**Aim: Implementation of call by value function.**

Code: -

// call by value

#include <iostream>

using namespace std;

int sum(int a, int b)

{

    int sum = a + b;

    return sum;

}

int main()

{

    int a, b;

    cin >> a >> b;

    int sum = sum(a, b);

    cout <<endl<<a<<endl<<b<<endl<<sum;

    return 0;

}

Output: -

|  |  |
| --- | --- |
| Input | Output |
| 100  20 | 100  20  120 |

**Aim: Passing a pointer in the function?**

Code:

// passing a pointer

#include <iostream>

using namespace std;

int sum(int \*a, int \*b)

{

    int sum = \*a + \*b;

    return sum;

}

int main()

{

    int a, b;

    cin >> a >> b;

    int sumy = sum(&a, &b);

    cout <<endl<<a<<endl<<b<<endl<<sumy;

    return 0;

}

Output: -

|  |  |
| --- | --- |
| Input | Output |
| 10  20 | 10  20  30 |

**Aim: Implementation of call by reference function.**

Code: -

// call by reference

#include <iostream>

using namespace std;

int sum(int &a, int &b)

{

    int sum = a + b;

    return sum;

}

int main()

{

    int a, b;

    cin >> a >> b;

    int sumy = sum(a, b);

    cout <<a<<endl<<b<<endl<<sumy;

    return 0;

}

Output: -

|  |  |
| --- | --- |
| Input | Output |
| 10  20 | 10  20  30 |

**Aim: Implementation of calling a function by a pointer.**

Code: -

#include <iostream>

using namespace std;

int sum(int a, int b)

{

    int sum = a + b;

    return sum;

}

int main()

{

    int a, b;

    cin >> a >> b;

    int (\*point\_fun)(int , int ) = &sum;

    int sumy = (\*point\_fun)(a,b);

    cout <<endl<<a<<endl<<b<<endl<<sumy;

    return 0;

}

Output: -

|  |  |
| --- | --- |
| Input | Output |
| 10  20 | 10  20  30 |
| 30  20 | 30  20  50 |

**Aim** :**To demonstrate array as an object.**

Code: -

#include<iostream>

using namespace std;

class Student// Class declaration for Student

{

int roll\_no;

char name[100];

public:

void getdata();// Function to input student data

void putdata();// Function to output student data

};

// Definition of function to input student data

void Student::getdata(){

cout << "Enter Roll Number : ";

cin >> roll\_no;

cout << "Enter Name : ";

cin >> name;

}

// Definition of function to output student data

void Student::putdata(){

cout << roll\_no << " ";

cout << name << " ";

cout << endl;

}

int main(){

// Declare an array of objects of class Student

Student name[100];

int n, i;

cout << "Enter Number of Students - ";

cin >> n;

// Input data for all the students

for(i = 0; i < n; i++)

name[i].getdata();

// Output the data for all the students

cout<< endl<< "Student Data - " << endl;

for(i = 0; i < n; i++)

name[i].putdata();

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
| Roll no: 1, Name: Mudit | Student data – 1, Mudit |
| Roll no: 2, Name: Sharad | Student data – 2, Sharad |
| Roll no: 3, Name: Yash | Student data – 3, Yash |

**Aim:** **Simple program to demonstrate the use of class.**

Code: -

#include <iostream>

using namespace std;

// Define the square class

class square{

public :

int side;

};

int main(){

// Create an object of the square class

square b1;

// Assign values to the member variables of b1

b1.side = 5;

// Calculate and output the area of square

cout << " area of the square is : " << b1.side \* b1.side ;

return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
| Side = 2 | Area of square is: 4 |
| Side = 10 | Area of square is : 100 |
| Side =5 | Area of square is : 25 |

**Aim:** **The use of “Private” access specifier.**

Code: -

#include <iostream>

using namespace std;

// Define a circle class

class circle {

private:

int radius;

public:

// methods to set the radius of the circle

void setradius(int r){

radius = r;

}

// Method to calculate the area of the circle

int area() {

return 3.14\*radius\*radius;

}

};

int main(){

// Create a circle object

circle r1;

// Set the height and width of the circle using the setter methods

r1.setradius(5);

// Calculate the area of the circle using the area() method and print it

cout<< "Require area is: " << r1.area() << endl;

return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
| 5 | 78 |
| 4 | 50 |
| 2 | 12 |

**Aim:** **To demonstrate the use of protected data members.**

Code: -

#include <bits/stdc++.h>

using namespace std;

// This is the base class that has a protected member variable

class Number{

protected:

int num;// protected member variable

};

class Child : public Number

{

public:

void giveNum(int id){

num = id;// accessing the protected member variable

}

// public member function that displays the value of the protected member variable

void displayNum(){

cout << "Protected number is: " << num << endl;

}

};

int main() {

Child obj1;

// calling the public member function

obj1.giveNum(81);

obj1.displayNum();

return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
| 81 | The value is: 81 |
| 100 | The value is: 100 |
| 230 | The value is: 230 |

**Aim:** **To make program containing simple Inline function.**

Code: -

#include <iostream>

using namespace std;

// inline function that returns the maximum of two integer values

inline int Max(int x, int y) {

return (x > y)? x : y; // if x is greater than y, return x, otherwise return y

}

// Main function for the program

int main() {

cout << "Max (20,10): " << Max(20,10) << endl; // calling the Max() function with x = 20 and y = 10

cout << "Max (-10,10): " << Max(-10,10) << endl; // calling the Max() function with x = -10 and y = 10

cout << "Max (1000,1001): " << Max(1000,1001) << endl; // calling the Max() function with x =1000 and y = 1001

return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
| 20,10 | Max (20,10): 20 |
| -10,10 | Max (-10,10): 10 |
| 1000,1001 | Max (1000,1001): 1001 |

**Aim:** **To use inline function for nesting.**

Code: -

#include <iostream> // includes the input/output stream library

using namespace std; // uses the standard namespace

class nest{ // defines a class called nest

int a, b, c; // defines three integer variables a, b, and c

inline int Avg(int a, int b, int c){ // defines an inline function called Avg that takes in three integer parameters and returns their average

return ((a+b+c)/3);

}

public:

void input\_num(){ // defines a public member function called input\_num that takes no parameters and prompts the user to enter three numbers

cout<< "Enter the three numbers: " << endl;

cin >> a >> b >> c; // reads in three numbers from the user and stores them in a, b, and c

}

void display\_num() { // defines a public member function called display\_num that takes no parameters and displays the average of the three numbers

int Average = Avg(a,b,c); // calculates the average of the three numbers using the Avg function

cout<< "The average of the three numbers is: " << Average; // displays the calculated average

}

};

int main(){ // the main function

nest A1; // creates an object of the nest class called A1

A1.input\_num(); // calls the input\_num member function of the A1 object to get user input

A1.display\_num(); // calls the display\_num member function of the A1 object to display the calculated average

return 0; // returns 0 to the operating system }

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
| 10,20,30 | The average of the three numbers is: 20 |
| -10,25,30 | The average of the three numbers is: 15 |
| 5,5,20 | The average of the three numbers is: 10 |

**Aim:** **To demonstrate the use of pointers in C++ by creating a pointer “ptr” that points to an integer variable “var”.**

Code: -

#include <iostream>

using namespace std;

int main(int argc, const char \* argv[]) {

int var = 20; // declare an integer variable named var, and initialize it to 20

int\* ptr; // declare a pointer variable named ptr, which can point to an integer

ptr = &var; // assign the address of var to ptr

// print the values of var, ptr, and the value pointed to by ptr using cout statements

cout << "value at ptr = " << ptr << "\n";

cout << "value at var = " << var << "\n";

cout << "value at \*ptr = " << \*ptr << "\n";

return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
| 20 | value at ptr = 0x61ff08  value at var = 20  value at \*ptr = 20 |
| 200 | value at ptr = 0x61ff08  value at var = 200  value at \*ptr = 200 |
| 150 | value at ptr = 0x61ff08  value at var = 150  value at \*ptr = 150 |

**Aim:** **To demonstrate the use of function overloading.**

Code: -

#include <iostream>

using namespace std;

// Global variable that is used to store the area

int area = 1;

// First version of the overloaded function that takes two integer arguments and calculates their area

void overload(int a, int b){

area = a \* b; // area the two integer arguments and storing the result in the global variable

cout << "area using 1st function is: " << area << "\n"; // displaying the result

}

// Second version of the overloaded function that takes three integer arguments and calculates their area

void overload(int a, int b, int c){

area = a \* b \* c; // area the three integer arguments and storing the result in the global variable

cout << "area using 2nd function is: " << area << "\n"; // displaying the result

}

// Main function for the program

int main(){

overload(6, 4); // calling the first version of the overloaded function with two integer arguments

overload(4, 5, 8); // calling the second version of the overloaded function with three integer arguments

return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
| area of (6,4) and area of (4,5,8) | area using 1st function is: 24  area using 2nd function is: 160 |
| area of (10,5) and area of (10,15,8) | area using 1st function is: 50  area using 2nd function is: 1200 |
| area of (10,20) and area of (10,15,20) | area using 1st function is: 200  area using 2nd function is: 3000 |

**Aim:** **To demonstrate the passing of default arguments.**

Code: -

#include <iostream>

using namespace std;

float area(int x, float base = 0, float hei = 0, float length = 0, float breath = 0, float height = 0, float rad = 0)

{

    // Check the shape type and calculate area accordingly

    if (x == 1)

    {

        return length \* breath;

    }

    if (x == 2)

    {

        return 2 \* (length \* breath + height \* length + height \* breath);

    }

    if (x == 3)

    {

        return 3.14 \* rad \* rad;

    }

    if (x == 4)

    {

        return 0.5 \* (hei \* base);

    }

}

int main()

{

    int x;

    float a4, a3, a1, a2;

    // Get the shape type from the user

    cin >> x;

    // Calculate area based on the shape type

    switch (x)

    {

    case 1:

    {

        float l, b;

        cout << "Enter the values of lenght and breath : ";

        cin >> l >> b;

        // Call the area function with the necessary parameters

        a1 = area(x, 0, 0, l, b);

        cout << "area = " << a1;

        break;

    }

    case 2:

    {

        float l, b, h;

        cout << "Enter the values of lenght and breath and height: ";

        cin >> l >> b >> h;

        // Call the area function with the necessary parameters

        a2 = area(x, 0, 0, l, b, h);

        cout << "area = " << a2;

        break;

    }

    case 3:

    {

        float r;

        cout << "Enter the values of radius : ";

        cin >> r;

        // Call the area function with the necessary parameters

        a3 = area(x, 0, 0, 0, 0, 0, r);

        cout << "area = " << a3;

        break;

    }

    case 4:

    {

        float b, h;

        cout << "Enter the values of  base and height: ";

        cin >> b >> h;

        // Call the area function with the necessary parameters

        a4 = area(x, b, h);

        cout << "area = " << a4;

        break;

    }

    }

    return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
| 3  Enter the values of radius : 5 | area = 78.5 |
| 2  Enter the values of lenght and breath and height: 10 20 30 | area = 2200 |

**Aim:** **To demonstrate the implementation of dynamic allocation of an array.**

Code: -

#include <iostream>

using namespace std;

int \*dup(int ar[], int n)

{

    // Dynamically allocate memory for an integer array of size 1000

    int\* arr = new int[1000];

    // Initialize all elements of the array to 0

    for (int i = 0; i < 1000; i++)

        arr[i] = 0;

    // Traverse the given array and increment the count of each element in the arr array

    for (int i = 0; i < n; i++)

        arr[ar[i]]++;

    // Return the pointer to the arr array

    return arr;

}

int main()

{

    int n;

    cin >> n;

    // Declare an integer array of size n

    int ar[n];

    // Input the elements of the array from the user

    for (int i = 0; i < n; i++)

    {

        cin >> ar[i];

    }

    // Call the dup function and store the pointer to the returned array in ptr

    int \*ptr = dup(ar, n);

    // Print the count of each element in the array

    cout<<" element"<<" number of times repeated"<<endl;

    for(int i=0;i<1000;i++){

    if(ptr[i]!=0)

    cout<<" "<<i<<"          "<<ptr[i]<<endl;

    }

    // Free the dynamically allocated memory

    delete[] ptr;

    return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
| 3  2  1  1 | element number of times repeated  1 2  2 1 |
| 5  4  3  2  2  2 | element number of times repeated  2 3  3 1  4 1 |

**Aim:** **To demonstrate the constructor overloading with student class.**

Code: -

#include <iostream>

using namespace std;

class Student

{

private:

    string name, branch, address;

    int rollno;

public:

    Student()

    {

        name = "0";

        branch = "0";

        rollno = 0;

        address = "0";

    }

    Student(string na, string bra, string add, int rol)

    {

        name = na;

        branch = bra;

        rollno = rol;

        address = add;

    }

    void display()

    {

        cout << endl

             << "Name of the student is " << name << endl;

        cout << "Rollno of the student is " << rollno << endl;

        cout << "Address of the student is " << address << endl;

        cout << "Branch of the student is " << branch << endl;

    }

    void modify()

    {

        int mod;

        cout << "Enter 1 for Name \n 2 for address \n 3 for rollno \n 4 for branch ";

        cin >> mod;

        switch (mod)

        {

        case 1:

            cout << "Enter the name of the student ";

            cin >> name;

            break;

        case 2:

            cout << "Enter the address of the student ";

            cin >> address;

            break;

        case 3:

            cout << "Enter the rollno of the student ";

            cin >> rollno;

            break;

        case 4:

            cout << "Enter the branch of the student ";

            cin >> branch;

            break;

        default:

            break;

        }

        display();

    }

};

int main()

{

    string na, bra, add;

    int ro;

    cout << "Enter the name of the student ";

    cin >> na;

    cout << "Enter the rollno of the student ";

    cin >> ro;

    cout << "Enter the address of the student ";

    cin >> add;

    cout << "Enter the branch of the student ";

    cin >> bra;

    Student su1(na, bra, add, ro);

    Student su2;

    su1.display();

    su2.display();

    su1.modify();

    return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
| Enter the name of the student Mudit  Enter the rollno of the student 21319  Enter the address of the student jaipur  Enter the branch of the student IT | Name of the student is Mudit  Rollno of the student is 21319  Address of the student is jaipur  Branch of the student is IT |
| Enter the name of the student yash  Enter the rollno of the student 21335  Enter the address of the student bareliy  Enter the branch of the student IT | Name of the student is yash  Rollno of the student is 21335  Address of the student is bareliy  Branch of the student is IT |

**Aim:** **To demonstrate the implementaion of constructror overloading with employee class.**

Code: -

#include <iostream>

using namespace std;

class Employe

{

private:

    // Declare pointers for the company ID and employee ID

    int \*ptr1, \*ptr2;

    // Declare strings for the company name, employee name, and company address

    string Cname, Ename, Address;

public:

    // Default constructor

    Employe()

    {

        // Initialize the pointers to null

        ptr1 = new int;

        ptr2 = new int;

        // Initialize the strings to empty

        Cname = "";

        Ename = "";

        Address = "";

        // Initialize the IDs to 0

        \*ptr1 = 0;

        \*ptr2 = 0;

    }

    // Constructor with parameters

    Employe(string Ena, string Cna, string add, int ci, int ei)

    {

        // Allocate memory for the ID pointers

        ptr1 = new int;

        ptr2 = new int;

        // Initialize the strings and IDs with the given values

        Cname = Cna;

        Ename = Ena;

        Address = add;

        \*ptr1 = ei;

        \*ptr2 = ci;

    }

    // Display method to print the information of the employee and company

    void display()

    {

        cout << endl

             << "Name of the Employe is : " << Ename << endl;

        cout << "Name of the Company is : " << Cname << endl;

        cout << "Address of the Company is : " << Address << endl;

        cout << "EID of the Employe is : " << \*ptr1 << endl;

        cout << "CID of the Company is : " << \*ptr2 << endl;

    }

};

// Main function

int main()

{

    // Declare variables to store input values

    int EID, CID;

    string Ename, Cname, address;

    // Prompt the user to enter information for the employee and company

    cout << "Enter the name of the Employe : ";

    cin >> Ename;

    cout << "Enter the name of the Company : ";

    cin >> Cname;

    cout << "Enter the address of the Company : ";

    cin >> address;

    cout << "Enter the EID of the Employe : ";

    cin >> EID;

    cout << "Enter the CID of the Company : ";

    cin >> CID;

    // Create an object of the Employe class using the default constructor

    Employe \*e1 = new Employe();

    // Create an object of the Employe class using the parameterized constructor

    Employe e2(Ename, Cname, address, CID, EID);

    // Display the information of the objects

    e1->display();

    e2.display();

    // Free the memory allocated for the object created with the default constructor

    delete e1;

    return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
| Enter the name of the Employe : sachin  Enter the name of the Company : LNT  Enter the address of the Company : delhi  Enter the EID of the Employe : 21  Enter the CID of the Company : 43 | Name of the Employe is : LNT  Name of the Company is : sachin  Address of the Company is : delhi  EID of the Employe is : 21  CID of the Company is : 43 |
| Enter the name of the Employe : mudit  Enter the name of the Company : google  Enter the address of the Company : washington  Enter the EID of the Employe : 21232  Enter the CID of the Company : 13322 | Name of the Employe is : mudit  Name of the Company is : google  Address of the Company is : washington  EID of the Employe is : 21232  CID of the Company is : 13322 |

**Aim:** **To demonstrate the constructor and destructor calls.**

Code: -

#include <iostream>

#include <string>

using namespace std;

class checking

{

private:

    const string m\_id; // constant string member variable

public:

    // Constructor with initialization list

    checking(const string id) : m\_id(id)

    {

        cout << "Constructor called: " << m\_id << endl;

    }

    // Destructor

    ~checking()

    {

        cout << "Destructor called: " << m\_id << endl;

    }

};

// Global object of checking class

checking globalObj("Global\_object");

// Main function

int main()

{

    cout << "Starting main function" << endl;

    // Automatic object of checking class

    checking autoObj("Auto\_Object");

    {

        cout << "Entering new scope" << endl;

        // Static object of checking class

        static checking staticObj("Static\_Object");

        // Register object of checking class

        register checking regObj("Register\_Object");

        cout << "Leaving new scope" << endl;

    }

    cout << "Exiting main function" << endl;

    return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
|  | Constructor called: Global\_object  Starting main function  Constructor called: Auto\_Object  Entering new scope  Constructor called: Static\_Object  Constructor called: Register\_Object  Leaving new scope  Destructor called: Register\_Object  Exiting main function  Destructor called: Auto\_Object  Destructor called: Static\_Object  Destructor called: Global\_object |

**Aim:** **To demonstrate the inheritance concept with account and saving class.**

Code: -

#include<bits/stdc++.h>

using namespace std;

class account

{

protected:

int acc\_number, balance;

string holder;

public:

account()

{

cout<<"Enter account number:";

cin>>acc\_number;

cout<<"Enter name of the account holder:";

cin>>holder;

cout<<"Enter balance:";

cin>>balance;

}

void getdata()

{

cout<<"Account:"<<acc\_number<<" belongs to "<<holder<<" and balance is:"<<balance<<endl;

}

};

class saving:public account

{

protected:

int saving\_rate;

public:

saving()

{

cout<<"\nSaving Account"<<endl;

cout<<"Enter rate of interest in (%):";

cin>>saving\_rate;

balance= balance+balance\*saving\_rate;

}

void getdata()

{

cout<<"Account number:"<<acc\_number<<" belong to:"<<holder<<" and balance is:"<<balance<<" with interest rate of:"<<saving\_rate<<"%"<<endl;

}

};

class fd:public account

{

protected:

int maturity\_year;

int breaking\_year;

public:

fd()

{

cout<<"\nFixed deposite"<<endl;

cout<<"Enter maturity year of fd:";

cin>>maturity\_year;

breaking\_year=maturity\_year;

cout<<"Enter withdraw of year you want:";

cin>>breaking\_year;

}

void getdata()

{

if (breaking\_year < maturity\_year)

{

balance = balance-balance\*0.1;

cout<<"You broke your fd before mature year,So You will be charged with 10%"<<endl;

account::getdata();

}

else

{

balance = balance+balance\*0.1;

account::getdata();

}

}

};

int main()

{

saving s1;

fd f1;

s1.getdata();

f1.getdata();

return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
| Enter account number:21319  Enter name of the account holder:mudit  Enter balance:10000  Saving Account  Enter rate of interest in (%):10  Enter account number:21320  Enter name of the account holder:sarthak  Enter balance:20000  Fixed deposite  Enter maturity year of fd:23  Enter withdraw of year you want:12 | Account number:21319 belong to:mudit and balance is:110000 with interest rate of:10%  You broke your fd before mature year,So You will be charged with 10%  Account:21320 belongs to sarthak and balance is:17999 |

**Aim:** **To demonstrate the implematation of friend function by adding two different unit of distance.**

Code: -

#include <iostream>

using namespace std;

class dist1

{

private:

float m, cm;

public:

// Constructor to input the distance in meters and centimeters

dist1()

{

cout << "Enter the distance in m : ";

cin >> m;

cout << "Enter the distance in cm : ";

cin >> cm;

}

friend class sum;

};

class dist2

{

private:

float ft, in;

public:

// Constructor to input the distance in feet and inches

dist2()

{

cout << "Enter the distance in ft : ";

cin >> ft;

cout << "Enter the distance in in : ";

cin >> in;

}

friend class sum;

};

class sum

{

float a, b;

public:

void su(dist1 &d1, dist2 &d2)

{

d2.ft = d2.ft \* 0.3048; // converting feet to meters

d2.in = d2.in \* 2.54; // converting inches to centimeters

a = d1.m + d2.ft; // adding the distances in meters

b = d1.cm + d2.in; // adding the distances in centimeters

cout << "The sum of distance is : " << a << "m" << endl

<< b << "cm";

}

};

int main()

{

dist1 di1;

dist2 di2;

sum s;

s.su(di1, di2);

return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
| Enter the distance in m : 10  Enter the distance in cm : 2000  Enter the distance in ft : 24333  Enter the distance in in : 12  The sum of distance is : 7426.7m | The sum of distance is : 7426.7m  2030.48cm |
| Enter the distance in m : 21  Enter the distance in cm : 300  Enter the distance in ft : 45  Enter the distance in in : 30 | The sum of distance is : 34.716m  376.2cm |

**Aim:** **To demonstrate operator overloading using member function.**

Code: -

#include <iostream>

using namespace std;

class dist1 {

private:

float m, cm;

public:

// Constructor to get user input for distance in meters and centimeters

dist1() {

cout << "Enter the distance in m : ";

cin >> m;

cout << "Enter the distance in cm : ";

cin >> cm;

}

// Overloading ++ operator

void operator++() {

    m = m + 2;

    cm = cm + 2;

}

// Member

void getdist() {

    cout << "Value after overloading : " << m << " m   " << cm << " cm  " << endl;

}

// Overloading + operator as a member function to add two distances and display the sum

void operator+(dist1 obj) {

    obj.m = m + obj.m;

    obj.cm = cm + obj.cm;

    cout << "The sum of distance is : " << obj.m << "m" << endl << obj.cm << "cm";

}

};

int main() {

// Creating two objects of dist1 class

dist1 di1, di2;

// Overloading ++ operator on object di1

++di1;

// Displaying the value after operator overloading

di1.getdist();

// Overloading + operator on object di1 with parameter di2 and displaying the sum of distances

di1 + di2;

return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
| Enter the distance in m : 200  Enter the distance in cm : 2000  Enter the distance in m : 100  Enter the distance in cm : 1000 | Value after overloading : 202 m 2002 cm  The sum of distance is : 302m  3002cm |
| Enter the distance in m : 150  Enter the distance in cm : 300  Enter the distance in m : 450  Enter the distance in cm : 600 | Value after overloading : 152 m 302 cm  The sum of distance is : 602m  902cm |

**Aim:** **To demonstrate the operator overloading using friend function.**

Code: -

#include <iostream>

using namespace std;

class dist1

{

public:

float m, cm;

// Method to get the distance input in meters and centimeters

void getdata()

{

    cout << "Enter the distance in m : ";

    cin >> m;

    cout << "Enter the distance in cm : ";

    cin >> cm;

}

// Method to display the distance

void getdist()

{

    cout << "Value after overloading : " << m << " m   " << cm << " cm  " << endl;

}

friend void operator++(dist1 &);

friend dist1 operator+(dist1 &, dist1 &);

// Declare destructor for dist1 class

~dist1() {}

};

// Definition of operator++ as a friend function of dist1 class

void operator++(dist1 &t)

{

t.m = t.m + 2;

t.cm = t.cm + 2;

}

dist1 operator+(dist1 &d1, dist1 &d2)

{

dist1 d3;

d3.m = d1.m + d2.m;

d3.cm = d1.cm + d2.cm;

return d3;

}

// Main function

int main()

{

// Create objects of dist1 class

dist1 di1, di2, di3;

// Get the input for di1 and di2 objects

di1.getdata();

di2.getdata();

di3 = di1 + di2;

// Display the sum of distances

di3.getdist();

// Increment the distance in di1 object by 2 meters and 2 centimeters

++di1;

di1.getdist();

return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
| Enter the distance in m : 12  Enter the distance in cm : 24  Enter the distance in m : 13  Enter the distance in cm : 26 | Value after overloading : 25 m 50 cm  Value after overloading : 14 m 26 cm |
| Enter the distance in m : 25  Enter the distance in cm : 100  Enter the distance in m : 30  Enter the distance in cm : 200 | Value after overloading : 55 m 300 cm  Value after overloading : 27 m 102 cm |

**Aim:** **To demonstrate the set operations using function overloading.**

Code: -

#include <bits/stdc++.h>

using namespace std;

class Set

{

private:

int \*arr;

int size;

public:

//Constructor for initializing the size of array

Set(int s)

{

size = s;

arr = new int[size];

}

//Constructor for initializing the size and values of array

Set(int s, int \*values)

{

    size = s;

    arr = new int[size];

    for (int i = 0; i < size; i++)

    {

        arr[i] = values[i];

    }

    sort(arr, arr + size);

}

// Overloaded operator+ for set union operation

Set operator+(const Set &other)

{

    int \*temp = new int[size + other.size];

    int i = 0, j = 0, k = 0;

    while (i < size && j < other.size)

    {

        if (arr[i] < other.arr[j])

        {

            temp[k++] = arr[i++];

        }

        else if (other.arr[j] < arr[i])

        {

            temp[k++] = other.arr[j++];

        }

        else

        {

            temp[k++] = arr[i++];

            j++;

        }

    }

    while (i < size)

    {

        temp[k++] = arr[i++];

    }

    while (j < other.size)

    {

        temp[k++] = other.arr[j++];

    }

    Set unionSet(k, temp);

    delete[] temp;

    return unionSet;

}

// Overloaded operator- for set intersection operation

Set operator-(const Set &other)

{

    int \*temp = new int[size];

    int i = 0, j = 0, k = 0;

    while (i < size && j < other.size)

    {

        if (arr[i] < other.arr[j])

        {

            i++;

        }

        else if (other.arr[j] < arr[i])

        {

            j++;

        }

        else

        {

            temp[k++] = arr[i++];

            j++;

        }

    }

    Set intersectionSet(k, temp);

    delete[] temp;

    return intersectionSet;

}

// Function to print the value of the array

void print()

{

    for (int i = 0; i < size; i++)

    {

        cout << arr[i] << " ";

    }

    cout << endl;

}

// Destructor to delete the dynamically allocated memory

~Set()

{

    delete[] arr;

}

};

int main()

{

int s1, s2;

cout << "Enter the values of s1 and s2 : ";

cin >> s1 >> s2;

int values1[s1];

cout << "Enter the values of array 1 : ";

for (int i = 0; i < s1; i++)

{

    cin >> values1[i];

}

int values2[s2];

cout << "Enter the values of array 2 : ";

for (int i = 0; i < s2; i++)

{

    cin >> values2[i];

}

Set a(s1, values1);

Set b(s2, values2);

Set c = a + b; // set union operation

Set d = a - b; // set intersection operation

// Printing the result of set operations

cout << "Set union: ";

c.print();

cout << "Set intersection: ";

d.print();

return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
| Enter the values of s1 and s2 : 3 4  Enter the values of array 1 : 2 4 5  Enter the values of array 2 : 6 7 8 9 | Set union: 2 4 5 6 7 8 9  Set intersection: |

**Aim:** **To find the area of circle and rectangle using polymorphism.**

Code: -

#include <iostream>

#include <cmath>

using namespace std;

class Shape {

public:

virtual double area() = 0; // pure virtual function

};

class Circle : public Shape {

private:

double radius;

public:

Circle(double r) {

radius = r;

}

double area() {

return M\_PI \* radius \* radius;

}

};

class Rectangle : public Shape {

private:

double width, height;

public:

Rectangle(double w, double h) {

width = w;

height = h;

}

double area() {

return width \* height;

}

};

int main() {

Shape \*shape;

Circle circle(5);

Rectangle rectangle(4, 6);

// using polymorphism to calculate the area

shape = &circle;

cout << "Area of circle: " << shape->area() << endl;

shape = &rectangle;

cout << "Area of rectangle: " << shape->area() << endl;

return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
|  | Area of circle: 78.5398  Area of rectangle: 24 |

**Aim:** **To create a file and open it in read and write mode.**

Code: -

#include <iostream>

#include <fstream>

using namespace std;

int main()

{

    // Declare file name

    string fileName = "example.txt";

    // Open file in read mode

    ifstream inFile(fileName);

    if (inFile.is\_open())

    {

        // Read file contents

        cout << "File contents (read mode):" << endl;

        string line;

        while (getline(inFile, line))

        {

            cout << line << endl;

        // Close file

        inFile.close();

    }

    else

    {

        cout << "Failed to open file for reading" << endl;

    }

    // Open file in write mode

    ofstream outFile(fileName);

    if (outFile.is\_open())

    {

        // Write data to file

        cout << "Writing to file (write mode)..." << endl;

        outFile << "This is a test file." << endl;

        // Close file

        outFile.close();

    }

    else

    {

        cout << "Failed to open file for writing" << endl;

    }

    // Open file in binary mode

    fstream binFile(fileName, ios::binary | ios::in | ios::out);

    if (binFile.is\_open())

    {

        // Read data from file

        cout << "File contents (binary mode):" << endl;

        char buffer[256];

        binFile.read(buffer, sizeof(buffer));

        cout << buffer << endl

        // Write data to file

        cout << "Writing to file (binary mode)..." << endl;

        binFile.write("New data", 8);

        // Close file

        binFile.close();

    }

    else

    {

        cout << "Failed to open file for binary I/O" << endl;

    }

    return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
|  | Failed to open file for reading  Writing to file (write mode)...  File contents (binary mode):  This is a test file.  Writing to file (binary mode)... |

**Aim:** **To check whether constructor and destructor can be virtual.**

Code: -

#include <iostream>

using namespace std;

class check

{

public:

     virtual check() {}

    ~check() {}

    ~check() {}

};

// derived class

class derived : public check

{

public:

    // default constructor

    derived()

    {

    }

};

int main()

{

    derived d;

    return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
|  | constructors cannot be declared 'virtual' [-fpermissive]  virtual check() {} |

**Aim:** **To check whether constructor and destructor can be member function.**

Code: -

#include <iostream>

using namespace std;

class check

{

    private:

    int x;

    friend check();

       friend ~check();

};

// Constructor definition

check::check()

{

    cout << "Constructor"<< endl;

    cin>>x;

    cout<<" x : "<<x<< endl;

}

// Destructor definition

check::~check()

{

    cout << "  Destructor called" << endl;

}

// Main function

int main()

{

       check c;

    return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
|  | error: expected unqualified-id before ')' token  friend check(); |

**Aim:** **To copy the contents of one file into another file.**

Code:

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int main()

{

    string file1Name = "File1.txt";

    string file2Name = "file2.txt";

    // Open the first file for reading

    ifstream file1(file1Name);

    // Open the second file for writing

    ofstream file2(file2Name);

    // Check if both files were successfully opened

    if (file1.is\_open() && file2.is\_open())

    {

        string line;

        while (getline(file1, line))

        {

            file2 << line << endl;

        }

        // Close both files after copying is complete

        file1.close();

        file2.close();

        // Print a success message

        cout << "File copied successfully!" << endl;

    }

    else

    {

        // Print an error message if one or both files failed to open

        cout << "Failed to open files." << endl;

        return 1;

    }

    // Open the copied file for reading

    ifstream copiedFile(file2Name);

    // Check if the copied file was successfully opened

    if (copiedFile.is\_open())

    {

        // Print the contents of the copied file

        cout << "Contents of copied file:" << endl;

        string line;

        while (getline(copiedFile, line))

        {

            cout << line << endl;

        }

        // Close the copied file after reading is complete

        copiedFile.close();

    }

    else

    {

        cout << "Failed to open copied file." << endl;

        return 1;

    }

    return 0;

}

Input/Output Table: -

|  |  |
| --- | --- |
| Input | Output |
|  | File copied successfully! |