintro()
    {
        cout << "Enter name : ";
        cin.get(name, sizeof(name));
        cout << "Enter roll number : ";
        cin >> roll;
    }
    void displayintro()
    {
        cout << "Name : " << name << endl;
        cout << "Roll number : " << roll << endl;
    }
};
class Marks : virtual public Student
{
protected:
    int marks;

public:
    void getmarks()
    {
        cout << "Enter total marks : ";
        cin >> marks;
    }
    void displaymarks()
    {
        cout << "Total marks : " << marks << endl;
    }
};
class Score : virtual public Student
{
protected:
    int score;

public:
    void getscore()
    {
        cout << "Enter score : ";
        cin >> score;
    }
    void displayscore()
    {
        cout << "Score : " << score << endl;
    }
};
class Result : public Marks, public Score
{
private:
    int total_score;

public:
```

```
    void displayresult()
    {
        total_score = marks + score;
        displayintro();
        displaymarks();
        displayscore();
        cout << "The total score is " << total_score << endl;
    }
};
int main()
{
    Result r;
    r.getintro();
    r.getmarks();
    r.getscore();

    cout << endl;
    cout << "Now displaying the data of student as." << endl;
    r.displayresult();
    return 0;
}
```

80. Ambiguity in inheritance program.

```
#include <iostream>
using namespace std;
class Base1
{
public:
    void display()
    {
        cout << "Hello, i am Base1 class" << endl;
    }
};
class Base2
{
public:
    void display()
    {
        cout << "Hello, i am Base2 class" << endl;
    }
};

class Derived : public Base1, public Base2
{
public:
    void show()
    {
        Base1::display();
        Base2::display();
/*
if we dont use (::) operator with the class name, it will give error. Try the below line to
check if compilation error occurs or not.
*/
        // display();    // <-- uncomment this display() and check if error occurs or not..

    }
};
int main()
{
    Derived d;
```

```
    d.show();
    return 0;
}
```

81. Single inheritance same function name in both base and derived class.

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

class Base
{
public:
    void display()
    {
        cout << "Hello, I am base class !" << endl;
    }
};

class Derived : public Base
{
public:
    void display()
    {
        cout << "Hello, I am derived class !" << endl;
    }
};
int main()
{
    Derived d;
    d.display();
    d.Base::display();
    d.Derived::display();
    return 0;
}
```

82. Function overriding program.

```
#include <iostream>
using namespace std;
class Person
{
protected:
    char name[20];

public:
    void getdata()
    {
        cout << "Enter name : ";
        cin.get(name, sizeof(name));
    }
    void display()
    {
        cout << "Name : " << name << endl;
    }
};
class Student : public Person
{
protected:
    int roll;
```

```
public:
    void getdata()
    {
        cout << "Enter roll number : ";
        cin >> roll;
    }
    void display()
    {
        cout << "Roll number : " << roll << endl;
    }
};
int main()
{
    Student s;
    s.Person::getdata();
    s.getdata();
/*
 In s.person::getdata() function, if you dont use (::) operator, and just called the
 function as s.getdata() it will always call the getdata function of derived class.
*/
    cout << endl;
    cout << "Displaying the information" << endl;
    s.Person::display();
    s.display();
    return 0;
}
```

83. Containership demonstration program.

```
#include <iostream>
using namespace std;
class Student
{
private:
    int roll;
    char name[25];

public:
    void getinfo()
    {
        cout << "Enter name : ";
        cin.get(name, sizeof(name));
        cout << "Enter Roll number : ";
        cin >> roll;
    }
    void displayinfo()
    {
        cout << "Name : " << name << endl;
        cout << "Roll number : " << roll << endl;
    }
};
class Age
{
private:
    int age;

public:
    Student s;
    void getdata()
    {
        s.getinfo();
```

```
        cout << "Enter age : ";
        cin >> age;
    }
    void displaydata()
    {
        s.displayinfo();
        cout << "Age : " << age << endl;
    }
};

int main()
{
    Age a;
    a.getdata();

    cout << endl;
    cout << "Displaying the data of student as." << endl;
    a.displaydata();
    return 0;
}
```

Output:

```
Enter name : Bishal Bhat
Enter Roll number : 9
Enter age : 22
Displaying the data of student as.
Name : Bishal Bhat
Roll number : 9
Age : 22
```

84. Virtual function: Run time polymorphism.

```
/*
Virtual function is a member function whose functionality can be overriden in its derived
classes. we declare virtual functions using virtual keyword in the base class.
when we use virtual function, then we have to acess those function using pointer variable of
base class..
*/

#include <iostream>
using namespace std;
class Base
{

public:
    virtual void show()
    {
        cout << "This is base class (using show function)" << endl;
    }
    void display()
    {
        cout << "This is base class (using display function)" << endl;
    }
};

class derived1 : public Base
{
```

```
public:
    void show()
    {
        cout << "This is derived1 class (using show function)" << endl;
    }
    void display()
    {
        cout << "This is derived1 class (using display function)" << endl;
    }
};

class derived2 : public Base
{
public:
    void show()
    {
        cout << "This is derived2 class (using show function)" << endl;
    }

    void display()
    {
        cout << "This is derived2 class (using display function)" << endl;
    }
};

int main()
{
    Base *ptr, b;
    derived1 d1;
    derived2 d2;

    cout << endl;
    ptr = &d1;
    ptr->show();
    ptr->display();
/*
    even though ptr pointer has adress of d1 object, display()
    of derived1 class will not be called. The display() from
    base class will be called by ptr -> display();
    it is because display() in base class is not declared as
    virtual in base class. where as show() function is declared as
    virtual there.
*/

    cout << endl;
    ptr = &d2;
    ptr->show();
    ptr->display();

/*
    even though ptr pointer has adress of d2 object, display()
    of derived2 class will not be called. The display() from
    base class will be called by ptr -> display();
    it is because display() in base class is not declared as
    virtual in base class. where as show() function is declared as
    virtual there.
*/
```

```
    cout << endl;
    ptr = &b;
    ptr->show();
    ptr->display();

    return 0;
}
```

Output:

```
This is derived1 class (using show function)
This is base class (using display function)

This is derived2 class (using show function)
This is base class (using display function)

This is base class (using show function)
This is base class (using display function)
```

85. Pure virtual function demonstration program.

```
/*
   Pure virtual function is a virtual function which has no body. The pure virtual function is
   declared in base class using the syntax:
                        virtual return_type function_name() = 0;
   Its meaning is defined in the derived classes.
*/

   #include <iostream>
   using namespace std;
   class Base
   {
   public:
       virtual void show() = 0;
   };
   class Derived1 : public Base
   {
   public:
       void show()
       {
           cout << "This is Derived1 class" << endl;
       }
   };
   class Derived2 : public Base
   {
   public:
       void show()
       {
           cout << "This is Derived2 class" << endl;
       }
   };
   int main()
   {
       Base *ptr[2];
       Derived1 d1;
       Derived2 d2;

       ptr[0] = &d1;
       ptr[1] = &d2;
```

```
    ptr[0]->show();              // call to show() of Derived1 class
    ptr[1]->show();              // call to show() of Derived2 class
    return 0;
}
```

Output:

```
This is Derived1 class
This is Derived2 class
```

86. Abstract Class demonstration program.

```
/*
    A class that contains atleat one pure virtual function is
    called an Abstract Class. We cannot create the object of
    Abstract class. We can access the function of abstract class
    using the object of derived class.
    if we dont override the pure virtual function in the derived
    class, then the derived class also becomes abstract class and
    we cannot create the object of derived class.
*/
#include <iostream>
using namespace std;
class A
{
public:
    virtual void show() = 0;
    void display()
    {
        cout << "I am Base class !" << endl;
    }
};

class B : public A
{
public:
    void show()
    {
        cout << "I am derived clas !" << endl;
    }
};

int main()
{
    // A a;
 /*
   Above line will give error, because A is abstract class and we tried
   To create object of it.
 */
    B x;
    x.display();
    x.show();
    return 0;
}
```

Output:

```
I am Base class !
I am derived clas !
```

87. Exception Handling for division with single catch block.

```
#include <iostream>
using namespace std;
int main()
{
    float num1, num2, result;
    cout << "Enter any 2 positive numbers:" << endl;
    cin >> num1 >> num2;
    try
    {
        if (num2 != 0)
        {
            result = num1 / num2;
            cout << "Dividing " << num1 << " by " << num2 << " we get ";
            cout << "the result = " << result << endl;
        }
        else
            throw(num2);
    }


    catch (float n)
    {
        cout << "Divisible by 0 is not allowed";
    }

    return 0;
}
```

Output:

```
Enter any 2 positive numbers:
6 8
Dividing 6 by 8 we get the result = 0.75
```

```
Enter any 2 positive numbers:
7 0
Divisible by 0 is not allowed
```

88. Exception Handling for division with multiple catch block.

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

int main()
{
    char msg[] = "Divide by 0 not allowed !";
    int nums[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int i, counter;
    float n;
/*
   if we create integer variable n, then when dividing, we will get integer number, data will
   be lost when performing division so we create divisor variable 'n' as float.
*/
    try
    {
        cout << "Enter a number for divisor : ";
        cin >> n;
        if (n == 0)
```

```
            throw msg;
        cout << "Enter how many numbers in the arry to divide" << endl;
        cin >> counter;
        if (counter > 10)
            throw counter;
        for (i = 0; i < counter; i++)
        {
            cout << "The result on dividing " << nums[i] << " by " << n << " is ";
             cout << nums[i] / n << endl;
        }
    }
    catch (char str[])
    {
        cout << "EXCEPTION : " << str;
    }
    catch (int index)
    {
        cout << "EXCEPTION : " << index << " Index is out of bound !" << endl;
    }

    return 0;
}
```

Output:

```
Enter a number for divisor : 8
Enter how many numbers in the arry to divide
3
The result on dividing 1 by 8 is 0.125
The result on dividing 2 by 8 is 0.25
The result on dividing 3 by 8 is 0.375
```

```
Enter a number for divisor : 0
EXCEPTION : Divide by 0 not allowed !
```

```
Enter a number for divisor : 8
Enter how many numbers in the arry to divide
11
EXCEPTION : 11 Index is out of bound !
```

89. Namespace demonstration Program.

```
/*
    Namespace is a name given to scope or logical groupinh of variables, functions and classes
    in C++.
          Syntax for namespace is:
                        namespace namespace_name
                        {
                            -- code here --
                        }
*/
```

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

```cpp
namespace Number
{
    int num = 10;
}

int main()
{
    int num = 50;
    cout << "Inside main num = " << num << endl;              // num = 50;
    cout << "Outside main num = " << Number::num << endl;     // num = 10;
    return 0;
}

/*
    Both num have same name but representing different values.
    So, the main purpose of namespace is to prevent the ambiguity
    That may occur when 2 identifiers have same name.
*/
```

Output:

```
Inside main num = 50
Outside main num = 10
```

90. "using" keyword demonstration program.

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

namespace Intro
{
    int roll = 9;
    void display()
    {
        cout << "Name : Bishal Bhat !!" << endl;
    }
}

int main()
{

    using namespace Intro;
    display();
    cout << "Roll number = " << roll << endl;
    return 0;
}

/*
Here, we displayed roll number and name (using function) that are defined in namespace Intro
using "using" keyword.
*/
```

Output:

```
Name : Bishal Bhat !!
Roll number = 9
```

91. Template description.

Template is the frame with generic code. It allows us to write code that are independent of actual data types and one template can be used with various data types seamlessly. Templates are used for creating reusable and flexible code. we can use/create template in C++ by 2 ways. They are:

1> Function Template

Syntax:

```
template < class T >
return_type function_name (parameters list)
{
    ...body with type T...
}
```

2> Class Template

Syntax:

```
template < class T >
class classname
{
    class members with type T
// here is T is variable for data type
}
```

Syntax while declaring an object in main section :

```
class_name < specific_data_type > object_name;
```

92. Function template for displaying any 2 values.

```
#include <iostream>
using namespace std;
template <class T>
void show(T a, T b)
{
    cout << "A : " << a << endl;
    cout << "B : " << b << endl;
}
int main()
{
    int a = 3, b = 4;
    char c = 'M', d = 'N';
    float e = 8.6f, f = 5.3f;
    cout << "Displaying integer value:" << endl;
    show(a, b);
    cout << "Displaying character value:" << endl;
    show(c, d);
    cout << "Displaying Float value:" << endl;
    show(e, f);
    return 0;
}
```

Output:

```
Displaying integer value:
A : 3
B : 4
Displaying character value:
A : M
B : N
Displaying Float value:
A : 8.6
B : 5.3
```

93. Function template for determining the larger value between any 2 values.

```
#include <iostream>
using namespace std;
template <class T>
T larger(T x, T y)
{
    if (x > y)
        return x;
    else
        return y;
}
int main()
{
    int a, b;
    float m, n;
    char p, q;
    cout << "Enter 2 integers : ";
    cin >> a >> b;
    cout << "Enter 2 fractional numbers : ";
    cin >> m >> n;
    cout << "Enter any 2 characters : ";
    cin >> p >> q;
    cout << endl;
    cout << "The larger integer value is " << larger(a, b) << endl;
    cout << "The larger fractional value is " << larger(m, n) << endl;
    cout << "The larger character value is " << larger(p, q) << endl;

    return 0;
}
```

Output:

```
Enter 2 integers : 74 98
Enter 2 fractional numbers : 56.8 68.8
Enter any 2 characters : C c
The larger integer value is 98
The larger fractional value is 68.8
The larger character value is c
```

94. Class template for calculating the area of rectangle.

```
#include <iostream>
using namespace std;
template <class T>
class Rectangle
{
private:
    T length, breadth;

public:
    Rectangle()
    {
        length = 0;
        breadth = 0;
    }
    Rectangle(T x, T y)
    {
        length = x;
        breadth = y;
    }
```

```
    T area()
    {
        return (length * breadth);
    }
};

int main()
{
    Rectangle<int> reci(5, 10);
    Rectangle<float> recf(3.5, 4.5);

    cout << "The area with integer dimension is " << reci.area() << endl;
    cout << "The area with float dimension is " << recf.area() << endl;

    return 0;
}
```

Output:

```
The area with integer dimension is 50
The area with float dimension is 15.75
```

95. Stream classes description.

```
 /*
The stream classes:
        --> ifstream class : reading data from file
        --> ofstream class : writing data to file
        --> fstream class : for reading and writing.

There are 2 methods of opening the files. They are:
1) open file using constructor of the file related stream class.
    syntax:
            stream_class stream_object ("file_name", mode);

        filename : file name to be opened.
        mode : ios::in, ios::out, ios::app

2) open file using member function open().
    syntax:
            stream_class stream_object;
            stream_object.open("file_name", mode);

 */
```

96. Writing on a file using ofstream class.

```
// Program to write a message "Welcome to my college !" in a file named "welcome.txt" using
// ofstream.

#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ofstream file("welcome.txt");
    file << "Welcome to my college !";
    file.close();
    cout << "Data writing to file successfully completed !" << endl;
    return 0;
}
```

Output:

```
Data writing to file successfully completed !
```

19 file read and write > output > welcome.txt

```
1  Welcome to my college !
2  // Above line 1 data has been written into the file named "Welcome.txt"
3  // If you run this program on your computer then "welcome.txt" file will
4  // be automatically created and in that file above line 1 text will be
5  // written.
6  // However, if you run this program on online compilers, then "welcome.txt"
7  // file will not be created.
```

97. Writing on a file using fstream class.

```
// Program to write a message "My name is King Khan !" in a file named "welcome.txt" using
// fstream.

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

int main()
{
    fstream file("kingkhan.txt", ios::out);
    file << "My name is King Khan !";
    file.close();
    cout << "Data writing to file successfully completed !" << endl;
    return 0;
}
```

Output:

```
Data writing to file successfully completed !
```

19 file read and write > output > kingkhan.txt

```
1  My name is King Khan !
```

98. Writing (name, marks[5], roll) on a file using ofstream class.

```
// Program to write name, roll no, marks in 5 subjects, store the info in a file named
// "marks.txt".

#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    char name[20];
    int rollno;
    float phy, chem, cpp, math, eng;
    cout << "Enter name : ";
    cin.getline(name, sizeof(name));
    cout << "Enter roll : ";
    cin >> rollno;
    cout << "Enter marks in phy, chem, cpp, math, eng:" << endl;
    cin >> phy >> chem >> cpp >> math >> eng;
    cout << endl;
```

```
    ofstream file("marks.txt");
    file << "Name : " << name << endl;
    file << "Roll number : " << rollno << endl;
    file << "Marks in 5 subjects are:" << endl;
    file << "\t Physics : " << phy << endl;
    file << "\t Chemistry : " << chem << endl;
    file << "\t C++ : " << cpp << endl;
    file << "\t Math : " << math << endl;
    file << "\t English : " << eng << endl;
    file.close();

    cout << "Writing data on file sucessfully completed.." << endl;
    return 0;
}
```

Output:

```
Enter name : Bishal Bhat
Enter roll : 9
Enter marks in phy, chem, cpp, math, eng:
79 67 68 89 72

Writing data on file sucessfully completed..
```

19 file read and write > output > marks.txt

```
1  Name : Bishal Bhat
2  Roll number : 9
3  Marks in 5 subjects are:
4      Physics : 79
5      Chemistry : 67
6      C++ : 68
7      Math : 89
8      English : 72
```

99. Reading data of student (name, marks[5], roll) from file using ifstream.

```
// program to read name, roll number, and marks in 5 subjects from a file named "marks.txt".

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

int main()
{
    string str;
    ifstream file("marks.txt");
    while (getline(file, str))
    {
        cout << str << endl;
    }
    file.close();
    return 0;
}


// same program using fstream class:
```

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

int main()
{
    string str;
    fstream file("marks.txt", ios::in);
    while (getline(file, str))
    {
        cout << str << endl;
    }
    file.close();
    return 0;
}
```

Output: This output will be visible in Terminal window.

```
Name : Bishal Bhat
Roll number : 9
Marks in 5 subjects are:
        Physics : 79
        Chemistry : 67
        C++ : 68
        Math : 89
        English : 72
```

100. Writing on file using open() member function.

```cpp
// Program to write a "Welcome to my Domain Expansion !" in a file named "anime.txt" using
// open() function.

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

int main()
{
    ofstream file;
    file.open("anime.txt");
    file << "Welcome to my Domain Expansion !";
    file.close();

    cout << "Message writing to file successfully completed !" << endl;
    return 0;
}
```

Output:

```
Message writing to file successfully completed !
```

19 file read and write > output > anime.txt

```
1   Welcome to my Domain Expansion !
```

101. Writing on a file using terminal in real time.

```
// Writing on file directly from Terminal.

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

int main()
{
    string str;
    ofstream file;
    file.open("experiment.txt", ios::app);
    cout << "Write what you want to write here:" << endl;
    while (1)
    {
        getline(cin, str);
        if (str == "-1")
            break;
        file << str << endl;
    }
    file.close();

    return 0;
}
```

Output:

```
Write what you want to write here:
Hello every body, i'm Bishal Bhat.
I hope this message finds you well.
I made this pdf so that you can learn and practice  C++ program
without any problem.
Thank you !!
-1
```

19 file read and write > output > experiment.txt

```
1 Hello every body, i'm Bishal Bhat.
2 I hope this message finds you well.
3 I made this pdf so that you can learn and practice  C++ program
4 without any problem.
5 Thank you !!
```

102. Previous year Exam Qn.

```
/*
   A book shop in mnr sells Book and DVD. 2 class Book and DVD are inherited from the base
   class Media. The media class has common data members such as title and publication. The
   class Book has data member such as no_of_pages and DVD class has data member such as
   duration. Each class has member function such as getdata() and showdata().

   Write a C++ program for modeling the class hierarchy for Book shop and DVD shop. And process
   objects of these classes pointers of the base class Media.
*/

   #include <iostream>
   using namespace std;
   class Media
   {
   protected:
       char title[35], publication[50];

   public:
       virtual void getdata() = 0;
       virtual void showdata() = 0;
   };

   class Book : public Media
   {
   private:
       int no_of_pages;

   public:
       void getdata()
       {
           cout << " --> Enter Data for Book:" << endl;
           cout << "Enter Title : ";
           cin.getline(title, sizeof(title));
           cout << "Enter publication : ";
           cin.getline(publication, sizeof(publication));
           cout << "Enter no of pages : ";
           cin >> no_of_pages;
       }
       void showdata()
       {
           cout << endl;
           cout << " --> Displaying Data of Book:" << endl;
           cout << "Title : " << title << endl;
           cout << "Publication : " << publication << endl;
           cout << "No of pages : " << no_of_pages << endl;
       }
   };

   class DVD : public Media
   {
   private:
       float duration;

   public:
       ;
       void getdata()
       {
           cout << " --> Enter Data for DVD:" << endl;
           cin.ignore();
```

```
// used “cin.ignore();” because program was not waiting for taking title input.
        cout << "Enter Title : ";
        cin.getline(title, sizeof(title));
        cout << "Enter publication : ";
        cin.getline(publication, sizeof(publication));
        cout << "Enter duration in minutes : ";
        cin >> duration;
    }
    void showdata()
    {
        cout << endl;
        cout << " --> Displaying Data of DVD:" << endl;
        cout << "Title : " << title << endl;
        cout << "Publication : " << publication << endl;
        cout << "Duration : " << duration << endl;
    }
};

int main()
{
    Media *ptr[2];
    Book b;
    DVD d;

    ptr[0] = &b;
    ptr[1] = &d;

    ptr[0]->getdata();
    ptr[1]->getdata();

    ptr[0]->showdata();
    ptr[1]->showdata();

    return 0;
}
```

Output:

```
 --> Enter Data for Book:
Enter Title : The 48 Laws of Power
Enter publication : Viking Press
Enter no of pages : 452
 --> Enter Data for DVD:
Enter Title : Spider-Man
Enter publication : Columbia Pictures
Enter duration in minutes : 121

 --> Displaying Data of Book:
Title : The 48 Laws of Power
Publication : Viking Press
No of pages : 452

 --> Displaying Data of DVD:
Title : Spider-Man
Publication : Columbia Pictures
Duration : 121
```

***