**KGiSL**

***We make IT happen***

**BOOTCAMP – Tutorial**

*We build software to solve problems.   
People have problems.*

*Therefore, we build software for people.*

*Good software not only solves immediate problems, but it can be maintained and modified to address the inevitable changes that the customer will want.*

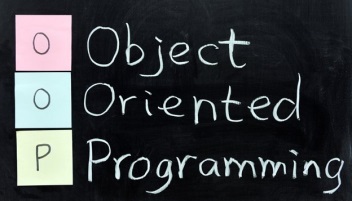
***OOP*** 

Why should I modify many lines to make a change???

Reusable???



Secure??

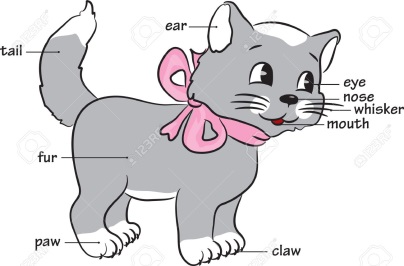


* Object-oriented programming is an approach to designing modular reusable software systems.

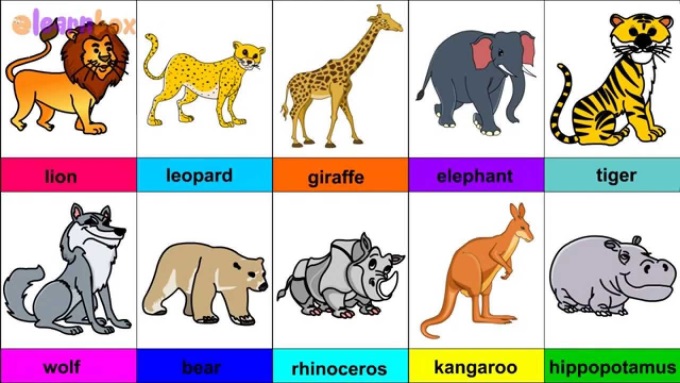
***Classes***



***Features/Attributes***

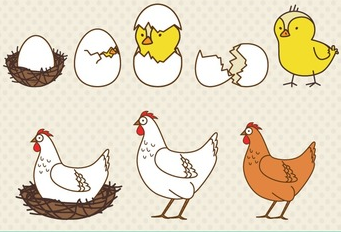


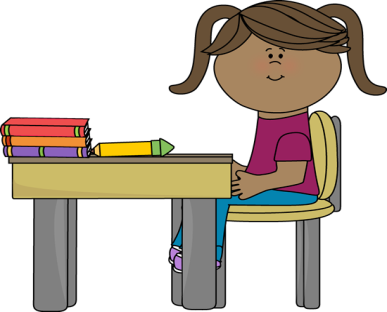
***Objects***





***Methods***

 ******



***Example:***

// Sample code for printing “Hello World”

#include<iostream.h>

class Display // class name

{ public: // access specifier – scope of access

void dispMsg() // method definition

{

cout<<”Hello World\n”; // print string

}

};

void main()

{

Display first,second,third; //objects

first.dispMsg(); //method invocation / Method binding

first.dispMsg();

first.dispMsg();

}

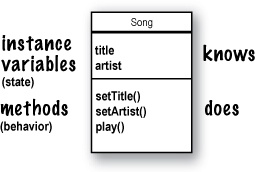
**Output:**

Hello World

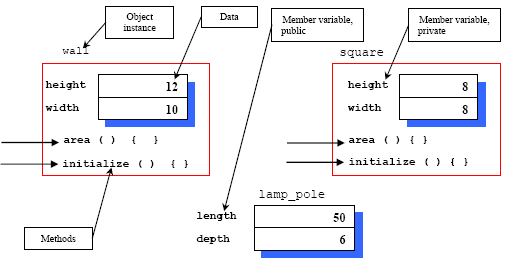
Hello World

Hello World

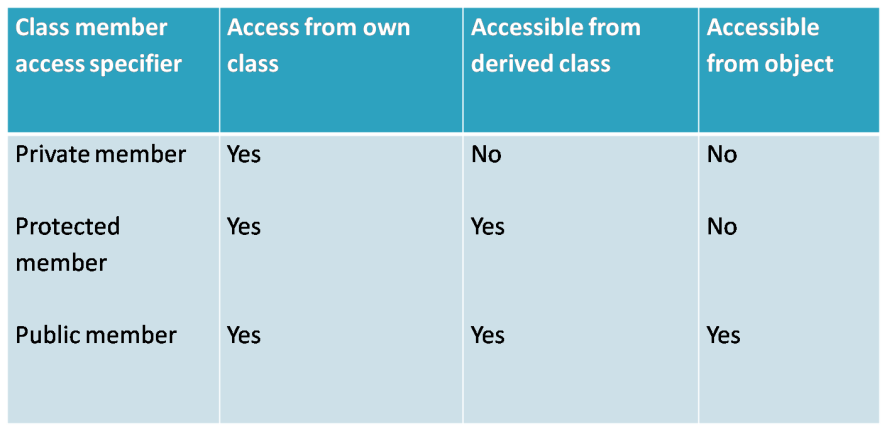
***Memory map & Access Specifiers***



***Class - Shape***

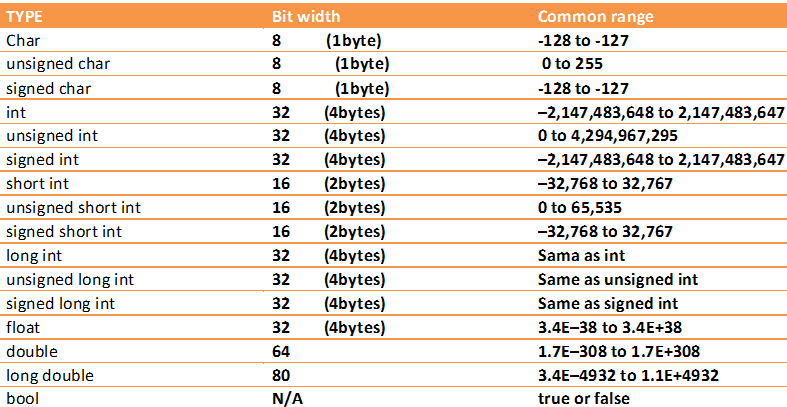
******

***Private, Public, Protected***



***(Protected will be revisited)***

***Datatypes***

***(one byte)***

***Example:***

//Program to enter and display students details

#include<iostream.h>

class Students

{

private: // default

int iRollNo;

char cName[15];

public:

void getValues();

void displayValues();

};

void Students::getValues() // scope resolution operator

{

cout<<”\n Enter your Roll No:”;

cin>>iRollNo;

cout<<”\n Enter your name:”;

cin>>cName;

}

void Students::displayValues()

{

cout<<”\n Roll No:”<< iRollNo;

cout<<”\n Name:”<< cName;

}

void main()

{

Students s[60];

int m;

for(m=1;m<=60;m++)

{

s[i].getValues();

s[i].displayValues();

}

}

**Output:**

Enter your Roll No: 1

Enter your name: Akila

Roll No: 1

Name: Akila

**.**

**.**

**.**

***Static attributes***

***Example:***

//Program to display the number of orders placed

#include<iostream.h>

class Shopping\_cart

{

int iItemNo;

char cItemName[15];

public:

static int iNo\_of\_Orders;

void orderItem();

{

cout<<”\n Enter ItemNo & ItemName:”;

cin>> iItemNo>> cItemName;

iNo\_of\_Orders++;

}

};

int Shopping\_cart:: iNo\_of\_Orders=0;

void main()

{

Shopping\_cart order1,order2,order3;

order1. orderItem();

order2. orderItem();

order3. orderItem();

cout<<”\n Total Orders: ”<< Shopping\_cart:: iNo\_of\_Orders;

}

**Output:**

Total Orders: 3

***Method Invocations with parameters***

***Example:***

// Program to add 2 numbers

#include<iostream.h>

class Addition

{

int iNo1,iNo2,iSum;

public:

int addNos(int iVal1,int iVal2)

{

iNo1=iVal1;

iNo2=iVal2;

return(iNo1+iNo2);

}

};

void main()

{

Addition set1,set2;

int iResult;

iResult=set1.addNos(12,78); //methods with parameters

cout<<”First set:”<< iResult;

iResult=set2.addNos(-12,8); //methods with parameters

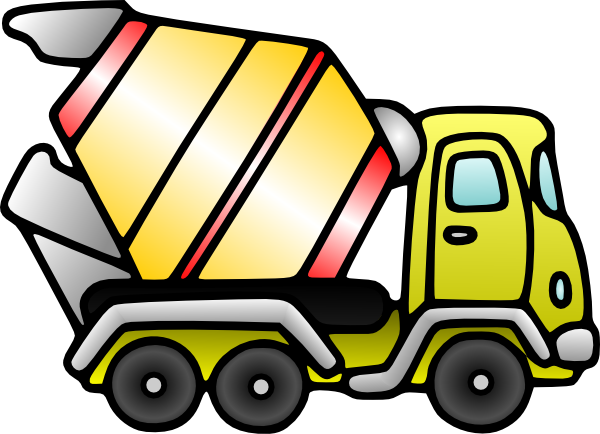
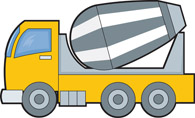
cout<<”Second set:”<< iResult;

}

***Practice with Coding Standards***

1. Write a C++ Program to display names, roll no’s, and grades of 3 students who have appeared in the examination. Declare the class of name, roll no’s and grade. Create an array of class objects. Read and display the contents of the array.
2. Write a C++ program to declare struct. Initialize and display contents of member variables.
3. Given that an EMPLOYEE class contains following members: data members: Employee number, Employee name, Basic, DA, IT, Net Salary and print data members. Write a C++ program to read the data of N employee and compute Net salary of each employee (DA=52% of Basic and Income Tax (IT) =30% of the gross salary) .
4. Write a C++ program to find the value of a number raised to its power that demonstrates a function using call by value.
5. Write a c++ program and to implement the concept of Call by Address.
6. Implement friend functions in C++.
7. Implement friend classes in C++.
8. Write a C++ program to show the significance of boolean operator.
9. Write a C++ program to find the largest of three numbers using inline function.
10. Write a C++ program to sort an array of integer in ascending order using a function called exchange( ) which accepts two integer arguments by reference.

***Constructors***



***Example:***

//Program using default constructor

#include<iostream.h>

#include<conio.h>

class stu

{

private:

char cName[20],cAdd[20];

int iRoll,iZip;

public:

stu ( ); //Constructor

~stu( ); //Destructor

void readDet( );

void dispDet( );

};

stu :: stu( )

{

cout<<”This is Student Details”<<endl;

}

void stu :: readDet( )

{

cout<<”Enter the student Name”;

cin>>cName;

cout<<”Enter the student roll no “;

cin>>iRoll;

cout<<”Enter the student address”;

cin>>cAdd;

cout<<”Enter the Zipcode”;

cin>>iZip;

}

void stu :: dispDet( )

{

cout<<”Student Name :”<< cName <<endl;

cout<<”Roll no is :”<<iRoll<<endl;

cout<<”Address is :”<<cAdd<<endl;

cout<<”Zipcode is :”<<iZip;

}

stu : : ~stu( )

{

cout<<”Student Detail is Closed”;

}

void main( )

{

stu s;

clrscr( );

s.read ( );

s.disp ( );

getch( );

}

**Output:**

Enter the student Name  
James  
Enter the student roll no  
01  
Enter the student address  
Newyork  
Enter the Zipcode  
919108

Student Name : James  
Roll no is : 01  
Address is : Newyork  
Zipcode is :919108

***Example:***

//Program with parameterized constructor

#include <iostream.h>   
#include<conio.h>   
class Myclass   
{   
     int iNo1,iNo2;   
    public:   
            myclass(int i, int j) // parameterized constructor

{   
              iNo1=i;   
             iNo2=j;   
        }   
              void showRes()   
      {   
             cout<< iNo1 << " " << iNo2;   
         }   
};

void main()   
{

           Myclass obNew(3, 5);   
           obNew.showRes();   
           getch();

}

**Output:**   
3 5

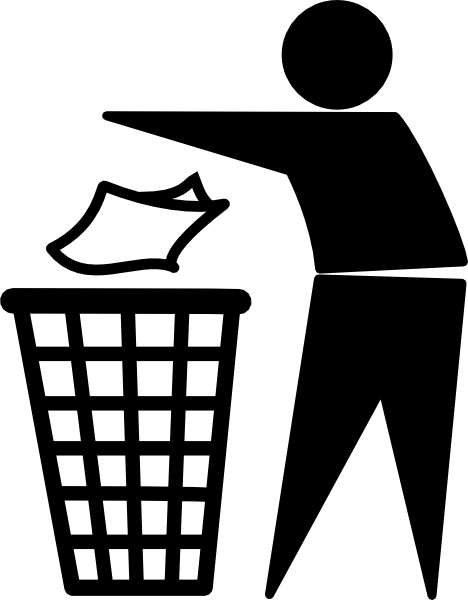
***Object creation with new operator and constructor***

**Point originOne = new Point(23, 94);**

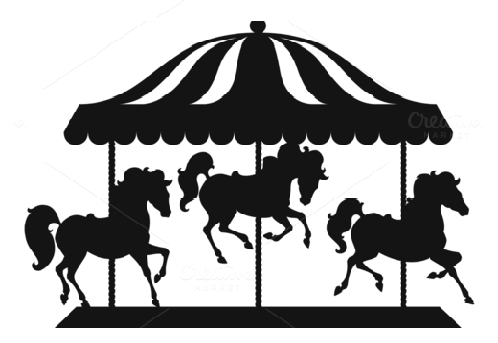
**Rectangle rectOne = new Rectangle(originOne, 100, 200);**

**Rectangle rectTwo = new Rectangle(50, 100);**

***Destructors***



***Overloading***

******

***Example:***

//Function Overloading

#include <iostream>

class PrintData

{

public:

void printDet(int i)

{

cout << "Printing int: " << i << endl;

}

void printDet(double f)

{

cout << "Printing float: " << f << endl;

}

void printDet(char\* c)

{

cout << "Printing character: " << c << endl;

}

};

void main()

{ PrintData pd; // Call print to print integer

pd.printDet(5); // Call print to print float

pd.printDet(500.263); // Call print to print character

pd.printDet("Hello C++");

}

**Output:**

Printing int: 5

Printing float: 500.263

Printing character: Hello C++

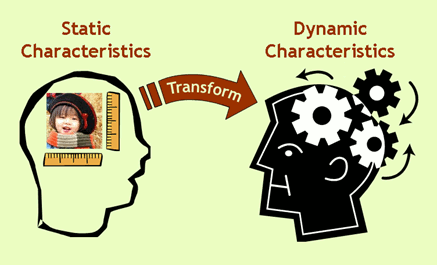
***Practice with Coding Standards***

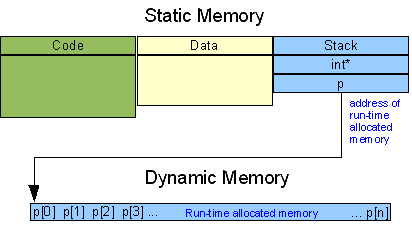
1. Create a class for counting the number of objects created and destroyed within various block using constructor and destructors.
2. Write a C++ program to count the number of persons inside a bank, by increasing count whenever a person enters a bank, using an increment (++) operator overloading function, and decrease the count whenever a person leaves the bank using a decrement (--) operator overloading function inside a class.
3. Write a C++ program to create two objects of a class called company and add their data members using an operator overloaded function for ‘+’ operator and ‘-‘operator.
4. Write a C++ program to implement flight class with data member as flight no, source, destination and fare.  Write a constructor and a member function to display the flight information.
5. Write a C++ program to implement a string object.  Include member functions to compare two strings and to concatenate two strings.
6. Write a C++ program to implement time class that has separate data members for hours, minutes and seconds.  Overload + Operator to add two times (object) and ++ operator to increment the time by one second.
7. Write a C++ program to implement a student class having roll no, name, rank, and addresses as data members.  Overload assignment operator =. Write a C++ program to implement user defined string class.  Overload the constructor and a member function to concatenate two strings.
8. Write a C++ program to find the number of characters, word and lines in the given text as input using constructors.
9. Write a C++ program to implement a telephone bill class with Name, Address, Tel. No., No. of calls as data members.  Compute the amount to be paid if the charges per call is Rs. 2/-.
10. Write a C ++ program to implement a date class with member functions as next, previous which return next date and

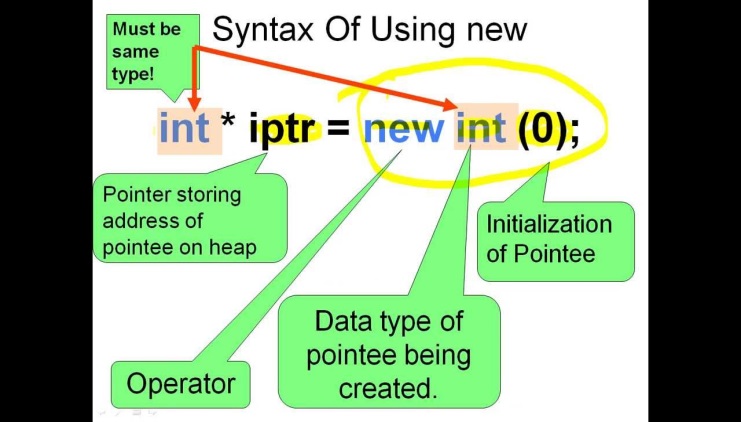
previous date objects.

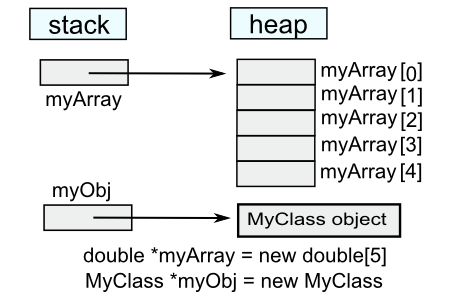
***Static & Dynamic Memory Allocation***









***Example:***

//C++ Program to store GPA of n number of students and display it where n is the number of students entered by user.

#include <iostream>

int main()

{

int n;

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

cin >> n;

float\* ptr;

ptr = new float[n]; // memory allocation for n number of floats

cout << "Enter GPA of students." <<endl;

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

{

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

cin >> \*(ptr + i);

}

cout << "\nDisplaying GPA of students." << endl;

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

{

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

}

delete [] ptr; // ptr memory is released

return 0;

}

**Output:**

Enter total number of students: 4

Enter GPA of students.

Student1: 3.6

Student2: 3.1

Student3: 3.9

Student4: 2.9

Displaying GPA of students.

Student1 :3.6

Student2 :3.1

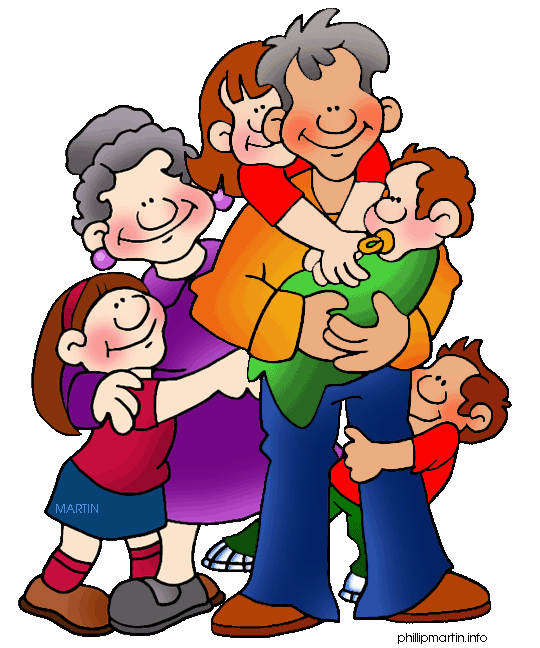
Student3 :3.9

Student4 :2.9

***Practice with Coding Standards***

1. Linked list – create, insert, delete, update, modify

***Inheritance***

***Protected (revisited)***

***Example:***

//Single Inheritance

class Shape

{

protected:

float fWidth, fHeight;

public:

void setData (float a, float b)

{

fWidth = a;

fHeight = b;

}

};

class Rectangle: public Shape

{

public:

float setArea ()

{

return (fWidth \* fHeight);

}

};

class Triangle: public Shape

{

public:

float setArea ()

{

return (fWidth \* fHeight / 2);

}

};

void main ()

{

Rectangle rect;

Triangle tri;

rect.setData (5,3);

tri.setData (2,5);

cout << rect.setArea() << endl;

cout << tri.setArea() << endl;

}

**Output :**

15  
5

* The object of the class Rectangle contains:   
  fWidth, fHeight inherited from Shape becomes the protected member of Rectangle.   
  setData() inherited from Shape becomes the public member of Rectangle  
  setArea() is Rectangle’s own public member
* The object of the class Triangle contains:   
  fWidth, fHeight inherited from Shape becomes the protected member of Triangle.  
  setData() inherited from Shape becomes the public member of Triangle  
  setArea() is Triangle’s own public member
* setData() and setArea() are public members of derived class and can be accessed from outside class i.e. from main()

|  |  |
| --- | --- |
| **Member Access Specifier** | **How Members of the Base Class Appear in the Derived Class** |
| Private | Private members of the base class are inaccessible to the derived class. |
| Protected members of the base class become private members of the derived class. |
| Public members of the base class become private members of the derived class. |
| Protected | Private members of the base class are inaccessible to the derived class. |
| Protected members of the base class become protected members of the derived class. |
| Public members of the base class become protected members of the derived class. |
| Public | Private members of the base class are inaccessible to the derived class. |
| Protected members of the base class become protected members of the derived class. |
| Public members of the base class become public members of the derived class. |

***Overriding***

#include <iostream>

class Area

{

public:

float areaCalc(float l,float b)

{

return l\*b;

}

};

class Perimeter

{

public:

float periCalc(float l,float b)

{

return 2\*(l+b);

}

};

/\* Rectangle class is derived from classes Area and Perimeter. \*/

class Rectangle : private Area, private Perimeter

{

private:

float length, breadth;

public:

Rectangle() : length(0.0), breadth(0.0) { }

void getData( )

{

cout<<"Enter length: ";

cin>>length;

cout<<"Enter breadth: ";

cin>>breadth;

}

float areaCalc() //overriding

{

/\* Calls areaCalc() of class Area and returns it. \*/

return Area::area\_calc(length,breadth);

}

float peri\_calc() //overriding

{

/\* Calls periCalc() function of class Perimeter and returns it. \*/

return Perimeter::peri\_calc(length,breadth);

}

};

int main()

{

Rectangle r;

r.getData();

cout<<"Area = "<<r.areaCalc();

cout<<"\nPerimeter = "<<r.periCalc();

return 0;

}

**Output**

Enter length: 5.1

Enter breadth: 2.3

Area = 11.73

Perimeter = 14.8

***Method binding (revisited) – static & dynamic***

#include <iostream>

using namespace std;

class B

{

public:

void display()

{ cout<<"Content of base class.\n"; }

};

class D : public B

{

public:

void display()

{ cout<<"Content of derived class.\n"; }

};

int main()

{

B \*b;

D d;

b->display();

b = &d; /\* Address of object d in pointer variable \*/

b->display();

return 0;

}

**Note:**An object(either normal or pointer) of derived class is type compatible with pointer to base class. So, b = &d; is allowed in above program.

**Output**

Content of base class.

Content of base class.

In above program, even if the object of derived class *d* is put in pointer to base class, display( ) of the base class is executed( member function of the class that matches the type of pointer ).

***Solution***

If you want to execute the member function of derived class then, you can declare display( ) in the base class virtual which makes that function existing in appearance only but, you can't call that function. In order to make a function virtual, you have to add keyword **virtual** in front of a function.

/\* Example to demonstrate the working of virtual function in C++ programming. \*/

#include <iostream>

using namespace std;

class B

{

public:

virtual void display() /\* Virtual function \*/

{ cout<<"Content of base class.\n"; }

};

class D1 : public B

{

public:

void display()

{ cout<<"Content of first derived class.\n"; }

};

class D2 : public B

{

public:

void display()

{ cout<<"Content of second derived class.\n"; }

};

int main()

{

B \*b;

D1 d1;

D2 d2;

/\* b->display(); // You cannot use this code here because the function of base class is virtual. \*/

b = &d1;

b->display(); /\* calls display() of class derived D1 \*/

b = &d2;

b->display(); /\* calls display() of class derived D2 \*/

return 0;

}

**Output**

Content of first derived class.

Content of second derived class.

After the function of base class is made virtual, code b->display( ) will call the display( ) of the derived class depending upon the content of pointer.

In this program, display( ) function of two different classes are called with same code which is one of the example of polymorphism in C++ programming using virtual functions.

***Practice with Coding Standards***

1. Create a bank application using inheritance
2. Create an online shopping cart application using inheritance

***this pointer***

Every object in C++ has access to its own address through an important pointer called **this** pointer. The **this** pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object.

Friend functions do not have a **this** pointer, because friends are not members of a class. Only member functions have a **this** pointer.

/\* Concept of this pointer\*/

#include <iostream>

class Box

{

public:

// Constructor definition

Box(double l=2.0, double b=2.0, double h=2.0)

{

cout <<"Constructor called." << endl;

length = l;

breadth = b;

height = h;

}

double Volume()

{

return length \* breadth \* height;

}

int compare(Box box)

{

return this->Volume() > box.Volume();

}

private:

double length; // Length of a box

double breadth; // Breadth of a box

double height; // Height of a box

};

int main(void)

{

Box Box1(3.3, 1.2, 1.5); // Declare box1

Box Box2(8.5, 6.0, 2.0); // Declare box2

if(Box1.compare(Box2))

{

cout << "Box2 is smaller than Box1" <<endl;

}

else

{

cout << "Box2 is equal to or larger than Box1" <<endl;

}

return 0;

}

When the above code is compiled and executed, it produces the following result:

Constructor called.

Constructor called.

Box2 is equal to or larger than Box1

***Exception handling***

Exceptions provide a way to react to exceptional circumstances (like runtime errors) in programs by transferring control to special functions called *handlers*.  
  
To catch exceptions, a portion of code is placed under exception inspection. This is done by enclosing that portion of code in a *try-block*. When an exceptional circumstance arises within that block, an exception is thrown that transfers the control to the exception handler. If no exception is thrown, the code continues normally and all handlers are ignored.  
  
An exception is thrown by using the throw keyword from inside the try block. Exception handlers are declared with the keyword catch, which must be placed immediately after the try block:

***Example:***

|  |  |  |
| --- | --- | --- |
| // Exceptions  #include <iostream>  using namespace std;  int main ()  {  try  {  throw 20;  }  catch (int e)  {  cout << "An exception occurred. Exception Nr. " << e << '\n';  }  return 0;  } |  |  |

The code under exception handling is enclosed in a try block. In this example this code simply throws an exception:

|  |  |  |
| --- | --- | --- |
|  | throw 20; |  |

A throw expression accepts one parameter (in this case the integer value 20), which is passed as an argument to the exception handler.  
  
The exception handler is declared with the catch keyword immediately after the closing brace of the try block. The syntax for catch is similar to a regular function with one parameter. The type of this parameter is very important, since the type of the argument passed by the throw expression is checked against it, and only in the case they match, the exception is caught by that handler.  
  
Multiple handlers (i.e., catch expressions) can be chained; each one with a different parameter type. Only the handler whose argument type matches the type of the exception specified in the throw statement is executed.

***Practice with Coding Standards***

1. Implement the above 2 applications using exception handling mechanism.
2. An insurance company follows the following rules to calculate premium:

a) If person’s health is excellent and the person is between 25 and 35 years of age and lives in a city and is male then the premium is Rs. 4 per thousand and his policy amount cannot exceed Rs. 2 lakhs.

b) If a person satisfies all the above conditions except that sex is female then the premium is Rs. 3 per thousand and her policy amount cannot exceed Rs. 1 lakh.

c) If person’s health is poor and the person is between 25 and 35 years of age and lives in a village and is male then the premium is Rs. 6 per thousand and his policy amount cannot exceed Rs. 10,000.

d) In all other cases the person is not insured.

Write a program to output whether the person should be insured or not, his/her premium rate and maximum amount for which he/she can be insured.