C++ OOPs Concepts

The major purpose of C++ programming is to introduce the concept of object orientation to the C programming language.

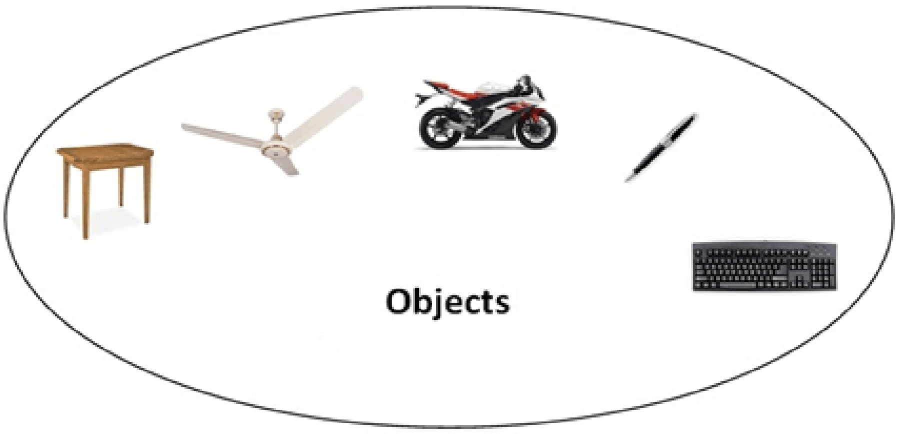
Object Oriented Programming is a paradigm that provides many concepts such as **inheritance, data binding, polymorphism etc.**

The programming paradigm where everything is represented as an object is known as truly object-oriented programming language. **Smalltalk** is considered as the first truly object-oriented programming language.

OOPs (Object Oriented Programming System)

Object means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation



Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class

Collection of objects is called class. It is a logical entity.

A Class in C++ is the foundational element that leads to Object-Oriented programming. A class instance must be created in order to access and use the user-defined data type's data members and member functions. An object's class acts as its blueprint. Take the class of cars as an example. Even if different names and brands may be used for different cars, all of them will have some characteristics in common, such as four wheels, a speed limit, a range of miles, etc. In this case, the class of car is represented by the wheels, the speed limitations, and the mileage.

Inheritance

When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

1. Sub class - Subclass or Derived Class refers to a class that receives properties from another class.
2. Super class - The term "Base Class" or "Super Class" refers to the class from which a subclass inherits its properties.
3. Reusability - As a result, when we wish to create a new class, but an existing class already contains some of the code we need, we can generate our new class from the old class thanks to inheritance. This allows us to utilize the fields and methods of the pre-existing class.

Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

Different situations may cause an operation to behave differently. The type of data utilized in the operation determines the behavior.

Abstraction

Hiding internal details and showing functionality is known as abstraction. Data abstraction is the process of exposing to the outside world only the information that is absolutely necessary while concealing implementation or background information.For example: phone call, we don't know the internal processing.

In C++, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

Encapsulation is typically understood as the grouping of related pieces of information and data into a single entity. Encapsulation is the process of tying together data and the functions that work with it in object-oriented programming. Take a look at a practical illustration of encapsulation: at a company, there are various divisions, including the sales division, the finance division, and the accounts division. All financial transactions are handled by the finance sector, which also maintains records of all financial data. In a similar vein, the sales section is in charge of all tasks relating to sales and maintains a record of each sale. Now, a scenario could occur when, for some reason, a financial official requires all the information on sales for a specific month. Under the umbrella term "sales section," all of the employees who can influence the sales section's data are grouped together. Data abstraction or concealing is another side effect of encapsulation. In the same way that encapsulation hides the data. In the aforementioned example, any other area cannot access any of the data from any of the sections, such as sales, finance, or accounts.

Dynamic Binding - In dynamic binding, a decision is made at runtime regarding the code that will be run in response to a function call. For this, C++ supports virtual functions.

Advantage of OOPs over Procedure-oriented programming language

1. OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.
2. OOPs provide data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.
3. OOPs provide ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

Why do we need oops in C++?

There were various drawbacks to the early methods of programming, as well as poor performance. The approach couldn't effectively address real-world issues since, similar to procedural-oriented programming, you couldn't reuse the code within the program again, there was a difficulty with global data access, and so on.

With the use of classes and objects, object-oriented programming makes code maintenance simple. Because inheritance allows for code reuse, the program is simpler because you don't have to write the same code repeatedly. Data hiding is also provided by ideas like encapsulation and abstraction.

Why is C++ a partial oop?

The object-oriented features of the C language were the primary motivation behind the construction of the C++ language.

The C++ programming language is categorized as a partial object-oriented programming language despite the fact that it supports OOP concepts, including classes, objects, inheritance, encapsulation, abstraction, and polymorphism.

1) The main function must always be outside the class in C++ and is required. This means that we may do without classes and objects and have a single main function in the application.

It is expressed as an object in this case, which is the first time Pure OOP has been violated.

2) Global variables are a feature of the C++ programming language that can be accessed by any other object within the program and are defined outside of it. Encapsulation is broken here. Even though C++ encourages encapsulation for classes and objects, it ignores it for global variables.

AD

Overloading

Polymorphism also has a subset known as overloading. An existing operator or function is said to be overloaded when it is forced to operate on a new data type.

Conclusion

You will have gained an understanding of the need for object-oriented programming, what C++ OOPs are, and the fundamentals of OOPs, such as polymorphism, inheritance, encapsulation, etc., after reading this course on OOPS Concepts in C++. Along with instances of polymorphism and inheritance, you also learned about the benefits of C++ OOPs.

← Prev Next →

C++ Object and Class

Since C++ is an object-oriented language, program is designed using objects and classes in C++.

C++ Object

In C++, Object is a real world entity, for example, chair, car, pen, mobile, laptop etc.

In other words, object is an entity that has state and behavior. Here, state means data and behavior means functionality.

Object is a runtime entity, it is created at runtime.

Object is an instance of a class. All the members of the class can be accessed through object.

Let's see an example to create object of student class using s1 as the reference variable.

```
Student s1;  //creating an object of Student
```

In this example, Student is the type and s1 is the reference variable that refers to the instance of Student class.

C++ Class

In C++, class is a group of similar objects. It is a template from which objects are created. It can have fields, methods, constructors etc.

Let's see an example of C++ class that has three fields only.

```
class Student
 {
    public:
    int id;  //field or data member
    float salary; //field or data member
    String name;//field or data member
 }
```

C++ Object and Class Example

Let's see an example of class that has two fields: id and name. It creates instance of the class, initializes the object and prints the object value.

```
#include <iostream>
using namespace std;
class Student {
   public:
      int id;//data member (also instance variable)
      string name;//data member(also instance variable)
};
int main() {
    Student s1; //creating an object of Student
    s1.id = 201;
    s1.name = "Sonoo Jaiswal";
    cout<<s1.id<<endl;
    cout<<s1.name<<endl;
    return 0;
}
```

Output:

```
201
Sonoo Jaiswal
```

AD

UBER

TAP A BUTTON. GET A RIDE.
Sign up here for $20 off your first trip!

C++ Class Example: Initialize and Display data through method

Let's see another example of C++ class where we are initializing and displaying object through method.

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

```
class Student {
  public:
     int id;//data member (also instance variable)
     string name;//data member(also instance variable)
     void insert(int i, string n)
      {
         id = i;
         name = n;
      }
     void display()
      {
         cout<<id<<"  "<<name<<endl;
      }
};
int main(void) {
    Student s1; //creating an object of Student
    Student s2; //creating an object of Student
    s1.insert(201, "Sonoo");
    s2.insert(202, "Nakul");
    s1.display();
    s2.display();
    return 0;
}
```

Output:

```
201  Sonoo
202  Nakul
```

C++ Class Example: Store and Display Employee Information

Let's see another example of C++ class where we are storing and displaying employee information using method.

```
#include <iostream>
using namespace std;
class Employee {
  public:
     int id;//data member (also instance variable)
     string name;//data member(also instance variable)
     float salary;
     void insert(int i, string n, float s)
      {
         id = i;
         name = n;
         salary = s;
      }
     void display()
      {
         cout<<id<<"  "<<name<<"  "<<salary<<endl;
      }
};
int main(void) {
    Employee e1; //creating an object of Employee
    Employee e2; //creating an object of Employee
    e1.insert(201, "Sonoo",990000);
    e2.insert(202, "Nakul", 29000);
    e1.display();
    e2.display();
    return 0;
}
```

Output:

```
201  Sonoo  990000
202  Nakul  29000
```

← Prev Next →

C++ Constructor

In C++, constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new object generally. The constructor in C++ has the same name as class or structure.

In brief, A particular procedure called a constructor is called automatically when an object is created in C++. In general, it is employed to create the data members of new things. In C++, the class or structure name also serves as the constructor name. When an object is completed, the constructor is called. Because it creates the values or gives data for the thing, it is known as a constructor.

The Constructors prototype looks like this:

```
<class-name> (list-of-parameters);
```

The following syntax is used to define the class's constructor:

```
<class-name> (list-of-parameters) { // constructor definition }
```

The following syntax is used to define a constructor outside of a class:

```
<class-name>: :<class-name> (list-of-parameters){ // constructor definition}
```

Constructors lack a return type since they don't have a return value.

There can be two types of constructors in C++.

- Default constructor
- Parameterized constructor

C++ Default Constructor

A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.

Let's see the simple example of C++ default Constructor.

```
#include <iostream>
```

```
using namespace std;
class Employee
 {
   public:
        Employee()
        {
            cout<<"Default Constructor Invoked"<<endl;
        }
};
int main(void)
{
    Employee e1; //creating an object of Employee
    Employee e2;
    return 0;
}
```

Output:

```
Default Constructor Invoked
Default Constructor Invoked
```

C++ Parameterized Constructor

A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinct objects.

Let's see the simple example of C++ Parameterized Constructor.

```
#include <iostream>
using namespace std;
class Employee {
   public:
       int id;//data member (also instance variable)
       string name;//data member(also instance variable)
       float salary;
       Employee(int i, string n, float s)
        {
            id = i;
            name = n;
            salary = s;
        }
        void display()
        {
            cout<<id<<"  "<<name<<"  "<<salary<<endl;
        }
};
int main(void) {
    Employee e1 =Employee(101, "Sonoo", 890000); //creating an object of Employee
    Employee e2=Employee(102, "Nakul", 59000);
    e1.display();
    e2.display();
    return 0;
}
```

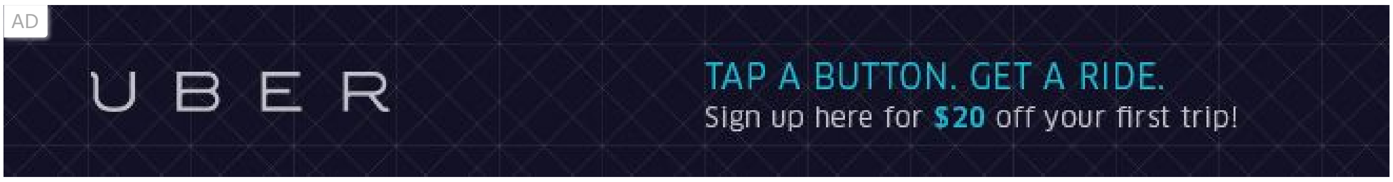
Output:

```
101  Sonoo  890000
102  Nakul  59000
```

What distinguishes constructors from a typical member function?

1. Constructor's name is the same as the class's
2. Default There isn't an input argument for constructors. However, input arguments are available for copy and parameterized constructors.
3. There is no return type for constructors.
4. An object's constructor is invoked automatically upon creation.
5. It must be shown in the classroom's open area.
6. The C++ compiler creates a default constructor for the object if a constructor is not specified (expects any parameters and has an empty body).

By using a practical example, let's learn about the various constructor types in C++. Imagine you visited a store to purchase a marker. What are your alternatives if you want to buy a marker? For the first one, you ask a store to give you a marker, given that you didn't specify the brand name or colour of the marker you wanted, simply asking for one amount to a request. So, when we just said, "I just need a marker," he would hand us whatever the most popular marker was in the market or his store. The default constructor is exactly what it sounds like! The second approach is to go into a store and specify that you want a red marker of the XYZ brand. He will give you that marker since you have brought up the subject. The parameters have been set in this instance thus. And a parameterized constructor is exactly what it sounds like! The third one requires you to visit a store and declare that you want a marker that looks like this (a physical marker on your hand). The shopkeeper will thus notice that marker. He will provide you with a new marker when you say all right. Therefore, make a copy of that marker. And that is what a copy constructor does!


AD
UBER
TAP A BUTTON. GET A RIDE.
Sign up here for $20 off your first trip!

What are the characteristics of a constructor?

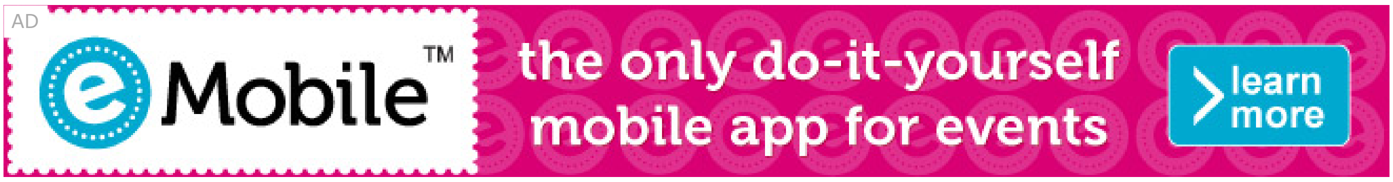
1. The constructor has the same name as the class it belongs to.
2. Although it is possible, constructors are typically declared in the class's public section. However, this is not a must.
3. Because constructors don't return values, they lack a return type.
4. When we create a class object, the constructor is immediately invoked.
5. Overloaded constructors are possible.
6. Declaring a constructor virtual is not permitted.
7. One cannot inherit a constructor.
8. Constructor addresses cannot be referenced to.
9. When allocating memory, the constructor makes implicit calls to the new and delete operators.

What is a copy constructor?

A member function known as a copy constructor initializes an item using another object from the same class-an in-depth discussion on Copy Constructors.

Every time we specify one or more non-default constructors (with parameters) for a class, we also need to include a default constructor (without parameters), as the compiler won't supply one in this circumstance. The best practice is to always declare a default constructor, even though it is not required.

A reference to an object belonging to the same class is required by the copy constructor.



```
Sample(Sample &t)
{
id=t.id;
}
```

What is a destructor in C++?

An equivalent special member function to a constructor is a destructor. The constructor creates class objects, which are destroyed by the destructor. The word "destructor," followed by the tilde () symbol, is the same as the class name. You can only define one destructor at a time. One method of destroying an object made by a constructor is to use a destructor. Destructors cannot be overloaded as a result. Destructors don't take any arguments and don't give anything back. As soon as the item leaves the scope, it is immediately called. Destructors free up the memory used by the objects the constructor generated. Destructor reverses the process of creating things by destroying them.

The language used to define the class's destructor

```
~ <class-name>()
        {
          }
```

The language used to define the class's destructor outside of it

```
<class-name>: : ~ <class-name>(){}
```

← Prev Next →

C++ Destructor

A destructor works opposite to constructor; it destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically.

A destructor is defined like constructor. It must have same name as class. But it is prefixed with a tilde sign (~).

> Note: C++ destructor cannot have parameters. Moreover, modifiers can't be applied on destructors.

C++ Constructor and Destructor Example

Let's see an example of constructor and destructor in C++ which is called automatically.

```
#include <iostream>
using namespace std;
class Employee
 {
   public:
        Employee()
        {
            cout<<"Constructor Invoked"<<endl;
        }
        ~Employee()
        {
            cout<<"Destructor Invoked"<<endl;
        }
};
int main(void)
{
    Employee e1; //creating an object of Employee
    Employee e2; //creating an object of Employee
    return 0;
}
```

Output:

```
Constructor Invoked
Constructor Invoked
Destructor Invoked
Destructor Invoked
```

← Prev Next →

C++ this Pointer

In C++ programming, **this** is a keyword that refers to the current instance of the class. There can be 3 main usage of this keyword in C++.

- It can be used **to pass current object as a parameter to another method.**
- It can be used **to refer current class instance variable.**
- It can be used **to declare indexers.**

C++ this Pointer Example

Let's see the example of this keyword in C++ that refers to the fields of current class.

```
#include <iostream>
using namespace std;
class Employee {
  public:
      int id; //data member (also instance variable)
      string name; //data member(also instance variable)
      float salary;
      Employee(int id, string name, float salary)
       {
            this->id = id;
           this->name = name;
           this->salary = salary;
       }
      void display()
       {
           cout<<id<<"  "<<name<<"  "<<salary<<endl;
       }
};
int main(void) {
    Employee e1 =Employee(101, "Sonoo", 890000); //creating an object of Employee
    Employee e2=Employee(102, "Nakul", 59000); //creating an object of Employee
    e1.display();
    e2.display();
    return 0;
}
```

Output:

```
101  Sonoo  890000
102  Nakul  59000
```

← Prev Next →

AD

Youtube For Videos Join Our Youtube Channel: Join Now

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

Learn Latest Tutorials

Splunk tutorial SPSS tutorial

C++ static

In C++, static is a keyword or modifier that belongs to the type not instance. So instance is not required to access the static members. In C++, static can be field, method, constructor, class, properties, operator and event.

Advantage of C++ static keyword

Memory efficient: Now we don't need to create instance for accessing the static members, so it saves memory. Moreover, it belongs to the type, so it will not get memory each time when instance is created.

C++ Static Field

A field which is declared as static is called static field. Unlike instance field which gets memory each time whenever you create object, there is only one copy of static field created in the memory. It is shared to all the objects.

It is used to refer the common property of all objects such as rateOfInterest in case of Account, companyName in case of Employee etc.

C++ static field example

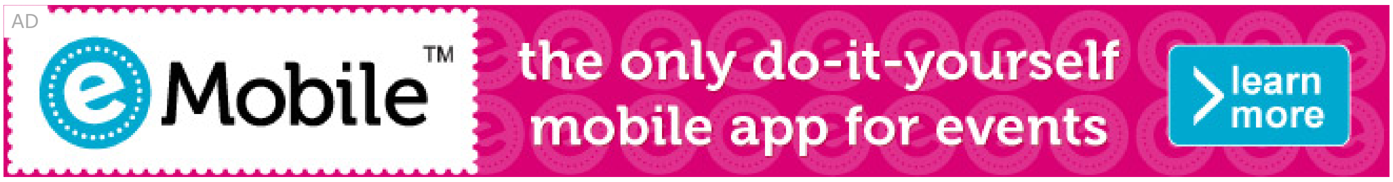
Let's see the simple example of static field in C++.

```
#include <iostream>
using namespace std;
class Account {
  public:
     int accno; //data member (also instance variable)
     string name; //data member(also instance variable)
     static float rateOfInterest;
     Account(int accno, string name)
      {
          this->accno = accno;
         this->name = name;
      }
```

```
        void display()
         {
            cout<<accno<< "<<name<< " "<<rateOfInterest<<endl;
         }
};
float Account::rateOfInterest=6.5;
int main(void) {
    Account a1 =Account(201, "Sanjay"); //creating an object of Employee
    Account a2=Account(202, "Nakul"); //creating an object of Employee
    a1.display();
    a2.display();
    return 0;
}
```

Output:

```
201 Sanjay 6.5
202 Nakul 6.5
```



C++ static field example: Counting Objects

Let's see another example of static keyword in C++ which counts the objects.

```
#include <iostream>
using namespace std;
class Account {
   public:
       int accno; //data member (also instance variable)
       string name;
       static int count;
       Account(int accno, string name)
        {
             this->accno = accno;
            this->name = name;
            count++;
        }
        void display()
        {
            cout<<accno<<" "<<name<<endl;
        }
};
int Account::count=0;
int main(void) {
    Account a1 =Account(201, "Sanjay"); //creating an object of Account
    Account a2=Account(202, "Nakul");
     Account a3=Account(203, "Ranjana");
    a1.display();
    a2.display();
    a3.display();
    cout<<"Total Objects are: "<<Account::count;
    return 0;
}
```

Output:

```
201 Sanjay
202 Nakul
203 Ranjana
Total Objects are: 3
```

← Prev Next →

C++ Structs

In C++, classes and structs are blueprints that are used to create the instance of a class. Structs are used for lightweight objects such as Rectangle, color, Point, etc.

Unlike class, structs in C++ are value type than reference type. It is useful if you have data that is not intended to be modified after creation of struct.

C++ Structure is a collection of different data types. It is similar to the class that holds different types of data.

The Syntax Of Structure

```
struct structure_name
{
     // member declarations.
}
```

In the above declaration, a structure is declared by preceding the **struct keyword** followed by the identifier(structure name). Inside the curly braces, we can declare the member variables of different types. **Consider the following situation:**

```
struct Student
{
    char name[20];
     int id;
     int age;
}
```

In the above case, Student is a structure contains three variables name, id, and age. When the structure is declared, no memory is allocated. When the variable of a structure is created, then the memory is allocated. Let's understand this scenario.

How to create the instance of Structure?

Structure variable can be defined as:

Student s;

Here, s is a structure variable of type **Student**. When the structure variable is created, the memory will be allocated. Student structure contains one char variable and two integer variable. Therefore, the memory for one char variable is 1 byte and two ints will be 2*4 = 8. The total memory occupied by the s variable is 9 byte.

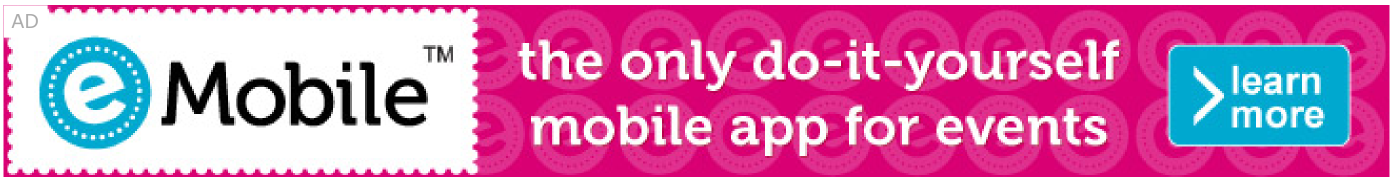
How to access the variable of Structure:

The variable of the structure can be accessed by simply using the instance of the structure followed by the dot (.) operator and then the field of the structure.

For example:

```
s.id = 4;
```

In the above statement, we are accessing the id field of the structure Student by using the **dot(.)** operator and assigns the value 4 to the id field.

AD


C++ Struct Example

Let's see a simple example of struct Rectangle which has two data members width and height.

```
#include <iostream>
using namespace std;
 struct Rectangle
{
   int width, height;


 };
int main(void) {
    struct Rectangle rec;
    rec.width=8;
    rec.height=5;
   cout<<"Area of Rectangle is: "<<(rec.width * rec.height)<<endl;
 return 0;
}
```

Output:

```
Area of Rectangle is: 40
```

C++ Struct Example: Using Constructor and Method

Let's see another example of struct where we are using the constructor to initialize data and method to calculate the area of rectangle.

```
#include <iostream>
using namespace std;
 struct Rectangle    {
   int width, height;
  Rectangle(int w, int h)
    {
        width = w;
        height = h;
    }
  void areaOfRectangle() {
    cout<<"Area of Rectangle is: "<<(width*height); }
 };
int main(void) {
    struct Rectangle rec=Rectangle(4,6);
    rec.areaOfRectangle();
   return 0;
}
```

Output:

```
Area of Rectangle is: 24
```

Structure v/s Class

Structure	Class

If access specifier is not declared explicitly, then by default access specifier will be public.	If access specifier is not declared explicitly, then by default access specifier will be private.

Syntax of Structure:
 	
struct structure_name
{
// body of the structure.
}
	Syntax of Class:
	
	class class_name
	{
	// body of the class.
	}

The instance of the structure is known as "Structure variable".	The instance of the class is known as "Object of the class".

← Prev Next →

C++ Enumeration

Enum in C++ is a data type that contains fixed set of constants.

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY and SATURDAY) , directions (NORTH, SOUTH, EAST and WEST) etc. The C++ enum constants are static and final implicitly.

C++ Enums can be thought of as classes that have fixed set of constants.

Points to remember for C++ Enum

- enum improves type safety
- enum can be easily used in switch
- enum can be traversed
- enum can have fields, constructors and methods
- enum may implement many interfaces but cannot extend any class because it internally extends Enum class

C++ Enumeration Example

Let's see the simple example of enum data type used in C++ program.

```
#include <iostream>
using namespace std;
enum week { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday };
int main()
{
    week day;
    day = Friday;
    cout << "Day: " << day+1<<endl;
    return 0;
}
```

Output:

```
Day: 5
```

← Prev

Next →

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

Learn Latest Tutorials

C++ Friend function

If a function is defined as a friend function in C++, then the protected and private data of a class can be accessed using the function.

By using the keyword friend compiler knows the given function is a friend function.

For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword friend.

Declaration of friend function in C++

```
class class_name
{
   friend data_type function_name(argument/s);          // syntax of friend function.
};
```

In the above declaration, the friend function is preceded by the keyword friend. The function can be defined anywhere in the program like a normal C++ function. The function definition does not use either the keyword **friend or scope resolution operator**.

Characteristics of a Friend function:

- The function is not in the scope of the class to which it has been declared as a friend.
- It cannot be called using the object as it is not in the scope of that class.
- It can be invoked like a normal function without using the object.
- It cannot access the member names directly and has to use an object name and dot membership operator with the member name.
- It can be declared either in the private or the public part.

C++ friend function Example

Let's see the simple example of C++ friend function used to print the length of a box.

```
#include <iostream>
using namespace std;
class Box
{
    private:
        int length;
    public:
        Box(): length(0) { }
        friend int printLength(Box); //friend function
};
int printLength(Box b)
{
   b.length += 10;
    return b.length;
}
int main()
{
    Box b;
    cout<<"Length of box: "<< printLength(b)<<endl;
    return 0;
}
```

Output:

```
Length of box: 10
```

Let's see a simple example when the function is friendly to two classes.

```
#include <iostream>
using namespace std;
class B;          // forward declarartion.
class A
{
    int x;
    public:
    void setdata(int i)
    {
        x=i;
    }
    friend void min(A,B);         // friend function.
};
```

```
class B
{
    int y;
    public:
    void setdata(int i)
    {
        y=i;
    }
    friend void min(A,B);                    // friend function
};
void min(A a,B b)
{
    if(a.x<=b.y)
    std::cout << a.x << std::endl;
    else
    std::cout << b.y << std::endl;
}
   int main()
{
   A a;
   B b;
   a.setdata(10);
   b.setdata(20);
   min(a,b);
    return 0;
 }
```

Output:

```
10
```

In the above example, min() function is friendly to two classes, i.e., the min() function can access the private members of both the classes A and B.

C++ Friend class

A friend class can access both private and protected members of the class in which it has been declared as friend.

Let's see a simple example of a friend class.

```
#include <iostream>

using namespace std;

class A
{
    int x =5;
    friend class B;           // friend class.
};
class B
{
  public:
    void display(A &a)
    {
        cout<<"value of x is : "<<a.x;
    }
};
int main()
{
    A a;
    B b;
    b.display(a);
    return 0;
}
```

Output:

```
value of x is : 5
```

In the above example, class B is declared as a friend inside the class A. Therefore, B is a friend of class A. Class B can access the private members of class A.

← Prev Next →

C++ Math Functions

C++ offers some basic math functions and the required header file to use these functions is <math.h>

Trignometric functions

Method	Description
cos(x)	It computes the cosine of x.
sin(x)	It computes the sine of x.
tan(x)	It computes the tangent of x.
acos(x)	It finds the inverse cosine of x.
asin(x)	It finds the inverse sine of x.
atan(x)	It finds the inverse tangent of x.
atan2(x,y)	It finds the inverse tangent of a coordinate x and y.

Hyperbolic functions

Method	Description
cosh(x)	It computes the hyperbolic cosine of x.
sinh(x)	It computes the hyperbolic sine of x.
tanh(x)	It computes the hyperbolic tangent of x.
acosh(x)	It finds the arc hyperbolic cosine of x.
asinh(x)	It finds the arc hyperbolic sine of x.
atanh(x)	It finds the arc hyperbolic tangent of x.

Exponential functions

Method	Description
exp(x)	It computes the exponential e raised to the power x.
frexp(value_type x,int* exp)	It breaks a number into significand and 2 raised to the power exponent.
ldexp(float x, int e)	It computes the product of x and 2 raised to the power e.
log(x)	It computes the natural logarithm of x.
log10(x)	It computes the common logarithm of x.
modf()	It breaks a number into an integer and fractional part.
exp2(x)	It computes the base 2 exponential of x.
expm1(x)	It computes the exponential raised to the power x minus one.
log1p(x)	It computes the natural logarithm of x plus one.
log2(x)	It computes the base 2 logarithm of x.
logb(x)	It computes the logarithm of x.
scalbn(x, n)	It computes the product of x and FLT_RADX raised to the power n.
scalbln(x, n)	It computes the product of x and FLT_RADX raised to the power n.
ilogb(x)	It returns the exponent part of x.

Floating point manipulation functions

Method	Description
copysign(x,y)	It returns the magnitude of x with the sign of y.
nextafter(x,y)	It represents the next representable value of x in the direction of y.

nexttoward(x,y)	It represents the next representable value of x in the direction of y.

Maximum,Minimum and Difference functions

Method	Description
fdim(x,y)	It calculates the positive difference between x and y.
fmax(x,y)	It returns the larger number among two numbers x and y.
fmin()	It returns the smaller number among two numbers x and y .

Power functions

Method	Description
pow(x,y)	It computes x raised to the power y.
sqrt(x)	It computes the square root of x.
cbrt(x)	It computes the cube root of x.
hypot(x,y)	It finds the hypotenuse of a right angled triangle.

Nearest integer operations

Method	Description
ceil(x)	It rounds up the value of x.
floor(x)	It rounds down the value of x.
round(x)	It rounds off the value of x.
lround(x)	It rounds off the value of x and cast to long integer.
llround(x)	It rounds off the value of x and cast to long long integer.
fmod(n,d)	It computes the remainder of division n/d.
trunc(x)	It rounds off the value x towards zero.
rint(x)	It rounds off the value of x using rounding mode.

lrint(x)	It rounds off the value of x using rounding mode and cast to long integer.
llrint(x)	It rounds off the value x and cast to long long integer.
nearbyint(x)	It rounds off the value x to a nearby integral value.
remainder(n,d)	It computes the remainder of n/d.
remquo()	It computes remainder and quotient both.

Other functions

Method	Description
fabs(x)	It computes the absolute value of x.
abs(x)	It computes the absolute value of x.
fma(x,y,z)	It computes the expression x*y+z.

Macro functions

Method	Description
fpclassify(x)	It returns the value of type that matches one of the macro constants.
isfinite(x)	It checks whether x is finite or not.
isinf()	It checks whether x is infinite or not.
isnan()	It checks whether x is nan or not.
isnormal(x)	It checks whether x is normal or not.
signbit(x)	It checks whether the sign of x is negative or not.

Comparison macro functions

Method	Description
isgreater(x,y)	It determines whether x is greater than y or not.
isgreaterequal(x,y)	It determines whether x is greater than or equal to y or not.

less(x,y)	It determines whether x is less than y or not.
islessequal(x,y)	It determines whether x is less than or equal to y.
islessgreater(x,y)	It determines whether x is less or greater than y or not.
isunordered(x,y)	It checks whether x can be meaningfully compared or not.

Error and gamma functions

Method	Description
erf(x)	It computes the error function value of x.
erfc(x)	It computes the complementary error function value of x.
tgamma(x)	It computes the gamma function value of x.
lgamma(x)	It computes the logarithm of a gamma function of x.

← Prev Next →

AD

Youtube For Videos Join Our Youtube Channel: Join Now

Feedback

- Send your Feedback to feedback@javatpoint.com