**OBJECT ORIENTED PROGRAMMING**

Table of Contents

[INTRODUCTION: 6](#_Toc125916991)

[DYNAMIC MEMORY ALLOCATION: 6](#_Toc125916992)

[Memory Management using New and delete keyword**.** 6](#_Toc125916993)

[Lab Work: (New and Delete opeartor) 8](#_Toc125916994)

[**Lab report 1:** 10](#_Toc125916995)

[1. C++ program to print the numbers from 1 to 10. 10](#_Toc125916996)

[2. C++ program to find the area of a circle using define directives. 10](#_Toc125916997)

[3. C++ program to check the largest number between two input numbers using if else. 10](#_Toc125916998)

[4. C++ program to print the largest number between 3 input numbers using nested if else. 10](#_Toc125916999)

[5. C++ program to illustrate setw, setprecision and setfill. 10](#_Toc125917000)

[6. C++ program to illustrate implicit and explicit type conversion. 10](#_Toc125917001)

[7. C++ program to store the score of n students and display it. 10](#_Toc125917002)

[**Control Statement:** 21](#_Toc125917003)

[1. for loop 21](#_Toc125917004)

[2. while loop 21](#_Toc125917005)

[3. do while loop 21](#_Toc125917006)

[**Class Works:** 21](#_Toc125917007)

[ C++ program to find the sum of two natural numbers using for loop. 21](#_Toc125917008)

[ C++ program to find the sum of two natural numbers using while loop. 21](#_Toc125917009)

[ C++ program to find the sum of two natural numbers using do while loop. 21](#_Toc125917010)

[ C++ program to find the factorial of the given number using for loop. 21](#_Toc125917011)

[ C++ program to find the factorial of the given number using while loop. 21](#_Toc125917012)

[ C++ program to find the factorial of the given number using do while loop. 21](#_Toc125917013)

[ C++ program to find the fibonacci series upto n number using while loop. 21](#_Toc125917014)

[ C++ program to find HCF/GCD. 21](#_Toc125917015)

[ C++ program to illustrate break and continue. 21](#_Toc125917016)

[ C++ program to create a calculator using switch statement. 21](#_Toc125917017)

[**Pointer** 33](#_Toc125917018)

[Operators in pointers: 33](#_Toc125917019)

[1. Reference operator (&): 33](#_Toc125917020)

[2. Dereference operator (\*) : 33](#_Toc125917021)

[**Pointer & Arrays :** 35](#_Toc125917022)

[ C++ program to demonstrate the Use of Array and pointer. 35](#_Toc125917023)

[**Null Pointer:** 36](#_Toc125917024)

[ C++ program to demonstrate null pointer. 36](#_Toc125917025)

[Function Call, Function Definition and Function Declaration Program : 37](#_Toc125917026)

[Function Overloading: 38](#_Toc125917027)

[**Function** 40](#_Toc125917028)

[Inline Function: 40](#_Toc125917029)

[Default Arguments: 42](#_Toc125917030)

[Pass by reference: 44](#_Toc125917031)

[Return by Reference: 46](#_Toc125917032)

[**Slope:** 47](#_Toc125917033)

[1. Local scope: 47](#_Toc125917034)

[2. Global Scope: 47](#_Toc125917035)

[ C++ program to demonstrate local and global scope of a variable. 47](#_Toc125917036)

[Class Scope: 48](#_Toc125917037)

[ C++ program to find the area of a rectangle using class scope. 48](#_Toc125917038)

[File Scope: 50](#_Toc125917039)

[ Take two inputs from the user, if both of the numbers are equal find its sum and if not then display their products. 51](#_Toc125917040)

[Chapter:3 53](#_Toc125917041)

[**Class and Objects** 53](#_Toc125917042)

[What is class? 53](#_Toc125917043)

[ C++ program to illustrate class & objects. 53](#_Toc125917044)

[ C++ program to find the area of a rectangle and volume using class and objects. 54](#_Toc125917045)

[ C++ program to find the area of a rectangle and volume of three boxes using class and objects. 56](#_Toc125917046)

[Scope Resolution Operator(::) 59](#_Toc125917047)

[ C++ program to illustrate scope resolution(::) operator 59](#_Toc125917048)

[**Constructor:** 60](#_Toc125917049)

[Types of Constructor 60](#_Toc125917050)

[1. Default constructor 60](#_Toc125917051)

[2. Parameterized constructor 61](#_Toc125917052)

[3. Copy constructor 62](#_Toc125917053)

[ C++ program to display the roll number and student name using default constructor 64](#_Toc125917054)

[Types of copy Constructor: 67](#_Toc125917055)

[1. Shallow Copy Constructor: 67](#_Toc125917056)

[2. Deep Copy Constructor: 69](#_Toc125917057)

[Destructor: 70](#_Toc125917058)

[ C++ program to find the area of a circle using destructor. 71](#_Toc125917059)

[**Q. Differentiate between constructor and member function with example.** 72](#_Toc125917060)

[**Object as a function Arguments:** 73](#_Toc125917061)

[ C++ program to demonstrate area of a rectangle using object as a function arguments. 73](#_Toc125917062)

[Structure in C++ 75](#_Toc125917063)

[Static data members: 77](#_Toc125917064)

[Unit: 4 78](#_Toc125917065)

[Operator Overloading 78](#_Toc125917066)

[**Overview of Unary & binary operator overloading:** 79](#_Toc125917067)

[Unary Operator Overloading: 79](#_Toc125917068)

[**Example of Unary Operator:** 79](#_Toc125917069)

[A. Unary plus: 79](#_Toc125917070)

[B. Unary minus: 79](#_Toc125917071)

[**Example of Decrement/increment Operator:** 80](#_Toc125917072)

[A. Decrement Operator: 80](#_Toc125917073)

[B. Increment Operator: 80](#_Toc125917074)

[**Example of Address Operator:** 81](#_Toc125917075)

[A. Address of the operator: 81](#_Toc125917076)

[**Example of sizeof Operator:** 82](#_Toc125917077)

[**Example of logic not Operator:** 83](#_Toc125917078)

[**Example of Bitwise not/ One’s complement Operator:** 83](#_Toc125917079)

[ C++ program to demonstrate the use of unary operator overloading, increment++ and decrement-— operator overloading. 85](#_Toc125917080)

[**Binary Operator Overloading:** 87](#_Toc125917081)

[ C++ program to demonstrate the binary operator overloading. 87](#_Toc125917082)

[**Assignment Operator Overloading:** 89](#_Toc125917083)

[ C++ program to demonstrate the assignment operator overloading. 89](#_Toc125917084)

[**Arithmetic Operators Overloading (Binary Operators): [ +, - , \*, /]** 91](#_Toc125917085)

[ C++ program to overload the binary operator + to add two complex numbers. 91](#_Toc125917086)

[**Comparision Operators Overloading(Binary)** 93](#_Toc125917087)

[ C++ program to demonstrate comparision operator. 93](#_Toc125917088)

[**Possible type conversion:** 96](#_Toc125917089)

[**1.** **Conversion from basic type to class type:** 96](#_Toc125917090)

[ C++ program to convert minutes into minutes and hours using “constructor”. 96](#_Toc125917091)

[ C++ program to convert inch into inch and feet using “constructor”. 97](#_Toc125917092)

[ C++ program to convert Celsius into Fahrenheit and Celsuis using “constructor”. 97](#_Toc125917093)

[**2.** **Conversion from class type to basic type.** 97](#_Toc125917094)

[ C++ program to convert hours and minutes into the minutes. 97](#_Toc125917095)

[ C++ program to convert inch and feet into the inch. 99](#_Toc125917096)

[**3.** **Conversion from one class type to another class type.** 100](#_Toc125917097)

[ C++ program to convert class time to another minute class. 100](#_Toc125917098)

[ C++ program to convert “basic to class” type using “operator overloading” to convert minutes into minutes and hours. 103](#_Toc125917099)

[Unit: 5 105](#_Toc125917100)

[**Inheritance** 105](#_Toc125917101)

[ C++ program to illustrate the inheritance by giving an example to find an area and a volume of a cuboid. 105](#_Toc125917102)

[There types of accessifier: 106](#_Toc125917103)

[Inherited class(child class) 107](#_Toc125917104)

[**Types of Inheritance:** 108](#_Toc125917105)

[1. Single Inheritance: 108](#_Toc125917106)

[ C++ program to add the numbers to demonstrate single inheritance. 108](#_Toc125917107)

[2. Multiple Inheritance 109](#_Toc125917108)

[ C++ program to store the marks of both subjects and sports of a Student and find the total marks of the student using multiple inheritance 109](#_Toc125917109)

[3. Multilevel Inheritance 111](#_Toc125917110)

[ C++ program to take the input in first class and find the area of a rectangle in second class and then volume in third class and display it. 111](#_Toc125917111)

[4. Hierarchical Inheritance 113](#_Toc125917112)

[5. Hybrid Inheritance 114](#_Toc125917113)

[Function Overriding: 116](#_Toc125917114)

[ C++ program to demonstrate the function overriding 116](#_Toc125917115)

[Ambiguity in Multiple Inheritance : 118](#_Toc125917116)

[C++ program to demonstrate 120](#_Toc125917117)

[Abstract base class : 121](#_Toc125917118)

[ C++ program to demonstrate abstract base class 121](#_Toc125917119)

[ C++ program to calculate the area using abstract base class. 121](#_Toc125917120)

[ C++ program to calculate the area of a triangle using abstract base class. 123](#_Toc125917121)

[Aggression ( class within class) 124](#_Toc125917122)

[constructor and destructor in derived class : 125](#_Toc125917123)

[Unit : 6 127](#_Toc125917124)

[Virtual funtion, Polymorphism and Miscellaneaous 127](#_Toc125917125)

[Virtual function: 127](#_Toc125917126)

[Rules of virtual function 127](#_Toc125917127)

[Behaviour of virtual function : 127](#_Toc125917128)

[Binding 128](#_Toc125917129)

[Early binding 128](#_Toc125917130)

[Late binding 128](#_Toc125917131)

[ C++ program to demonstrate the behaviour of virtual functions. 129](#_Toc125917132)

[Pure virtual function 130](#_Toc125917133)

[ C++ program using abstract base class to find the area of a circle and a rectangle. 131](#_Toc125917134)

[Friend function and friend class: 133](#_Toc125917135)

[ C++ program to demonstrate friend function 134](#_Toc125917136)

[ C++ program to add members of two members of two different classes using friend function 135](#_Toc125917137)

[This Pointer : 136](#_Toc125917138)

[ C++ program to demonstrate the working of “this” pointer 136](#_Toc125917139)

[Difference between abstract class and concrete class. 137](#_Toc125917140)

[Polymorphism: 138](#_Toc125917141)

**OBJECT ORIENTED PROGRAMMING**

INTRODUCTION: **<IOSTREAM> 🡪 Input Output Stream**

**Narrow characters (char);**

cin 🡪 stand input stream

cout 🡪 stand output stream

cerr 🡪 standard output stream for error

cerr 🡪 standard output stream for logging

<iomanip> *//? input output manipulator.*

**TYPES OF MANIPULATORS IN C++(few examples)**

* Dec 🡪 Used to turn on dec flag
* Hex 🡪 Used to turn on hex flag
* Oct 🡪 Used to turn on oct flag
* Boolalpha 🡪 Used to turn on boolalpha flag
* Noboolalpha 🡪 Used to turn off boolalpha flag
* setiosflags(fmtflags f) 🡪 Used to turn on flags in f
* resetiosflags (fmtflags f) 🡪 Used to turn off flags in f
* setbase(int base) 🡪 Used to set base
* endl

## DYNAMIC MEMORY ALLOCATION:

### Memory Management using New and delete keyword**.**

* C++ allow us to allocate the memory of a variable or an array in run time. This is known as dynamic memory allocation.
* In C++ we need to deallocate the dynamic allocated memory manually after we have no use for the variable.
* We can allocate and then deallocate the memory dynamically using new and delete operators respectively.

#### Syntax for new operator :

pointer\_variable = new data\_type;

#### Syntax for delete operator :

delete pointer\_variable;

## Lab Work: (New and Delete opeartor)

1. Write a C++ program to store the score of n students and display it.

*// 1. C++ program to store the score of n students and display it.*

#include <iostream>

using namespace std;

int main()

{

    int number, i;

    cout << "Enter the total number of the students : ";

    cin >> number;

    float\* p;

    p = new float[number]; *//? new is a keyword.*

    cout << "Enter the score of the students : " << endl;

    for (i = 0; i < number; i++)

    {

// cout << "Student " << i + 1 << " : ";

// cin >> p[i];

        cout << "Student " << i + 1 << " : ";

        cin >> \*(p + i);

    }

    cout << "Display the score of students : " << endl;

    for (i = 0; i < number; i++)

    {

     cout << "Student " << i + 1 << " : " << \*(p + i) << endl;

//cout << "Student " << i + 1 << " : " << p[i] << endl;

    }

    delete[] p; *//? delete is a keyword.*

    return 0;

}

**Output:**

Enter the total number of the students : 5

Enter the score of the students :

Student 1 : 98

Student 2 : 95

Student 3 : 98

Student 4 : 97

Student 5 : 100

Displaying the score of the students

Student 1 : 98

Student 2 : 95

Student 3 : 98

Student 4 : 97

Student 5 : 100

## **Lab report 1:**

### [C++ program to print the numbers from 1 to 10.](#OnePrint1to10)

### [C++ program to find the area of a circle using define directives.](#TwoArea)

### [C++ program to check the largest number between two input numbers using if else.](#ThreeLargest)

### [C++ program to print the largest number between 3 input numbers using nested if else.](#FourLargestAmongThree)

### [C++ program to illustrate setw, setprecision and setfill.](#FiveSet)

### [C++ program to illustrate implicit and explicit type conversion.](#SixTypeConversion)

### [C++ program to store the score of n students and display it.](#SevenScoreDisplay)

**Q.no. 1**

*// 1. C++ program to print from 1 to 10.*

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

    cout << ("\n1.C++ program to print from 1 to 10.\n");

    int i;

    for (i = 1; i <= 10; i++)

    {

        cout << i << setw(4);

    }

    return 0;

}

**Output:**

*1.C++ program to print from 1 to 10.*

*1   2   3   4   5   6   7   8   9  10*

*PS C:\C++\Purna\Lab1>*

**Q.no. 2**

*// 2. C++ program to find the area of a circle using define directives.*

#include <iostream>

#define pi 3.14

using namespace std;

int main()

{

    cout << "\n2. C++ program to find the area of a circle using define directives.\n";

    float radius, area;

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

    cin >> radius;

    area = pi \* radius \* radius;

    cout << "Area of a circle is " << area << " sq. units.";

    return 0;

}

**Output:**

*2. C++ program to find the area of a*

*circle using define directives.*

*Enter the radius of a circle : 5*

*Area of a circle is 78.5 sq. units.*

*PS C:\C++\Purna\Lab1>*

**Q.no.3**

*//3. C++ program to check the largest number between two numbers using if else.*

#include <iostream>

using namespace std;

int main()

{

    cout << "\n3. C++ program to check the largest number between two input numbers using if else.\n";

    int x, y, large = 0;

    cout << "Enter the two numbers : \n";

    cin >> x >> y;

    if (x > y)

    {

        cout << "Largest number among " << x << " and " << y << " is " << x << '.';

    }

    else

    {

        cout << "Largest number among " << x << " and " << y << " is " << y << '.';

    }

    return 0;

}

**Output:**

*3. C++ program to check the largest number*

*between two input numbers using if else.*

*Enter the two numbers :*

*5 8*

*Largest number among 5 and 8 is 8.*

*PS C:\C++\Purna\Lab1>*

**Q.no.4**

*// 4. C++ program to print the largest number between three input numbers using nested if else.*

#include <iostream>

using namespace std;

int main()

{

    cout << "\n4. C++ program to print the largest number between 3 input numbers using nested if else.\n";

    int x, y, z, large = 0;

    cout << "Enter the three numbers : \n";

    cin >> x >> y >> z;

    if (x > y)

    {

        if (x > z)

        {

            cout << "Largest number among " << x << ", " << y << " and " << z << " is " << x << '.';

*//? here printing x.*

        }

        else

        {

            cout << "Largest number among " << x << ", " << y << " and " << z << " is " << z << '.';

*//? here printing z.*

        }

    }

    else

    {

        if (y > z)

        {

            cout << "Largest number among " << x << ", " << y << " and " << z << " is " << y << '.';

*//? here printing y.*

        }

        else

        {

            cout << "Largest number among " << x << ", " << y << " and " << z << " is " << z << '.';

*//? here printing z.*

        }

    }

}

**Output:**

*4. C++ program to print the largest number*

*between 3 input numbers using nested if else.*

*Enter the three numbers :*

*4       7       9*

*Largest number among 4, 7 and 9 is 9.*

*PS C:\C++\Purna\Lab1>*

**Q.no.5**

*// 5. C++ program to illustrate setw, setprecision and setfill.*

#include <iostream>

#include <iomanip> *//? input output manipulator.*

using namespace std;

int main()

{

     cout<<"\n5. C++ program to illustrate setw, setprecision and setfill.\n";

     float a = 3.33333333;

     int b = 3333;

     cout << "Before using setw:" << endl;

     cout << a << endl;

     cout << "After using setw:" << endl;

     cout << setw(10) << a << endl;

     cout << "Before using setprecision:" << endl;

     cout << a << endl;

     cout << "After using setprecision:" << endl;

     cout << setprecision(3) << a << endl;

     cout << "Before setting the fill char: \n"

          << setw(10);

     cout << b << endl;

     cout << "After setting the fill char"

          << " setfill to \*: \n"

          << setfill('\*')

          << setw(10);

     cout << b << endl;

     return 0;

}

**Output:**

*5. C++ program to illustrate setw,*

*setprecision and setfill.*

*Before using setw:*

*3.33333*

*After using setw:*

*3.33333*

*Before using setprecision:*

*3.33333*

*After using setprecision:*

*3.33*

*Before setting the fill char:*

*3333*

*After setting the fill to \*:*

*\*\*\*\*\*\*3333*

*PS C:\C++\Purna\Lab1>*

**Q.no.6**

*// 6. C++ program to illustrate implicit and explicit type conversion.*

#include <iostream>

using namespace std;

int main()

{

    cout << "\n6. C++ program to illustrate implicit and explicit type conversion.\n";

    int x = 11;

    float y = 3.14;

    double number;

    number = x - y; *// implicit type conversion.*

    cout << "Implicit type conversion : " << number << endl;

    y = (double)x / 2; *// explicit type conversion.*

    cout << "Explicit type conversion : " << y;

    return 0;

}

**Output:**

*6. C++ program to illustrate implicit*

*and explicit type conversion.*

*Implicit type conversion : 7.86*

*Explicit type conversion : 5*

**Q.no. 7**

*// 7. C++ program to store the score of n students and display it.*

#include <iostream>

using namespace std;

int main()

{

    cout<<"\n7. C++ program to store the score of n students and display it.\n";

    int number, i;

    cout << "Enter the total number of the students : ";

    cin >> number;

    float \*p;

    p = new float[number]; *//? new is a keyword.*

    cout << "Enter the score of the students : " << endl;

    for (i = 0; i < number; i++)

    {

        cout << "Student " << i << " : ";

        cin >> \*(p + i);

    }

    cout << "Printing the score of students : " << endl;

    for (i = 0; i < number; i++)

    {

        cout << "Student " << i << " : " << \*(p + i) << endl;

    }

    delete[] p;

    return 0;

}

**Output:**

*7.C++ program to store the score*

*of n students and display it.*

*Enter the total number*

*of the students : 3*

*Enter the score of students:*

*Student 1 : 100*

*Student 2 : 99*

*Student 3 : 97*

*Print the score of students:*

*Student 1 : 100*

*Student 2 : 99*

*Student 3 : 97*

*PS C:\C++\Purna\Lab1>*

# **Control Statement:**

## for loop

## while loop

## do while loop

### **Class Works:**

### [C++ program to find the sum of two natural numbers using for loop.](#OneSumForLoop)

### [C++ program to find the sum of two natural numbers using while loop.](#TwoSumFWhileLoop)

### [C++ program to find the sum of two natural numbers using do while loop.](#ThreeSumDoWhileLoop)

### [C++ program to find the factorial of the given number using for loop.](#FourFactorialForLoop)

### [C++ program to find the factorial of the given number using while loop.](#FiveFactorialWhileLoop)

### [C++ program to find the factorial of the given number using do while loop.](#SixFactorialDoWhileLoop)

### [C++ program to find the fibonacci series upto n number using while loop.](#SevenFibonacciWhileLoop)

### [C++ program to find HCF/GCD.](#EightHCFForLoop)

### [C++ program to illustrate break and continue.](#NineBreakContinue)

### [C++ program to create a calculator using switch statement.](#TenSwitch)

**Q.no.1**

*// 1. C++ program to find the sum of natural numbers for loop.*

#include <iostream>

using namespace std;

int main()

{

    int i, number, sum = 0;

    cout << "Enter the natural numbers : ";

    cin >> number;

    for (i = 1; i <= number; i++)

    {

        sum += i;

    }

    cout << "The total sum of the natural numbers is " << sum << endl;

    return 0;

}

**Output:**

*Enter the natural numbers : 9*

*The total sum of the natural numbers is 45*

*PS C:\C++ class\CS\loop>*

**Q.no.2**

*// 1. C++ program to find the sum of natural numbers using while loop..*

#include <iostream>

using namespace std;

int main()

{

    int i, number, sum = 0;

    cout << "Enter the natural numbers : ";

    cin >> number;

    while (i <= number)

    {

        sum += i;

        i++;

    }

    cout << "The total sum of the natural numbers is " << sum << endl;

    return 0;

}

**Output:**

*Enter the natural numbers : 9*

*The total sum of the natural numbers is 45*

*PS C:\C++ class\CS\loop>*

**Q.no.3**

*// 1. C++ program to find the sum of natural numbers using do while loop..*

#include <iostream>

using namespace std;

int main()

{

    int i, number, sum = 0;

    cout << "Enter the natural numbers : ";

    cin >> number;

    do

    {

        sum += i;

        i++;

    } while (i <= number);

    cout << "The total sum of the natural numbers is " << sum << endl;

    return 0;

}

**Output:**

*Enter the natural numbers : 9*

*The total sum of the natural numbers is 45*

*PS C:\C++ class\CS\loop>*

**Q.no.4**

*// 1. C++ program to find the factorial using for loop.*

#include <iostream>

using namespace std;

int main()

{

    int i, number, fact = 1;

    cout << "Enter the number : ";

    cin >> number;

    if (number < 0)

    {

        cout << "Error!!";

    }

    else if (number == 0)

    {

        cout << "The factorial of " << number << " is " << fact << '.' << endl;

        exit(0);

    }

    else

    {

        for (i = 1; i <= number; i++)

        {

            fact \*= i;

        }

    }

    cout << "The factorial of " << number << " is " << fact << '.' << endl;

    return 0;

}

**Output:**

*Enter the number : 5*

*The factorial of 5 is 120.*

*PS C:\C++ class\CS\loop>*

**Q.no.5**

*// 1. C++ program to find the factorial using while loop.*

#include <iostream>

using namespace std;

int main()

{

    int i = 1, number, fact = 1;

    cout << "Enter the number : ";

    cin >> number;

    if (number < 0)

    {

        cout << "Error!!";

    }

    else if (number == 0)

    {

        cout << "The factorial of " << number << " is " << fact << '.' << endl;

        exit(0);

    }

    else

    {

        while (i <= number)

        {

            fact \*= i;

            i++;

        }

    }

    cout << "The factorial of " << number << " is " << fact << '.' << endl;

    return 0;

}

**Output:**

*Enter the number : 5*

*The factorial of 5 is 120.*

*PS C:\C++ class\CS\loop>*

**Q.no.6**

*// 1. C++ program to find the factorial using do while loop.*

#include <iostream>

using namespace std;

int main()

{

    int i = 1, number, fact = 1;

    cout << "Enter the number : ";

    cin >> number;

    if (number < 0)

    {

        cout << "Error!!";

    }

    else if (number == 0)

    {

        cout << "The factorial of " << number << " is " << fact << '.' << endl;

        exit(0);

    }

    else

    {

        do

        {

            fact \*= i;

            i++;

        } while (i <= number);

    }

    cout << "The factorial of " << number << " is " << fact << '.' << endl;

    return 0;

}

**Output:**

*Enter the number : 5*

*The factorial of 5 is 120.*

*PS C:\C++ class\CS\loop>*

**Q.no.7**

*// 7.Write a C++ program to find the fibonacci series upto n number using while loop.*

*//!Another method.*

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

    int number, f1 = 0, f2 = 1, sum = 0;

    cout << "Find the number of terms you want to print upto : ";

    cin >> number;

    cout << "The fibonacci series " << f1 << setw(3) << f2 << setw(3);

    sum = f1 + f2;

    while (sum <= number)

    {

        cout << sum << setw(3);

        f1 = f2;

        f2 = sum;

        sum = f1+ f2;

    }

    cout << setw(-3) << '.';

    return 0;

}

**Output:**

*Find the number of terms you want to print upto : 15*

*The fibonacci series 0  1  1  2  3  5  8 13.*

*PS C:\C++ class\CS\loop>*

**Q.no.8**

*// C++ program to find the HCF.*

#include <iostream>

using namespace std;

int main()

{

    int num1, num2, hcf = 1;

    cout << "Enter the two numbers whose HCF is to be found : " << endl;

    cin >> num1 >> num2;

    for (int i = 1; (i <= num1 || i <= num2); i++)

    {

        if (num1 % i == 0 && num2 % i == 0)

            hcf = i;

    }

    cout << "HCF of " << num1 << " and " << num2 << " is " << hcf;

    return 0;

}

**Output:**

*Enter the two numbers whose HCF is to be found :*

*24 30*

*HCF of 24 and 30 is 6*

*PS C:\C++ class\CS\loop>*

**Q.no.9**

*// C++ program to print from 1 to 10 and break and continue it when reached it.*

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

    int i;

    for (i = 1; i <= 10; i++)

    {

        if (i == 5)

        {

            break;

        }

        cout << i << setw(3);

    }

    cout << setw(-3) << endl;

    for (i = 1; i <= 10; i++)

    {

        if (i == 5)

        {

            continue;

        }

        cout << i << setw(3);

    }

    return 0;

}

**Output:**

*1  2  3  4*

*1  2  3  4  6  7  8  9 10*

*PS C:\C++ class\CS\switch>*

**Q.no.10**

*// C++ program to make a calculator using switch case*

#include <iostream>

using namespace std;

int main()

{

    char ch;

    int a, b;

    cout << "Enter '\*' for multiplication " << endl

         << "Enter '+' for addition " << endl

         << "Enter '-' for subtraction " << endl

         << "Enter '/' for division " << endl;

    cin >> ch;

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

    cin >> a >> b;

    switch (ch)

    {

    case '\*':

        cout << "Multiplication = " << a \* b;

        break;

    case '+':

        cout << "Addition = " << a + b;

        break;

    case '/':

        cout << "Division = " << a / b;

        break;

    case '-':

        cout << "Subtraction = " << a - b;

        break;

    default:

        cout << "Wrong Choice !!" << endl;

    }

    return 0;

}

**Output:**

*Enter '\*' for multiplication*

*Enter '+' for addition*

*Enter '-' for subtraction*

*Enter '/' for division*

*/*

*Enter the two numbers :*

*10*

*5*

*Division = 2*

*PS C:\C++ class\CS\switch>*

# **Pointer**

Pointer is a variable in C++ that holds the address of the another variable.

**For example:**

An integer type pointer can hold the address of an integer variable.

**Syntax:**

datatype \*variable\_name;

**Example:**

int \*x;

float \*y;

## Operators in pointers:

### Reference operator (&):

### Dereference operator (\*) :

*// C++ program to illustrate pointer.*

#include <iostream>

using namespace std;

int main()

{

    int x = 27;

    int \*p;

    p = &x;

    cout << "Value of x is : ";

    cout << x << endl;

    cout << "Value of p is : ";

    cout << p << endl;

    cout << "Value of \*p is : ";

    cout << \*p << endl;

    return 0;

}

**Output:**

*Value of x is : 27*

*Value of p is : 0x47ec3ff654*

*Value of \*p is : 27*

*PS C:\C++ class\CS\switch>*

# **Pointer & Arrays :**

Array & pointers work based on related concept. The array name itself denotes the base address of the array. This means that to assign the address of an array to a pointer, you should not use an ampersand(&).

Example**:**

p = arr; //correct

p = &arr; //incorrect

Example**:**

### C++ program to demonstrate the Use of Array and pointer.

*// C++ program to Array & Pointer.*

#include <iostream>

using namespace std;

int main()

{

    int \*p, i;

    int arr[] = {5, 6, 7};

    p = arr;

    for (i = 1; i <= 6; i++)

    {

        cout << \*p << ' ';

        p++;

    }

    return 0;

}

**Output:**

*5 6 7 4 -1866467352 114*

*PS C:\C++ class\CS\PointerArray>*

## **Null Pointer:**

If there is no exact address that is to be assigned then, the pointer variable can be assigned as a **NULL**.

* It should be done during declaration. Such a pointer is known as null pointer.
* Its value is zero.

**Example:**

### C++ program to demonstrate null pointer.

#include <iostream>

using namespace std;

int main()

{

    int \*p = NULL;

    cout << "Value of p is : " << p;

    return 0;

}

**Output:**

*Value of p is : 0*

*PS C:\C++ class\CS\PointerArray>*

## Function Call, Function Definition and Function Declaration Program :

*// C++ program to illustrate function call, function definition and function declaration.*

#include <iostream>

using namespace std;

void sum(int *x*, int *y*); *// function prototype*

void diff(int *x*, int *y*); *// function prototype*

void product(int *x*, int *y*); *// function prototype*

void division(int *x*, int *y*); *// function prototype*

int main()

{

    int a, b;

    cout << "Enter the value of a and b :\n";

    cin >> a >> b;

    sum(a, b); *// function call*

    diff(a, b); *// function call*

    product(a, b); *// function call*

    division(a, b); *// function call*

    return 0;

}

void sum(int *x*, int *y*) *// function defintion*

{

    cout << "The total sum of " << *x* << " and " << *y* << " is " << *x* + *y* << '.' << endl; *// function definition*

}

void diff(int *x*, int *y*) *// function defintion*

{

    cout << "The total sum of " << *x* << " and " << *y* << " is " << *x* - *y* << '.' << endl; *// function definition*

}

void product(int *x*, int *y*) *// function defintion*

{

    cout << "The total sum of " << *x* << " and " << *y* << " is " << *x* \* *y* << '.' << endl; *// function definition*

}

void division(int *x*, int *y*) *// function defintion*

{

    cout << "The total sum of " << *x* << " and " << *y* << " is " << *x* / *y* << '.' << endl; *// function definition*

}

**Output:**

*Enter the value of a and b :*

*10*

*5*

*The total sum of 10 and 5 is 15.*

*The total sum of 10 and 5 is 5.*

*The total sum of 10 and 5 is 50.*

*The total sum of 10 and 5 is 2.*

*PS C:\C++ class\CS\function>*

## Function Overloading:

Function overloading is defined as the process of having two or more functions with the same name but different in parameter.

Example**:**

*//C++ program to illustrate function overloading.*

#include <iostream>

using namespace std;

void add(int *a*, int *b*)

{

    cout << "Printing of int data type, \n"

         << "Sum = " << (*a* + *b*);

}

void add(double *a*, double *b*)

{

    cout << "\nPrinting of double data type, \n"

         << "Sum = " << (*a* + *b*);

}

int main()

{

    add(10, 2);

    add(5.3, 6.2);

    return 0;

}

**Output:**

*Printing of int data type,*

*sum = 12*

*Printing of double data type,*

*sum = 11.5*

*PS C:\C++ class\CS\function>*

# **Function**

Inline Function:In C++, we declare a function as inline. This copies the function to the location of the function call in compile time and may make the program execution faster.

* To create inline function, use the inline keyword.

***Syntax:***

*inline return\_type function\_name(parameters)*

*{*

*// function code*

*}*

***Example:***

// *C++ program to calculate the area of a circle using inline function.*

#*include* <iostream>

#*define* *pi* 3.14

using namespace std;

*inline* int *display*(int area)

{

    cout *<<* "Area of a circle : " *<<* *area << " sq. units." << end*l;

*return* 0;

}

int *main*()

{

    float radius, area = 0;

    cout *<<* "Enter the radius of the circle : ";

    cin *>>* radius;

*display* (*pi* \* radius \* radius);

*return* 0;

}

**Output:**

*Enter the radius of the circle : 10*

*Area of a circle : 314 sq. units.*

*PS C:\C++ class\CS\function\inlineFunction>*

## Default Arguments:

In C++, we provide default values for function parameters.

* If a function with default arguments is called without passing arguments, then default parameters are used. However, if arguments are passed while calling the function, the default arguments are ignored.

**Example:**

// *C++ Program to illustrate Default Arguments*

#*include* <iostream>

using namespace std;

/\*

*A function with default arguments,*

*it can be called with*

*2 arguments or 3 arguments or 4 arguments.*

\*/

int *sum*(int x, int y, int z = 0, int w = 0)

// *assigning default values to z,w as 0*

{

*return* (x + y + z + w);

}

int *main*()

{

// *Statement 1*

    cout *<<* *sum*(10, 15) *<<* *endl*;

// *Statement 2*

    cout *<<* *sum*(10, 15, 25) *<<* *endl*;

// *Statement 3*

    cout *<<* *sum*(10, 15, 25, 30) *<<* *endl*;

*return* 0;

}

**Output:**

*25*

*50*

*80*

*PS C:\C++ class\CS\function\defaultFunction>*

### Pass by reference:

When a variable is passed as a reference to a function, the address of the variable is stored in a pointer variable inside the function. Hence, the variable inside the function is an alias from the passed variable.

Therefore, any operations performed on the variable inside the function will be reflected in the calling function.

* This ability to reflect changes could return more than one value function.
* Also, a void functin could technically return values using this method.

**Example:**

//*C++ program to illustrate pass by reference.*

#*include*<iostream>

using namespace std;

void *fun*(int *&*x)

{

    x--;

}

int *main* ()

{

    int a = 5;

    cout *<<* a *<<* *endl*;

*fun*(a);

    cout *<<* a *<<* *endl*;

*return* 0;

}

**Output:**

*5*

*4*

*PS C:\C++ class\CS\function\passByReference>*

**Example:**

// *C++ program to swap two numbers using pass by reference*

#*include* <iostream>

using namespace std;

void *swap*(int *&*x, int *&*y)

{

    int z = x;

    x = y;

    y = z;

}

int *main*()

{

    int a = 45, b = 35;

    cout *<<* "Before Swap\n";

    cout *<<* "a = " *<<* a *<<* " b = " *<<* b *<<* "\n";

*swap*(a, b);

    cout *<<* "After Swap with pass by reference\n";

    cout *<<* "a = " *<<* a *<<* " b = " *<<* b *<<* "\n";

}

**Output:**

*Before Swap*

*a = 45 b = 35*

*After Swap with pass by reference*

*a = 35 b = 45*

*PS C:\C++ class\CS\function\passByReference>*

### Return by Reference:

Return by reference is very different from Call by reference. Functions behaves a very important role when variable or pointers are returned as reference.

*dataType& functionName(parameters);*

where,  
*dataType* is the return type of the function, and *parameters* are the passed arguments to it.

**Example:**

//*C++ program to illustrate the return by reference.*

#*include*<iostream>

using namespace std;

int num;//*?global variable*

int *&fun*();

int *main* ()

{

*fun*() = 6;

    cout *<<* num;

*return* 0;

}

int *&fun*()

{

*return* num;

}

*Output:*

*6*

*PS C:\C++ class\CS\function\returnByReference>*

## **Slope:**

When you declare a program element such as class function, or variable, its name can only be “seen” and used in certain parts of program. The context in which a name is visible called *scope.*

**Example:**

If you declare a variable ‘x’ within a function, x is only visible within that function body. It has local scope.

### Local scope:

### Global Scope:

### C++ program to demonstrate local and global scope of a variable.

//*C++ program to illustrate global and local variable (SCOPE)*

#*include*<iostream>

using namespace std;

//*?global variable*

int global = 5;

int *main* ()

{

//*?local variable with the same name as the global variable.*

    int global = 2;

    cout *<<* global *<<endl*;

*return* 0;

}

**Output:**

*2*

*PS C:\C++ class\CS\function\scope>*

## Class Scope:

A name declared within a member function hides a declaration of the same name whose scope extends to or past the end of the member function's class.

When the scope of a declaration extends to or past the end of a class definition, the regions defined by the member definitions of that class are included in the scope of the class. Members defined lexically outside of the class are also in this scope. In addition, the scope of the declaration includes any portion of the declarator following the identifier in the member definitions.

The name of a class member has class scope and can only be used in the following cases:

* In a member function of that class
* In a member function of a class derived from that class
* After the . (dot) operator applied to an instance of that class
* After the . (dot) operator applied to an instance of a class derived from that class, as long as the derived class does not hide the name
* After the -> (arrow) operator applied to a pointer to an instance of that class
* After the -> (arrow) operator applied to a pointer to an instance of a class derived from that class, as long as the derived class does not hide the name
* After the :: (scope resolution) operator applied to the name of a class
* After the :: (scope resolution) operator applied to a class derived from that class

**Example:**

### C++ program to find the area of a rectangle using class scope.

//*C++ program to illustrate class scope.*

#*include* <iostream>

using namespace std;

class A

{

private:// *specific to this class and can be used under this class(Unavailable to any other class or function)*

    float length, breadth;// *can be used only throughout this class A*

public:// *can be used anywhere in program when declared public*

    void *getLen*()

    {

        cout *<<* "Enter the length : ";

        cin *>>* length;// *takes it from this class and can be used by functions under this class only*

    }

    void *getBre*()

    {

        cout *<<* "Enter the breadth : ";

        cin *>>* breadth;// *takes it from this class and can be used by functions under this class only*

    }

    float *area*()

    {

*return* length \* breadth;

    }

    void *displayArea*()

    {

        cout *<<* "The area of the rectangle is " *<<* *area*() *<<* " sq units." *<<* *endl*;

    }

};

int *main*()

{

    A area;

    area.*getLen*();// *Public function with private parameters*

    area.*getBre*();

    area.*displayArea*();

}

**Output:**

*Enter the length : 5*

*Enter the breadth : 10*

*The area of the rectangle is 50 sq units.*

*PS C:\C++ class\CS\function\scope>*

## File Scope:

If the declarator or type specifier that declares the identifier appears outside of any block or list of parameters, the identifier has file scope, which terminates at the end of the translation unit.

This scope consists of local scope as

The variables defined with the file scope are known as global variable.

**Example:**

// *C++ program to illustrate the file scope.*

#*include* <iostream>

using namespace std;

int d;

void *display*();

int *main*()

{

    d = 5;

    cout *<<* "in main function(), ";

    cout *<<* "The value of d is " *<<* d *<<* *endl*;

*display*();

*return* 0;

}

void *display*()

{

    cout *<<* "in void display(), ";

    cout *<<* "The value of d is " *<<* d *<<* *endl*;

}

**Output:**

*in main function(), The value of d is 5*

*in void display(), The value of d is 5*

*PS C:\C++ class\CS\function\scope>*

### Take two inputs from the user, if both of the numbers are equal find its sum and if not then display their products.

// *Take two inputs from the user, if both of the numbers are equal find its sum and if not then display their products in OOP?*

#*include* <iostream>

using namespace std;

int x, y;//*?global variable*

void *sum*();

void *product*();

int *main*()

{

    cout *<<* "Enter the value of x and y : \n";

    cin *>>* x *>>* y;

*if* (x != y)

    {

*product*();

    }

*else*

    {

*sum*();

    }

*return* 0;

}

void *sum*()

{

    int sum = x + y;//*?local variable*

    cout *<<* "Sum of " *<<* x *<<* " and " *<<* y *<<* " is " *<<* sum;

}

void *product*()

{

    int product = x \* y;//*?local varia*

    cout *<<* "Product of " *<<* x *<<* " and " *<<* y *<<* " is " *<<* product;

}

**Output:**

*Enter the value of x and y :*

*5*

*10*

*Product of 5 and 10 is 1*

*Enter the value of x and y :*

*5*

*10*

*Product of 5 and 10 is 50*

*PS C:\C++ class\CS\Question>*

# Chapter:3

# **Class and Objects**

## What is class?

* Class is a blueprint that defines the variables and methods common to all objects of a certain kind.
* Class is user defined data types that holds **data members** and **member function**, which can be accessed & used by creating instances of objects.

**Example:**

### C++ program to illustrate class & objects.

// *C++ program to illustrate class and objects.*

#*include* <iostream>

using namespace std;

class A

{

public:

    int b;

    int *display*()

    {

        cout *<<* "The value of b is " *<<* b *<<* *endl*;

*return* 0;

    }

}; //or you can declare obj here as well -> }obj;

int *main*()

{

    A obj; //this line is not needed if you declared obj above.

    obj.b = 5;

    obj.*display*();

*return* 0;

}

**Output:**

*The value of b is 5*

*PS C:\C++ class\CS\ClassObjects>*

## C++ program to find the area of a rectangle and volume using class and objects.

// *C++ program to find the area of a rectangle and volume using class and objects.*

#*include* <iostream>

using namespace std;

class Room

{

public:

    double length, breadth, height;

    double *calculateArea*()

    {

        double area = length \* breadth;

        cout *<<* "Area of a rectangle is " *<<* area *<<* " sq. units." *<<* *endl*;

*return* 0;

    }

    double *calculateVolume*()

    {

        double volume = length \* breadth \* height;

        cout *<<* "Volume of a cuboid is " *<<* volume *<<* " cubic units." *<<* *endl*;

*return* 0;

    }

};

int *main*()

{

    Room room1;

    cout *<<* "Enter the value of length, breadth and height" *<<* *endl*;

    cin *>>* room1.length *>>* room1.breadth *>>* room1.height;

    room1.*calculateArea*();

    room1.*calculateVolume*();

*return* 0;

}

**Output:**

*Enter the value of length, breadth and height*

*5.5*

*4.5*

*3.5*

*Area of a rectangle is 24.75 sq. units.*

*Volume of a cuboid is 86.625 cubic units.*

*PS C:\C++ class\CS\ClassObjects>*

## C++ program to find the area of a rectangle and volume of three boxes using class and objects.

// *C++ program to find the area of a rectangle and volume using class and objects of three boxes.*

#*include* <iostream>

using namespace std;

class Box

{

public:

    double length, breadth, height;

    double *calculateArea*()

    {

        double area = length \* breadth;

        cout *<<* "\tArea of a rectangle is " *<<* area *<<* " sq. units." *<<* *endl*;

*return* 0;

    }

    double *calculateVolume*()

    {

        double volume = length \* breadth \* height;

        cout *<<* "\tVolume of a cuboid is " *<<* volume *<<* " cubic units." *<<* *endl*;

*return* 0;

    }

};

int *main*()

{

    int i;

    Box box1, box2, box3;

    Box box[i];

// *prints upto the enter value of length, breadth and height.*

*for* (i = 0; i < 3; i++)

    {

        cout *<<* "For box : " *<<* i + 1 *<<* *endl*;

// *cout << "\tEnter the size of length, breadth and height : " << endl;*

// *cin >> box[i].length >> box[i].breadth >> box[i].height;*

        cout *<<* "\tEnter the size of length : \t";

        cin *>>* box[i].length;

        cout *<<* "\tEnter the size of breadth : \t";

        cin *>>* box[i].breadth;

        cout *<<* "\tEnter the size of height : \t";

        cin *>>* box[i].height;

    }

// *prints the calculation after values are entered.*

*for* (i = 0; i < 3; i++)

    {

        cout *<<* "For box : " *<<* i + 1 *<<* *endl*;

        box[i].*calculateArea*();

        box[i].*calculateVolume*();

    }

*return* 0;

}

**Output:**

*For box : 1*

*Enter the size of length :      1*

*Enter the size of breadth :     2*

*Enter the size of height :      3*

*For box : 2*

*Enter the size of length :      4*

*Enter the size of breadth :     5*

*Enter the size of height :      6*

*For box : 3*

*Enter the size of length :      7*

*Enter the size of breadth :     8*

*Enter the size of height :      9*

*For box : 1*

*Area of a rectangle is 2 sq. units.*

*Volume of a cuboid is 6 cubic units.*

*For box : 2*

*Area of a rectangle is 20 sq. units.*

*Volume of a cuboid is 120 cubic units.*

*For box : 3*

*Area of a rectangle is 56 sq. units.*

*Volume of a cuboid is 504 cubic units.*

*PS C:\C++ class\CS\ClassObjects>*

## Scope Resolution Operator(::)

If the member function is defined inside the class definition it can be defined directly but if it is defined outside the class then, we have to use scope resolution(::) operator along with the class name along with the function name.

**Example:**

### C++ program to illustrate scope resolution(::) operator

// *C++ program to illustrate scope resolution (::)*

#*include* <iostream>

using namespace std;

class cube

{

public:

    double side;

    double *getVolume*();

};

double cube::*getVolume*()

{

*return* side \* side \* side;

}

int *main*()

{

    cube volume;

    cout *<<* "Enter the side : ";

    cin *>>* volume.side;

    cout *<<* "Volume of the given side is : " *<<* volume.*getVolume*() *<<* " cubic units." *<<* *endl*;

*return* 0;

}

**Output:**

*Enter the side : 5.5*

*Volume of the given side is : 166.375 cubic units.*

*PS C:\C++ class\CS\scopeResolution>*

## **Constructor:**

Constructor is a special member function of a class that initializes the object of the class. Constructor name is same as the class name & it doesn’t have return type.

* Constructor is a special type of member function that is called automatically when object is created.

**Example:**

class CLASSNAME

{

    public : *CLASSNAME*([parameter\_list]) //?constructor definition

    {

// *object initialization*

    }

};

## Types of Constructor

### Default constructor

* A constructor to which no arguments are passed is called the Default constructor. It is also called a constructor with no parameters.

**Example:**

// *C++ program to illustrate default constructor.*

#*include* <iostream>

using namespace std;

class cube

{

public:

    double side;

*cube*()

    {

        side = 5.5;

    }

};

int *main*()

{

    cube c;

    cout *<<* c.side *<<* " cm " *<<* *endl*;

*return* 0;

}

### Parameterized constructor

* Constructor with parameters.
* Using this constructor you can provide different values to data members of different objects by passing the appropriate values as an arguments.

**Example:**

// *C++ program to illustrate parametrized constructor.*

#*include* <iostream>

using namespace std;

class cube

{

public:

    double side;

*cube*(float x)

    {

        side = x;

    }

};

int *main*()

{

    cube *a*(10);

    cube *b*(20);

    cout *<<* "The size of side a is " *<<* a.side *<<* "cm." *<<* *endl*;

    cout *<<* "The size of side b is " *<<* b.side *<<* "cm." *<<* *endl*;

*return* 0;

}

**Output:**

The size of side a is 10cm.

The size of side b is 20cm.

PS C:\C++ class\CS\constructor>

### Copy constructor

* It is a type of constructor which is used to create a copy of already existing object of a class type.
* The compile provides a default copy constructor to all the classes.

**Syntax:**

*classname(const classname &obj)*

*{  
 //body of constructor.  
}*

**Example:**

## C++ program to display the roll number and student name using default constructor

// *C++ program to display the roll number and student name using default constructor*

#*include* <iostream>

#*include* <string.h>

using namespace std;

class Student //*?student is a class.*

{

    char \_name[10];

    int \_id, \_rollno;

public:

*Student*(char name[10], int id, int rollno)

    {

*strncpy\_s*(\_name, name, 9);

        \_id = id;

        \_rollno = rollno;

        cout *<<* "Student Data :" *<<* *endl*;// *cout << "\student data: ";*

        cout *<<* "The name of the student is: " *<<* \_name *<<* *endl*;

        cout *<<* "The id of the student is: " *<<* \_id *<<* *endl*;

        cout *<<* "The roll no of the student is: " *<<* \_rollno *<<* *endl*;

        cout *<<* '\n';

    }

*const* char *\*getName*() { *return* \_name; }

*const* int *getId*() { *return* \_id; }

*const* int *getRollNo*() { *return* \_rollno; }

};

int *main*()

{

    int id, rollno;

    char name[10];

    Student \*s[2];

// *for (int i{0}; i < 2; i++)*

*for* (int i = 0; i < 2; i++)

    {

// *cout << "enter the details of the student " << i << endl;*

        cout *<<* "Enter the id of the student: ";

        cin *>>* id;

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

        cin *>>* name;

        cout *<<* "Enter the roll no of the student: ";

        cin *>>* rollno;

        s[i] = new *Student*(name, id, rollno);

        cout *<<* "Stored data -> Id: "

*<<* s[i]->*getId*()

*<<* ", Name: "

*<<* s[i]->*getName*()

*<<* ", Roll no: "

*<<* (\*s[i]).*getRollNo*()

*<<* *endl*

*<<* *endl*;

    }

*return* 0;

}

**Output:**

Enter the id of the student: 1

Enter the name of the student: Purna

Enter the roll no of the student: 21

Student Data :

The name of the student is: Purna

The id of the student is: 1

The roll no of the student is: 21

Stored data -> Id: 1, Name: Purna, Roll no: 21

Enter the id of the student: 2

Enter the name of the student: Shrestha

Enter the roll no of the student: 22

Student Data :

The name of the student is: Shrestha

The id of the student is: 2

The roll no of the student is: 22

Stored data -> Id: 2, Name: Shrestha, Roll no: 22

PS C:\C++ class\CS\constructor>

## Types of copy Constructor:

### Shallow Copy Constructor:

**Example:**

// *C++ program to create a shallow copy constructor.*

#*include* <iostream>

#*include* <cstring>

using namespace std;

class copyConstructor

{

    char \*s\_copy;

public:

*copyConstructor*(*const* char *\**str)

    {

        s\_copy = new char[16];

*strcpy*(s\_copy, str);

    }

    void *concatenate*(*const* char *\**str)

    {

*strcat*(s\_copy, str);

    }

    void *display*()

    {

        cout *<<* s\_copy *<<* *endl*;

    }

};

int *main*()

{

    copyConstructor *c1*("Copy");

    copyConstructor c2 = c1;

    c1.*display*();

    c2.*display*();

    c1.*concatenate*("constructor");

    c1.*display*();

    c2.*display*();

*return* 0;

}

**Output:**

Copy

Copy

Copyconstructor

Copyconstructor

PS C:\C++ class\CS\constructor\shallow>

### Deep Copy Constructor:

Destructor:A destructor is a member function that is invoked automatically when object goes out of the scope.

*Or Explicity destroyed by a call to ‘delete’.*

**Syntax:**

*class name  
{  
 public:*

*name()  
{*

*//constructor statements.  
}*

*~name()  
{*

*//destructor name.  
}*

*};*

### C++ program to find the area of a circle using destructor.

// *C++ program to find the area of a circle using destructor.*

#*include* <iostream>

#*define* *pi* 3.14

using namespace std;

class Circle

{

private:

    float radius;

public:

*Circle*(float r)

    {

        radius = r;

    }

    float *area*()

    {

*return* *pi* \* radius \* radius;

    }

*~Circle*()

    {

        cout *<<* "\nThis is a Destructor.";

    }

};

int *main*()

{

    Circle *obj*(7.5);

    cout *<<* "Area of circle is " *<<* obj.*area*() *<<* " sq units.";

*return* 0;

}

**Output:**

Area of circle is 176.625 sq units.

This is a Destructor.

PS C:\C++ class\CS\ClassObjects\destructor>

### **Q. Differentiate between constructor and member function with example.**

## **Object as a function Arguments:**

*function\_name(object\_name);*

**Example:**

### C++ program to demonstrate area of a rectangle using object as a function arguments.

// *C++ program to demonstrate area of a rectangle using object as a function arguments.*

#*include* <iostream>

using namespace std;

class rect

{

private:

    float length, breadth, area;

public:

    void *set*(float x, float y)

    {

        length = x;

        breadth = y;

    }

    void *area\_circle*(rect obj1)

    {

        area = obj1.length \* obj1.breadth;

        cout *<<* "Area of a rectangle is " *<<* area *<<* " sq. units" *<<* *endl*;

    }

};

int *main*()

{

    rect r1;

    r1.*set*(10.5, 5.5);

    r1.*area\_circle*(r1);

*return* 0;

}

**Output:**

Area of a rectangle is 57.75 sq. units

PS C:\C++ class\CS\objectasFunctionArguments>

## Structure in C++

**Example:**

**Example:**

## Static data members:

**Syntax:**

*static data\_type data\_member\_name;*

**Example:**

# Unit: 4

# Operator Overloading

* An overloaded operator is called an operator function. Declare an operator function with keyword operator preceding the operator.
* Overloaded operators are distinct from overloaded functions, but like overloaded function, they are distinguished by number and types of operands used with the operator.

**Syntax:**

*return\_type classname:: operator keyOperatorSymbol(argument\_list)  
{  
 //function body  
}*

* Operator overloading can be done by implementing a function which can be

1. Member function
2. Non-member function
3. Friend function

* Operator overloading function can be a member function if the left operand is an object of that class.
* But if the left operand is different, then operator overloading must be non-member function.
* Operator overloading function can be made friend function if it needs access to the private & protected member of that class.
* There are few operator which cannot be overloaded.

They are:

1. Scope operator ‘ :: ’
2. Sizeof operator ‘ sizeof ’
3. Member function ‘ . ’
4. Member pointer selector ‘ \* ’
5. Ternary operator ‘ ?: ’

**Restrictions on Operator Overloading:**

* Precedence & Associativity of an operator cannot be changed.
* Arity(number of operands) cannot be changed unary operator remain unary binary remains binary etc.
* No new operator can be created, only can be overloaded.
* Cannot redefine the remeaning of a procedure.
* You cannot overload the preprocessor symbols #.

## **Overview of Unary & binary operator overloading:**

Unary Operator Overloading:As the name suggests, unary operators operate on single operand or data.

### **Example of Unary Operator:**

### Unary plus:

**Example:**

// *C++ program to illustrate unary plus.*

#*include* <iostream>

using namespace std;

int *main*()

{

    int y = 10;

    cout *<<* "y is " *<<* y;

*return* 0;

}

**Output:**

y is 10

PS C:\C++ class\operatorOverloading>

### Unary minus:

**Example:**

//*C++ program to illustrate unary minus.*

#*include*<iostream>

using namespace std;

int *main*()

{

    int y;

    float x = 3.14;

    y = (int)x;

    x = -x;

    cout *<<* "x is " *<<* x *<<* *endl*;

    cout *<<* "y is " *<<* y *<<* *endl*;

*return* 0;

}

**Output:**

x is -3.14

y is 3

PS C:\C++ class\operatorOverloading>

### **Example of Decrement/increment Operator:**

### Decrement Operator:

This operator is denoted by –- operator.

**Example:**

// *C++ program to illustrate the increment operator.*

#*include* <iostream>

using namespace std;

int *main*()

{

    int p = 10;

    int q = 20;

    cout *<<* "Decrement" *<<* *endl*;

    cout *<<* "p is " *<<* --p *<<* *endl*;

    cout *<<* "q is " *<<* q-- *<<* *endl*;

*return* 0;

}

### Increment Operator:

This operator is denoted by ++ operator.

**Example:**

// *C++ program to illustrate the increment operator.*

#*include* <iostream>

using namespace std;

int *main*()

{

    int p = 10;

    int q = 20;

    cout *<<* "Increment" *<<* *endl*;

    cout *<<* "p is " *<<* ++p *<<* *endl*;

    cout *<<* "q is " *<<* q++ *<<* *endl*;

*return* 0;

}

### **Example of Address Operator:**

### Address of the operator:

The address of the operator is denoted by the symbol ‘&’. It returns the address of any variable.

**Example:**

// *C++ program to illustrate the address operator.*

#*include* <iostream>

using namespace std;

int *main*()

{

    int x, y = 15;

    cout *<<* "Value of x is " *<<* x *<<* *endl*;

    cout *<<* "Address of y is " *<<* &y *<<* *endl*;

*return* 0;

}

**Example:**

Value x is 0

Address y is 0xc8ecdffc78

PS C:\C++ class\operatorOverloading>

### **Example of sizeof Operator:**

The size of the operator is denoted by the symbol “sizeof”. This returns the variable and object occupied size. This returns the size of any data types.

**Example:**

//*C++ program to illustrate sizeof operator.*

#*include* <iostream>

using namespace std;

int *main*()

{

    int p = 10;

    cout *<<* "The size of the p is " *<<* sizeof(p);

*return* 0;

}

**Output:**

The size of the p is 4

PS C:\C++ class\operatorOverloading>

### **Example of logic not Operator:**

It is denoted by the symbol(|). This operator reverses the meaning of its operand.

**Example:**

// *C++ program to illustrate logic not operator.*

#*include* <iostream>

using namespace std;

int *main*()

{

    int a = 10;

*if* (!a)

        cout *<<* "a is 0";

*else*

        cout *<<* "a is not 0";

*return* 0;

}

**Output:**

a is not 0

PS C:\C++ class\operatorOverloading>

### **Example of Bitwise not/ One’s complement Operator:**

It is denoted by ‘~’ symbol. This operator yields the bitwise one’s complement of the operand.

**Example:**

// *C++ program to illustrate bitwise operator.*

#*include* <iostream>

using namespace std;

int *main*()

{

    int a = 5, b = 9;

    cout *<<* "a&b = " *<<* (a & b) *<<* *endl*;

    cout *<<* "a|b = " *<<* (a | b) *<<* *endl*;

    cout *<<* "a^b = " *<<* (a ^ b) *<<* *endl*;

    cout *<<* "~a = " *<<* (~a) *<<* *endl*;

    cout *<<* "a<<1 = " *<<* (a << 1) *<<* *endl*;

    cout *<<* "b>>1 = " *<<* (b >> 1) *<<* *endl*;

// *return 0;*

// *unsigned short x = 0xFFFF;*

// *cout << hex << x << endl;*

// *x = ~x;*

// *cout << hex << x << endl;*

*return* 0;

}

**Output:**

*a&b = 1*

*a|b = 13*

*a^b = 12*

*~a = -6*

*a<<1 = 10*

*b>>1 = 4*

*PS C:\C++ class\operatorOverloading>*

### C++ program to demonstrate the use of unary operator overloading, increment++ and decrement-— operator overloading.

// *C++ program to demonstrate the use of unary operator overloading, increment++ and decrement-- operator overloading.*

#*include* <iostream>

using namespace std;

class count

{

public:

    int countplus;

    int countminus;

*count*()

    {

        countplus = 0;

        countminus = 0;

    }

    void *operator++*()

    {

        ++countplus;

    }

    void *operator--*()

    {

        --countminus;

    }

};

int *main*()

{

// *count c1;*

// *++c1;*

// *++c1;*

// *++c1;*

// *--c1;*

// *cout << "The value of countplus is " << c1.countplus << endl;*

// *cout << "The value of countminus is " << c1.countminus << endl;*

// *return 0;*

    count x, y;

    cout *<<* "x = " *<<* x.countplus *<<* *endl*;

    cout *<<* "y = " *<<* y.countminus *<<* *endl*;

*++*x;

*--*y;

    cout *<<* "After increment " *<<* *endl*;

    cout *<<* "x = " *<<* x.countplus *<<* *endl*;

    cout *<<* "y = " *<<* y.countminus *<<* *endl*;

*return* 0;

}

**Output:**

*x = 0*

*y = 0*

*After increment*

*x = 1*

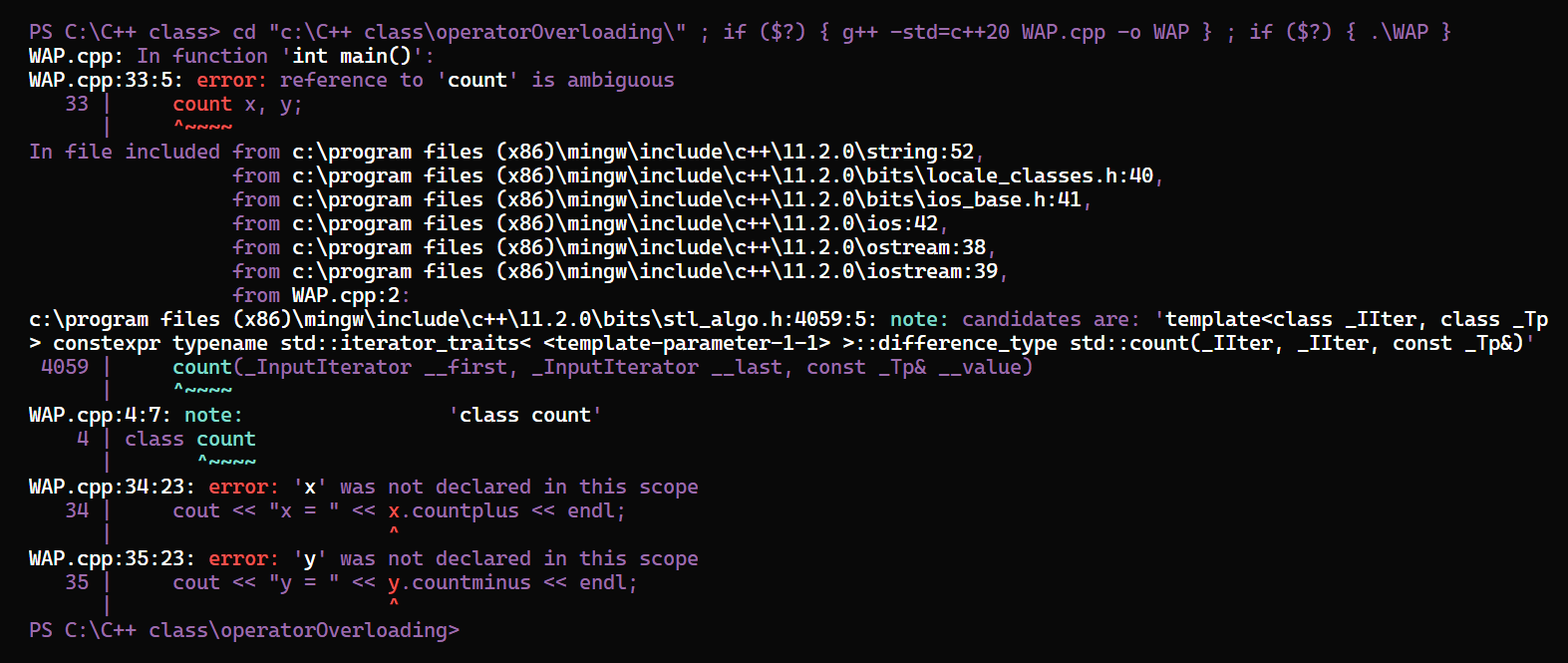
*y = -1*

--------------------------------

Process exited after 0.09975 seconds with return value 0

Press any key to continue . . .

**NOTE: same code didn’t work on my VS Code whereas it worked on Dev-C++.**



## **Binary Operator Overloading:**

Those operators which operates on two operands or data are called binary operators.

### C++ program to demonstrate the binary operator overloading.

// *C++ program to demonstrate the binary operator overloading.*

#*include* <iostream>

using namespace std;

class complex

{

private:

    int real, img;

public:

    void *getValue*()

    {

        cout *<<* "Enter the value of the real number : " *<<* *endl*;

        cin *>>* real;

        cout *<<* "Enter the value of the imaginary number : " *<<* *endl*;

        cin *>>* img;

    }

    complex *operator+*(complex obj)

    {

        complex temp;

        temp.real = real + obj.real;

        temp.img = img + obj.img;

*return* temp;

    }

    void *display*()

    {

        cout *<<* real *<<* " + "

*<<* "(" *<<* img *<<* ")"

*<<* "i" *<<* *endl*;

// *<< "," << endl;*

    }

};

int *main*()

{

    complex c1, c2, c3;

    c1.*getValue*();

    c2.*getValue*();

    c3 *=* c1 *+* c2;

    cout *<<* "Result is : ";

    c3.*display*();

*return* 0;

}

**Output:**

*Enter the value of the real number :*

*1*

*Enter the value of the imaginary number :*

*-1*

*Enter the value of the real number :*

*2*

*Enter the value of the imaginary number :*

*-2*

*Result is : 3 + (-3)i*

*PS C:\C++ class\operatorOverloading>*

## **Assignment Operator Overloading:**

To perform the operation on user defined data. All values of one object can be copied to another object.

**Syntax:**

*return\_type::operator=(parameter\_list)  
{  
 //statements to be executed.  
}*

### C++ program to demonstrate the assignment operator overloading.

// *C++ program to demonstrate the assignment operator overloading.*

#*include* <iostream>

using namespace std;

class employee

{

public:

    int salary;

*employee*(int sal)

    {

        salary = sal;

    }

    employee *operator=*(employee n)

    {

        employee temp = n.salary;

*return* temp;

    }

};

int *main*()

{

// *employee e1(1000);*

// *employee e2(2000);*

// *e1 = e2;*

// *cout << "e1.salary = " << e1.salary << endl;*

// *cout << "e2.salary = " << e2.salary << endl;*

// *return 0;*

    employee *e1*(200);

    employee *e2*(500);

    employee e3 = e1;

    cout *<<* "e3.salary = " *<<* e3.salary *<<* *endl*;

*return* 0;

}

**Output:**

*e3.salary = 200*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\C++\operatorOverloading>*

## **Arithmetic Operators Overloading (Binary Operators): [ +, - , \*, /]**

### C++ program to overload the binary operator + to add two complex numbers.

// *C++ program to overload the binary operator + to add two complex numbers.*

#*include* <iostream>

using namespace std;

class complex

{

    int n1, n2;

public:

    void *getData*()

    {

        cout *<<* "Enter the first number: ";

        cin *>>* n1;

        cout *<<* "Enter the second number: ";

        cin *>>* n2;

    }

    complex *operator+*(complex obj)

    {

        complex temp;

        temp.n1 = n1 + obj.n1;

        temp.n2 = n2 + obj.n2;

*return* temp;

    }

    void *display*()

    {

        cout *<<* "The sum of two complex numbers is: " *<<* n1 *<<* " + " *<<* n2 *<<* "i" *<<* *endl*;

    }

};

int *main*()

{

    complex c1, c2, c3;

    c1.*getData*();

    c2.*getData*();

    c3 *=* c1 *+* c2;

    c3.*display*();

*return* 0;

}

**Example:**

*Enter the first number: 3*

*Enter the second number: 6*

*Enter the first number: 4*

*Enter the second number: 8*

*The sum of two complex numbers is: 7 + 14i*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\C++\operatorOverloading>*

## **Comparision Operators Overloading(Binary)**

**i.e** < , > , ==, != , <= , >=

### C++ program to demonstrate comparision operator.

// *C++ program to demonstrate comparision operator.*

#*include* <iostream>

using namespace std;

class priceComparision

{

private:

    int price;

public:

    void *getPrice*()

    {

        cout << "Enter the price: ";

        cin >> price;

    }

    bool operator!=(priceComparision obj)

    {

*if* (price != obj.price)

*return* true;

*else*

*return* false;

    }

    void *display*()

    {

        cout << "The price is: " << price << endl;

    }

};

int *main*()

{

    priceComparision p1, p2;

    p1.*getPrice*();

    p2.*getPrice*();

*if* (p1 != p2)

        cout << "The prices are not equal." << endl;

*else*

        cout << "The prices are equal." << endl;

*return* 0;

}

// *#include <iostream>*

// *using namespace std;*

// *class priceComparision*

// *{*

// *private:*

// *int price;*

// *public:*

// *void getData()*

// *{*

// *cin >> price;*

// *}*

// *bool operator==(const priceComparision &p)*

// *{*

// *if (price == p.price)*

// *return true;*

// *else*

// *return false;*

// *}*

// *};*

// *int main()*

// *{*

// *priceComparision p1, p2;*

// *// p1.getData();*

// *// p2.getData();*

// *cout << "Enter the price of the first products : " << endl;*

// *// cin >> p1.getData();*

// *cout << "Enter the price of the second products : " << endl;*

// *// cin >> p2.getData();*

// *if (p1 == p2)*

// *cout << "The prices are equal." << endl;*

// *else*

// *cout << "The prices are not equal." << endl;*

// *return 0;*

// *}*

**Output:**

*Enter the price: 20*

*Enter the price: 50*

*The prices are not equal.*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\C++\operatorOverloading>*

## **Possible type conversion:**

There are three types of possible type conversion.

1. Conversion from basic type to class type.
2. Conversion from class type to basic type.
3. Conversion from one class type to another class type.

**Explannation:**

### **Conversion from basic type to class type:**

* This type of conversion is possible in two ways.
  + using constructor.
  + Using operator overloading.

**Using Constructor:**

* C++ program to convert minutes into minutes and hours using “constructor”.(basic type to class type)

// *C++ program to demonstrate "using constructor"*

#*include* <iostream>

using namespace std;

class Time

{

    int hrs, min;

public:

*Time*(int);

    void *display*();

};

Time::*Time*(int t)

{

    cout *<<* "Basic Type to Class Type " *<<* *endl*;

    hrs = t / 60;

    min = t % 60;

}

void Time::*display*()

{

    cout *<<* hrs *<<* "hrs" *<<* *endl*;

    cout *<<* min *<<* "min" *<<* *endl*;

}

int *main*()

{

    int dur;

    cout *<<* "Enter the time duration in minutes : ";

    cin *>>* dur;

    Time t1 = dur;

    t1.*display*();

*return* 0;

}

**Output:**

*Enter the time duration in minutes : 255*

*Basic Type to Class Type*

*4hrs*

*15min*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\C++\typeConversion >*

**Assignments: to convert inch feet , to convert Celsius Fah**

## C++ program to convert inch into inch and feet using “constructor”.

## C++ program to convert Celsius into Fahrenheit and Celsuis using “constructor”.

### **Conversion from class type to basic type.**

**Syntax:**

*operator typename ()  
{  
 //body of statement.  
}*

## C++ program to convert hours and minutes into the minutes.

// *C++ program to demonstrate the class type to basic type conversion.*

#*include* <iostream>

using namespace std;

class Time

{

    int hrs, min;

public:

*Time*(int, int);

    operator int();

};

Time::*Time*(int a, int b)

{

    hrs = a;

    min = b;

}

Time::operator int()

{

*return* (hrs \* 60 + min);

}

int *main*()

{

    int h, m, duration;

    cout *<<* "Enter the time in hours : ";

    cin *>>* h;

    cout *<<* "Enter the time in minutes : ";

    cin *>>* m;

    Time *t*(h, m);

    duration = t;

    cout *<<* "Total minutes: " *<<* duration *<<* *endl*;

*return* 0;

}

***Output:***

*Enter the time in hours : 5*

*Enter the time in minutes : 45*

*Total minutes: 345*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\C++\typeConversion >*

## C++ program to convert inch and feet into the inch.

// *C++ program to convert inch and feet into the inch using class type to basic type conversion.*

#*include* <iostream>

using namespace std;

class convert

{

    int inch, feet;

public:

*convert*(int, int);

    operator int();

};

convert::*convert*(int a, int b)

{

    inch = a;

    feet = b;

}

convert::operator int()

{

*return* (feet \* 12 + inch);

}

int *main*()

{

    int f, i, distance;

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

    cin *>>* i;

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

    cin *>>* f;

    convert *d*(i, f);

    distance = d;

    cout *<<* "Total distance in inch : " *<<* distance *<<* *endl*;

*return* 0;

}

***Output:***

*Enter the distance in inch : 32*

*Enter the distance in feet : 5*

*Total distance in inch : 92*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\C++\typeConversion>*

### **Conversion from one class type to another class type.**

Conversion from one class to another class can be performed either by using constructor or type conversion function.

**Example:**

## C++ program to convert class time to another minute class.

// *C++ program to convert class time to another minute class.*

#*include* <iostream>

using namespace std;

class Time

{

    int hrs, min;

public:

*Time*(int h, int m)

    {

        hrs = h;

        min = m;

    }

    int *getMin*()

    {

*return* (hrs \* 60 + min);

    }

    void *display*()

    {

        cout *<<* "Time in hours : " *<<* hrs *<<* *endl*;

        cout *<<* "Time in minutes : " *<<* min *<<* *endl*;

    }

};

class minute

{

    int min;

public:

*minute*()

    {

        min = 0;

    }

    void *operator=*(Time t)

    {

        min = t.*getMin*();

    }

    void *display*()

    {

        cout *<<* "Total time in minutes is : " *<<* min *<<* *endl*;

    }

};

int *main*()

{

    int h, m;

    int t1, t2;

    cout *<<* "Enter the time in hours : ";

    cin *>>* h;

    cout *<<* "Enter the time in minutes : ";

    cin *>>* m;

    Time *t*(h, m);

    t.*display*();

    minute m1;

    m1 *=* t;

    m1.*display*();

*return* 0;

}

**Output:**

*Enter the time in hours : 3*

*Enter the time in minutes : 45*

*Time in hours : 3*

*Time in minutes : 45*

*Total time in minutes is : 225*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\C++\typeConversion>*

## C++ program to convert “basic to class” type using “operator overloading” to convert minutes into minutes and hours.

// *C++ program to convert basic to class type using operator overloading to convert minutes into minutes and hours.*

#*include* <iostream>

using namespace std;

class Time

{

    int hrs, min;

public:

// *Time()*

// *{*

// *hrs = 0;*

// *min = 0;*

// *}*

    void *operator=*(int);

// *void operator=(const &time);*

    void *display*();

};

void Time::*operator=*(int t)

{

    cout *<<* "Basic Type to Class Type " *<<* *endl*;

    hrs = t / 60;

    min = t % 60;

}

void Time::*display*()

{

    cout *<<* hrs *<<* "hrs" *<<* *endl*;

    cout *<<* min *<<* "min" *<<* *endl*;

}

int *main*()

{

    int dur;

    cout *<<* "Enter the time duration in minutes : ";

    cin *>>* dur;

    Time t1;

    t1 *=* dur;

    t1.*display*();

*return* 0;

}

**Output:**

*Enter the time duration in minutes : 350*

*Basic Type to Class Type*

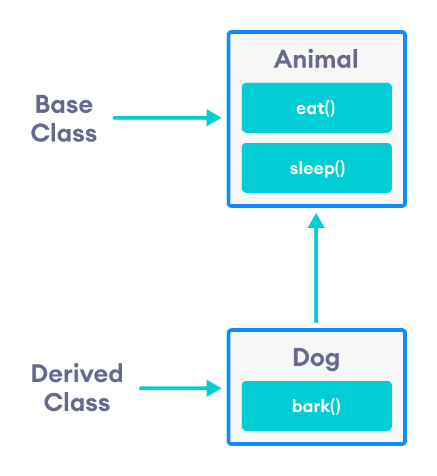
*5hrs*

*50min*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\C++\question>*

# Unit: 5

# **Inheritance**

 **Syntax:**

*class base  
 {  
 //statements;  
 };*

*class derived\_classname:Accessifier baseclassname  
 {  
 //statements;  
 };*

## C++ program to illustrate the inheritance by giving an example to find an area and a volume of a cuboid.

// *C++ program to illustrate inheritance by giving an example to find an area and a volume of a cuboid.*

#*include* <iostream>

using namespace std;

class cuboid

{

public:

    int length, breadth, area;

    void *\_area*()

    {

        area = length \* breadth;

        cout *<<* "Area = " *<<* area *<<* " sq. units." *<<* *endl*;

    }

// *void \_volume()*

// *{*

// *int height, volume;*

// *volume = area \* height;*

// *cout << "Volume = " << volume;*

// *}*

};

class cuboid1 : public cuboid

{

public:

    int height, volume;

    void *\_volume*()

    {

        volume = area \* height;

        cout *<<* "Volume = " *<<* volume *<<* " cubic units." *<<* *endl*;

    }

};

int *main*()

{

    cuboid1 c1;

    c1.length = 10;

    c1.breadth = 20;

    c1.height = 30;

    c1.*\_area*();

    c1.*\_volume*();

*return* 0;

}

**Output:**

*Area = 200 sq. units.*

*Volume = 6000 cubic units.*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\Inheritance>*

### There types of accessifier:

* Public
* Private
* Protected

### Inherited class(child class)

#### Public accessifier:

|  |  |  |  |
| --- | --- | --- | --- |
| Base Class | Private | Protected | Public |
| Child Class | No | Yes | Yes |

#### Protected accessifier:

|  |  |  |  |
| --- | --- | --- | --- |
| Base Class | Private | Protected | Public |
| Child Class | No | Yes | Yes |

#### Private accessifier:

|  |  |  |  |
| --- | --- | --- | --- |
| Base Class | Private | Protected | Public |
| Child Class | No | Yes | Yes |

## **Types of Inheritance:**

### Single Inheritance:

### C++ program to add the numbers to demonstrate single inheritance.

### Multiple Inheritance

### C++ program to store the marks of both subjects and sports of a Student and find the total marks of the student using multiple inheritance

// *C++ program to store the marks of both subjects and sports of a Student and find the total marks of the student using multiple inheritance.*

#*include* <iostream>

using namespace std;

class subject

{

public:

    int sub1, sub2, sub3;

// *protected:*

    void *getSubject*()

    {

        cout *<<* "Enter the marks of subject1 : ";

        cin *>>* sub1;

        cout *<<* "Enter the marks of subject2 : ";

        cin *>>* sub2;

    }

};

class sports

{

public:

    int sport;

// *protected:*

    void *getSport*()

    {

        cout *<<* "Enter the marks of sports : ";

        cin *>>* sport;

    }

};

class totalMarks : public subject, public sports

{

public:

    int total;

    void *display*()

    {

        total = sub1 + sub2 + sport;

        cout *<<* "Total marks:" *<<* total;

    }

};

int *main*()

{

// *int total;*

// *subject s;*

// *sports sp;*

// *s.getSubject();*

// *sp.getSport();*

// *total = s.sub1 + s.sub2 + sp.sport;*

// *cout << "Total marks:" << total;*

// *return 0;*

    totalMarks t;

    t.*getSubject*();

    t.*getSport*();

    t.*display*();

*return* 0;

}

Output:

*Enter the marks of subject1 : 99*

*Enter the marks of subject2 : 98*

*Enter the marks of sports : 100*

*Total marks:297*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\Inheritance>typesInheritance*

### Multilevel Inheritance

### C++ program to take the input in first class and find the area of a rectangle in second class and then volume in third class and display it.

/\*

*C++ program to take the input in first class and find the area of a rectangle in second class and then volume in third class and display it.*

*Using multilevel inheritance.*

\*/

#*include* <iostream>

using namespace std;

class value

{

public:

    double l, b;

    void *get*()

    {

        cout *<<* "Enter the length of the rectangle:";

        cin *>>* l;

        cout *<<* "Enter the breadth of the rectangle:";

        cin *>>* b;

    }

};

class \_area : public value

{

public:

    double area;

    void *findArea*()

    {

        area = l \* b;

    }

};

class \_volume : public \_area

{

public:

    double volume, h;

    void *findVolume*()

    {

        cout *<<* "Enter the height of the cuboid : ";

        cin *>>* h;

        volume = area \* h;

    }

};

class display : public \_volume

{

public:

    void *show*()

    {

        cout *<<* "Area of the rectangle is : " *<<* area *<<* *endl*;

        cout *<<* "Volume of the cuboid is : " *<<* volume *<<* *endl*;

    }

};

int *main*()

{

    display d;

    d.*get*();

    d.*findArea*();

    d.*findVolume*();

    d.*show*();

*return* 0;

}

Output:

*Enter the length of the rectangle:10*

*Enter the breadth of the rectangle:5*

*Enter the height of the cuboid : 6*

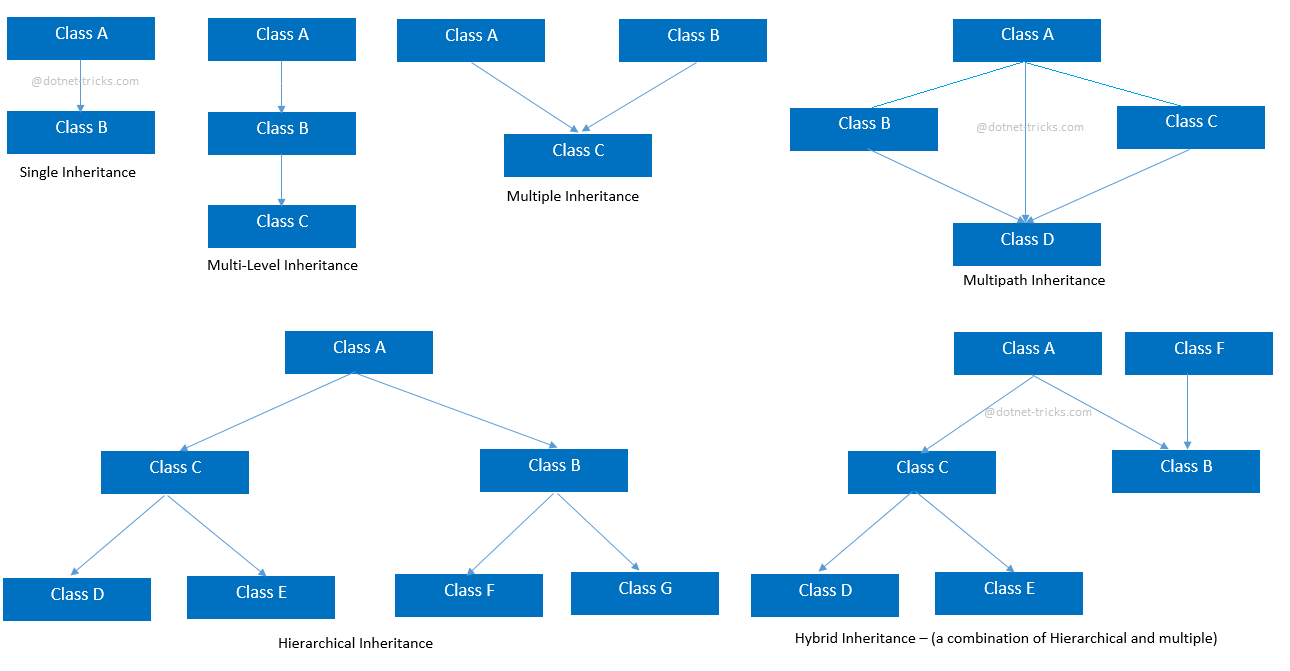
*Area of the rectangle is : 50*

*Volume of the cuboid is : 300*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\Inheritance>typesInheritance*

### Hierarchical Inheritance

### Hybrid Inheritance



## Function Overriding:

* Function Overloading is a feature that allows to have a function in child class which is already present in parent class.
* It is like creating a new version of an old function, in child class.

### C++ program to demonstrate the function overriding

// *C++ program to demonstrate the concept of function overriding*

#*include* <iostream>

using namespace std;

class Base

{

public:

    void *display*()

    {

        cout *<<* "This is a parent class." *<<* *endl*;

    }

};

class Derived : public Base

{

public:

    void *display*()

    {

        cout *<<* "This is a child class." *<<* *endl*;

    }

};

int *main*()

{

    Base b;

    Derived d;

    b.*display*();

//*? Using Scope Resolution.*

// *d.Base::display();*

//*? Using Pointer.*

// *Base \*ptr = &d;*

// *ptr->display();*

// *you may use this methods instead of Base b & b.display();*

    d.*display*();

*return* 0;

}

**Output :**

*This is a parent class.*

*This is a child class.*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\functionOverriding>*

## Ambiguity in Multiple Inheritance :

The most obvious problem with multiple inheritance occurs during overriding.

##### Why Ambiguity is required?

#*include* <iostream>

using namespace std;

class A

{

public:

    void *display*()

    {

        cout *<<* "This is a A class." *<<* *endl*;

    }

};

class B

{

public:

    void *display*()

    {

        cout *<<* "This is a B class." *<<* *endl*;

    }

};

class C : public A, public B

{

    cout << "This is a C class." << endl;

};

int *main*()

{

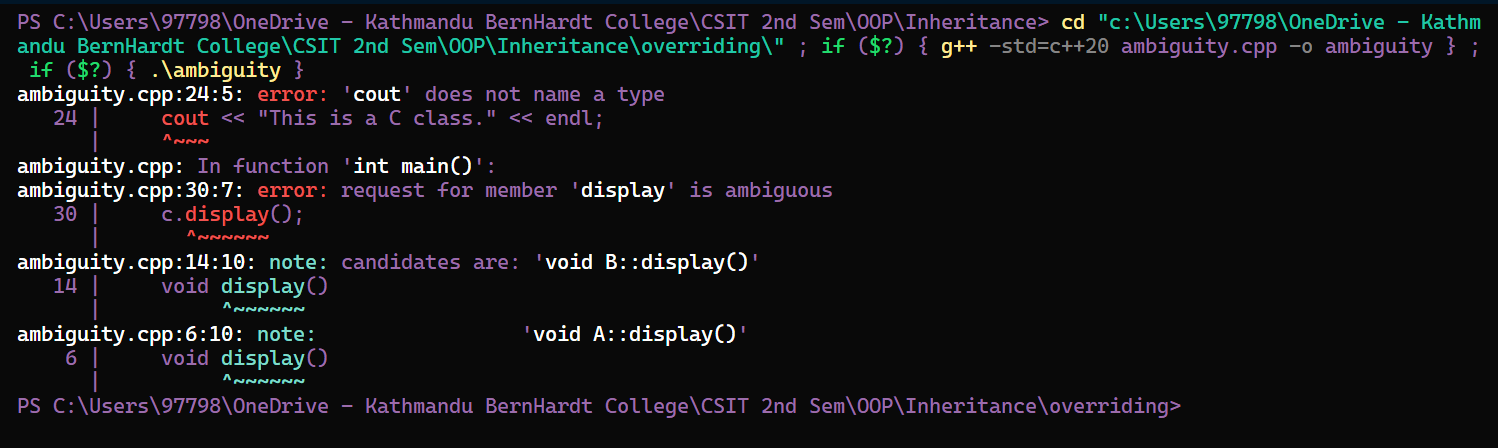
    C c;

    c.*display*();

*return* 0;

}

**Error generated by this program :**

****

### C++ program to demonstrate

## Abstract base class :

Class that contain at least one pure virtual function is known as abstract base class.

### C++ program to demonstrate abstract base class

class rect

{

protected:

    int length;

    int breadth;

public:

    void *setvalues*(int a, int b)

    {

        length = a;

        breadth = b;

    }

*virtual* int *area*() = 0;// *pure virtual function.*

};

### C++ program to calculate the area using abstract base class.

// *C++ program to calculate the area using abstract base class.*

#*include* <iostream>

using namespace std;

class poly

{

protected:

    int length;

    int breadth;

public:

    void *setvalues*(int a, int b)

    {

        length = a;

        breadth = b;

    }

*virtual* int *area*() = 0;// *pure virtual function.*

};

class Rect : public poly

{

public:

    int *area* ()

    {

*return* length \* breadth;

    }

};

int *main*()

{

// *Rect r;*

// *r.setvalues(3, 2);*

// *cout << "Area = " << r.area() << " sq. units" << endl;*

// *return 0;*

    Rect r1;

    poly \*p1 = &r1;

    p1->*setvalues*(3, 2);

    cout *<<* "Area = " *<<* p1->*area*() *<<* " sq. units" *<<* *endl*;

*return* 0;

}

Output:

*Area = 6 sq. units*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\Abstraction>*

### C++ program to calculate the area of a triangle using abstract base class.

// *C++ PROGRAM TO CALCULATE THE AREA OF A TRIANGLE USING ABSTRACT BASE CLASS.*

#*include* <iostream>

using namespace std;

class poly

{

protected:

    int base;

    int height;

public:

    void *setValues*(int a, int b)

    {

        base = a;

        height = b;

    }

*virtual* int *area*() = 0;// *pure virtual function.*

};

class Triangle : public poly

{

public:

    int *area*()

    {

*return* 0.5 \* base \* height;

    }

};

int *main*()

{

// *Triangle t;*

// *t.setValues(3, 2);*

// *cout << "Area = " << t.area() << " sq. units" << endl;*

// *return 0;*

    Triangle t1;

    poly \*p1 = &t1;

    p1->*setValues*(3, 2);

    cout *<<* "Area = " *<<* p1->*area*() *<<* " sq. units" *<<* *endl*;

*return* 0;

}

Output:

*Area = 3 sq. units*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\Abstraction>*

## Aggression ( class within class)

Aggression is a process in which one class defines another class as any entity reference. It is another way to reuse the class. It is a form of association that represent has a relationship.

## constructor and destructor in derived class :

In heritance, when an object of derived class is created than constructor of derived class get executed and then it calls the constructor of base class.

**Syntax:**

//C++ program calling the parent class constructor using child class constructor.  
class A  
{   
 public:   
 A()  
 {  
 **//body of constructor.** }  
};  
class B : public A  
{   
 public :   
 B(A) : A()  
 {  
 **//body of constructor.** }  
};

// *C++ program to demonstrate the constructor in parent & derived class*

#*include* <iostream>

using namespace std;

class Parent

{

protected:

    int a;

public:

*Parent*(int i)

    {

        a = i;

    }

};

class Child : public Parent

{

    int b;

public:

// *Child(int j) : Parent()*

// *{*

// *b = j;*

// *}*

*Child*(int x, int y) : *Parent*(x)

    {

        b = y;

    }

    void *display*()

    {

        cout *<<* "a = " *<<* a *<<* *endl*

*<<* "b = " *<<* b;

    }

};

int *main*()

{

// *Child c1(10);*

    Child *c1*(2, 3);

    c1.*display*();

*return* 0;

}

**Output:**

a = 2

b = 3

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\*

# Unit : 6

# Virtual funtion, Polymorphism and Miscellaneaous

## Virtual function:

* A virtual function in C++ is a base class member function that you can redefine in a derived class to achieve polymorphism.
* To define virtual function, we use the virtual keyword.
* Once declare the function in the base class, we use a pointer or reference to call the virtual class and execute its virtual version in derived class.

### Rules of virtual function

* The functin cannot be static.
* You derive them using the the virtual keyword.
* They can be a friend function of another class.
* The prototype of these functions should be the same for both the base class and derived class.
* Virtual functions are accessible using object pointers.
* You can create a virtual destructor but not a constructor.
* Redefining the virtual function in the derived class is optional, but it needs to be defined in the base class.

### Behaviour of virtual function :

* Compile time behaviour.
* Run time behaviour.

|  |  |  |
| --- | --- | --- |
| **Base** | **Compile Time** | **Run Time** |
| Alternative name | Early binding | Late binding |
| It is achieve | The types of pointer | Depending on the location where pointer is pointing. |

### Binding

The process of converting identifiers into adresses.

#### Types of binding :

* Early binding
* Late binding

### Early binding

* It is a compile time polymorphism.
* It directly associates an address to the function call.
* Function overloading is an example of early binding.

### Late binding

* It is a run time polymorphism.
* This is achieved by using virtual function.

## C++ program to demonstrate the behaviour of virtual functions.

// *C++ program to demonstrate the behaviour of the virtual functions.*

#*include* <iostream>

using namespace std;

class Base

{

public:

*virtual* void *show*()

    {

// *cout << " In Base \ n";*

        cout *<<* "This is a base class." *<<* *endl*;

    }

    void *print*()

    {

        cout *<<* "Print base class " *<<* *endl*;

    }

};

class Derived : public Base

{

public:

    void *show*()

    {

        cout *<<* "This is a derived class. " *<<* *endl*;

    }

    void *print*()

    {

        cout *<<* "Print derived class " *<<* *endl*;

    }

};

int *main*()

{

    Base \*bptr;

    Derived d;

    bptr = &d;

// *virtual function, binded at runtime*

    bptr->*show*();

*return* 0;

}

Output:

*This is a derived class.*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\Polymorphism>*

### Pure virtual function

A function that doesn’t perform any task. (do-nothing). It is declared in abstract base class.

**Syntax :**

virtual void show () = **0** ;

## C++ program using abstract base class to find the area of a circle and a rectangle.

// *C++ program using abstract base class to find the area of a circle and a rectangle.*

#*include* <iostream>

#*define* *pi* 3.14

using namespace std;

class Shape

{

public:

*virtual* void *area*() = 0;

};

class Circle : public Shape

{

public:

    void *area*()

    {

        float r, a;

        cout *<<* "Enter the radius of the circle: ";

        cin *>>* r;

        a = *pi* \* r \* r;

        cout *<<* "Area of the circle is: " *<<* a *<<* " sq. units." *<<* *endl*;

    }

};

class Rectangle : public Shape

{

public:

    void *area*()

    {

        float l, b, a;

        cout *<<* "Enter the length and breadth of the rectangle: ";

        cin *>>* l *>>* b;

        a = l \* b;

        cout *<<* "Area of the rectangle is: " *<<* a *<<* " sq units." *<<* *endl*;

    }

};

int *main*()

{

    Shape \*sptr;

    Circle c;

    Rectangle r;

    sptr = &c;

    sptr->*area*();

    sptr = &r;

    sptr->*area*();

// *c.area();*

// *r.area();*

*return* 0;

}

Output:

*Enter the radius of the circle: 5*

*Area of the circle is: 78.5 sq. units.*

*Enter the length and breadth of the rectangle: 10 5*

*Area of the rectangle is: 50 sq units.*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\Polymorphism>*

## Friend function and friend class:

-Friend functions that allow us to access member functions from outside the class

-It has the authority to access all protected and private members of the class

**Syntax:**

class classname{  
 friend returnType Function\_name(arguments);   
};

**Another method of declaration:**

class classB;  
class classA{  
 friend class classB;  
}  
class classB{

};

## C++ program to demonstrate friend function

// *C++ PROGRAM TO DEMONSTRATE THE WORKING OF FRIEND FUNCTION.*

#*include* <iostream>

using namespace std;

class Distance

{

private:

    int m;

*friend* int *add*(Distance);

public:

*Distance*() : *m*(0)

    {

    }

};

int *add*(Distance d)

{

    d.m += 5;

*return* d.m;

}

int *main*()

{

    Distance D;

    cout *<<* "Distance = " *<<* *add*(D) *<<* "m." *<<* *endl*;

*return* 0;

}

Output:

*Distance = 5m.*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\Polymorphism>*

## C++ program to add members of two members of two different classes using friend function

// *C++ program to add members of two different classes using friend function.*

#*include* <iostream>

using namespace std;

class class\_B;

class class\_A

{

public:

*class\_A*() : *num\_A*(6)

    {

    }

private:

    int num\_A;

*friend* int *add*(class\_A, class\_B);

};

class class\_B

{

public:

*class\_B*() : *num\_B*(7)

    {

    }

private:

    int num\_B;

*friend* int *add*(class\_A, class\_B);

};

int *add*(class\_A A, class\_B B)

{

*return* A.num\_A + B.num\_B;

}

int *main*()

{

    class\_A A;

    class\_B B;

    cout *<<* "Sum = " *<<* *add*(A, B) *<<* *endl*;

*return* 0;

}

Output:

*Sum = 13*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\Polymorphism>*

## This Pointer :

* The “this” pointer holds the address of the current object, in simple words you can say that this pointer points to the current object of the class.

## C++ program to demonstrate the working of “this” pointer

// *C++ program to demonstrate the working of this pointer.*

#*include* <iostream>

using namespace std;

class A

{

private:

    int num;

    char ch;

public:

    void *setValues*(int num, char ch)

    {

        this->num = num;

        this->ch = ch;

    }

    void *display*()

    {

        cout *<<* "num = " *<<* num *<<* *endl*;

        cout *<<* "ch = " *<<* ch *<<* *endl*;

    }

};

int *main*()

{

    A obj;

    obj.*setValues*(21, 'p');

    obj.*display*();

*return* 0;

}

Output:

*num = 21*

*ch = p*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\Polymorphism>*

### Difference between abstract class and concrete class.

* The main difference between abstract class and concrete class is that it is not possible to create objects using an abstract class while using a concrete class, it is possible to create objects.

### Polymorphism:

When same entity (function or object) behaves differently in different scenario, it is known as polymorphism.

‘+’ operator perform two specific functions at two different scenaries.

1. When ‘+’ operator is used in numbers, it performs addition.

int a = 5;  
int b = 6;  
int s = a + b;

1. ‘+’ operator is used in string, it performs concatenation.

string s1 = “Purna”;  
string s2 = “Shrestha”  
string name = s1 + s2;

# Unit = 7

# function templates & Exception handling

## Exceptional Handling :

Exception handling is the process of handling errors and exceptions in such way that they do not hinder normal execution of the system.

Example :   
User divides a number by zero, this c0mpile successfully but a run time errors occur. To avoid this we will introduce Exception handling techniques.

**Error handling is done using three keywords.**

* Try
* Catch
* Throw

**Syntax :**

try {  
 //protected code  
 throw parameter.  
}  
catch (Exception\_Name ex) {  
 //code to handle exception.  
}

## C++ program to demonstrate error.

#*include* <iostream>

using namespace std;

int *main*()

{

    int a = 10, b = 0, c;

    c = a / b;

    cout *<<* c *<<* *endl*;

*return* 0;

}

Output:

Here, output will not be printed.

## C++ to demonstrate try, catch and throw.

#*include* <iostream>

using namespace std;

int *main*()

{

    int a = 10, b = 0, c;

*try*

    {

*if* (b == 0)

        {

*throw* "Division by zero is not possible.";

        }

        c = a / b;

    }

*catch* (*const* char \*ex)

    {

        cout *<<* ex;

    }

*return* 0;

}

Output :

*Division by zero is not possible.*

*PS C:\Users\Purna\KBC\CSIT 2nd Sem\OOP\Exception>*

Example :

## multiple catch blocks

* C++ program contains multiple catch block to handle different types of exception in different way.