OOP in c++

Scope of Variables in C++

Local Variables

Variables defined within a function or block are said to be local to those functions.

- Anything between '{' and '}' is said to inside a block.
- Local variables do not exist outside the block in which they are declared, i.e. they **can not** be accessed or used outside that block.
- **Declaring local variables**: Local variables are declared inside a block.

```
using namespace std;

void func()
{
    // this variable is local to the
    // function func() and cannot be
    // accessed outside this function
    int age=18;
}

int main()
{
    cout<<"Age is: "<<age;

    return 0;
}
```

Output:

```
Error: age was not declared in this scope
```

OOP in c++

Correct code

```
using namespace std;

void func()
{
    // this variable is local to the
    // function func() and cannot be
    // accessed outside this function
    int age=18;
    cout<<age;
}

int main()
{
    cout<<"Age is: ";
    func();

    return 0;
}
```

Output:

```
Age is: 18
```

Global Variables

As the name suggests, Global Variables can be accessed from any part of the program.

- They are available through out the life time of a program.

OOP in c++

- They are declared at the top of the program outside all of the functions or blocks.
- **Declaring global variables**: Global variables are usually declared outside of all of the functions and blocks, at the top of the program. They can be accessed from any portion of the program.

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

// global variable
int global = 5;

// global variable accessed from
// within a function
void display()
{
    cout<<global<<endl;
}

// main function
int main()
{
    display();

    // changing value of global
    // variable from main function
    global = 10;
    display();
}
```

Output:

OOP in c++

```
5

10
```

look

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

// global variable
int global = 5;

// main function
int main()
{
    // local variable with same
    // name as that of global variable

    int global = 2;
    cout << global << endl;
}
```

Look at the above program. The variable “global” declared at the top is global and stores the value 5 where as that declared within main function is local and stores a value 2. So, the question is when the value stored in the variable named “global” is printed from the main function then what will be the output? 2 or 5?

- Usually when two variable with same name are defined then the compiler produces a compile time error. But if the variables are defined in different scopes then the compiler allows it.

OOP in c++

- Whenever there is a local variable defined with same name as that of a global variable then the **compiler will give precedence to the local variable**

How to access a global variable when there is a local variable with same name?

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

// Global x
int x = 0;

int main()
{
  // Local x
  int x = 10;
  cout << "Value of global x is " << ::x;
  cout<< "\nValue of local x is " << x;
  return 0;
}
```

Output:

```
Value of global x is 0

Value of local x is 10
```

OOP in c++

What Are OOPS Concepts In C++?

OOPs, or Object-oriented programming is an approach or a programming pattern where the programs are structured around objects rather than functions and logic. It makes the data partitioned into two memory areas, i.e., data and functions, and helps make the code flexible and modular.

Object-oriented programming mainly focuses on objects that are required to be manipulated. In OOPs, it can represent data as objects that have attributes and functions.

Why Do You Need Object-Oriented Programming?

The earlier approaches to programming were not that good, and there were several limitations as well. Like in procedural-oriented programming, you cannot reuse the code again in the program, and there was the problem of global data access, and the approach couldn't solve the real-world problems very well.

In object-oriented programming, it is easy to maintain the code with the help of classes and objects. Using inheritance, there is code reusability, i.e., you don't have to write the same code again and again, which increases the simplicity of the program. Concepts like encapsulation and abstraction provide data hiding as well.

OOP in c++

Now have a look at some basic concepts of C++ OOPs.

Basic Object-Oriented Programming (OOPS) Concept in C++

There are some basic concepts that act as the building blocks of OOPs.

- Classes & Objects
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

- **Object**

An Object can be defined as an entity that has a state and behavior, or in other words, anything that exists physically in the world is called an object. It can represent a dog, a person, a table, etc.

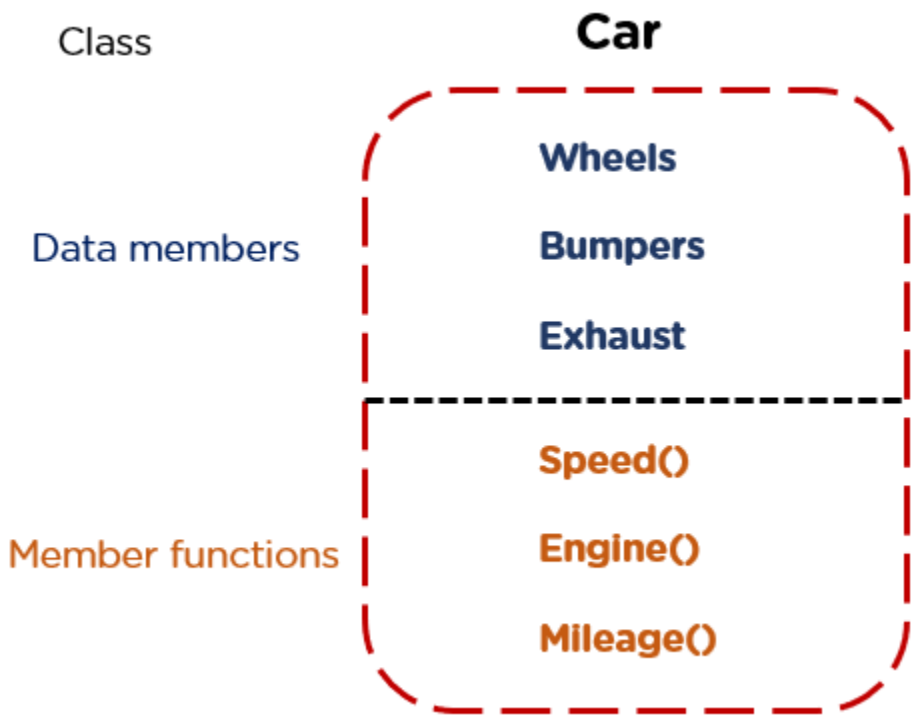
An object means the combination of data and programs, which further represent an entity.

OOP in c++

- ## Classes

Class can be defined as a blueprint of the object. It is basically a collection of objects which act as building blocks.

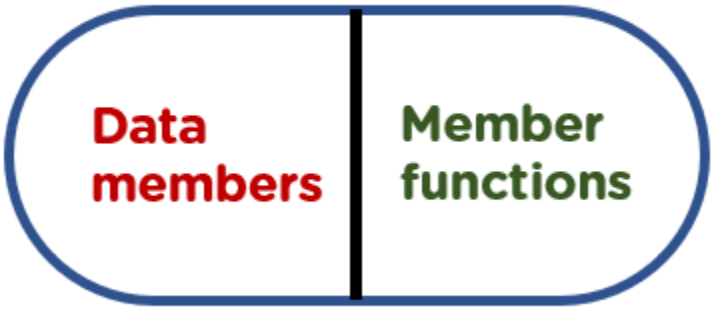
class.



- ## Abstraction

Abstraction helps in the data hiding process. It helps in displaying the essential features without showing the details or the functionality to the user. It avoids unnecessary information or irrelevant details and shows only that specific part which the user wants to see.

OOP in c++



- ## Encapsulation

  The wrapping up of data and functions together in a single unit is known as encapsulation. It can be achieved by making the data members' scope private and the member function’s scope public to access these data members. Encapsulation makes the data non-accessible to the outside world.

- ## Inheritance

  nheritance is the process in which two classes have an is-a relationship among each other and objects of one class acquire properties and features of the other class. The class which inherits the features is known as the child class, and the

class whose features it inherited is called the parent class. For example, Class Vehicle is the parent class, and Class Bus, Car, and Bike are child classes.

- **Polymorphism**

  Polymorphism means many forms. It is the ability to take more than one form. It is a feature that provides a function or an operator with more than one definition. It can be implemented using function overloading, operator overload, function overriding, virtual function.

Advantages of OOPs

There are various advantages of object-oriented programming.

- OOPs provide reusability to the code and extend the use of existing classes.
- In OOPs, it is easy to maintain code as there are classes and objects, which helps in making it easy to maintain rather than restructuring.
- It also helps in data hiding, keeping the data and information safe from leaking or getting exposed.
- Object-oriented programming is easy to implement.

OOP in c++

C++ Classes and Objects

++ is an object-oriented programming language.

Everything in C++ is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an **object**. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

Attributes and methods are basically **variables** and **functions** that belongs to the class. These are often referred to as "class members".

A class is a user-defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the `class` keyword:

Example

Create a class called "`MyClass`":

```
class MyClass {       // The class
  public:             // Access specifier
    int myNum;        // Attribute (int variable)
    string myString;  // Attribute (string variable)
```

```
};
```

Example explained

- The `class` keyword is used to create a class called `MyClass`.
- The `public` keyword is an **access specifier**, which specifies that members (attributes and methods) of the class are accessible from outside the class. You will learn more about access specifiers later.
- Inside the class, there is an integer variable `myNum` and a string variable `myString`. When variables are declared within a class, they are called **attributes**.
- At last, end the class definition with a semicolon `;`.

Create an Object

In C++, an object is created from a class. We have already created the class named `MyClass`, so now we can use this to create objects.

To create an object of `MyClass`, specify the class name, followed by the object name.

To access the class attributes (`myNum` and `myString`), use the dot syntax (`.`) on the object:

Example

Create an object called "`myObj`" and access the attributes:

OOP in c++

```
class MyClass {       // The class
  public:             // Access specifier
    int myNum;        // Attribute (int variable)
    string myString;  // Attribute (string variable)
};

int main() {
  MyClass myObj;  // Create an object of MyClass

  // Access attributes and set values
  myObj.myNum = 15;
  myObj.myString = "Some text";

  // Print attribute values
  cout << myObj.myNum << "\n";
  cout << myObj.myString;
  return 0;
}
```

Multiple Objects

```
class Car {
  public:
    string brand;
    string model;
    int year;
};

int main() {
  // Create an object of Car
  Car carObj1;
```

```
  carObj1.brand = "BMW";
  carObj1.model = "X5";
  carObj1.year = 1999;

  // Create another object of Car
  Car carObj2;
  carObj2.brand = "Ford";
  carObj2.model = "Mustang";
  carObj2.year = 1969;

  // Print attribute values
  cout << carObj1.brand << " " << carObj1.model << " " <<
carObj1.year << "\n";
  cout << carObj2.brand << " " << carObj2.model << " " <<
carObj2.year << "\n";
  return 0;
}
```

Access Specifiers

The `public` keyword is an **access specifier.** Access specifiers define how the members (attributes and methods) of a class can be accessed. In the example above, the members are `public` - which means that they can be accessed and modified from outside the code.

However, what if we want members to be private and hidden from the outside world?

In C++, there are three access specifiers:

- `public` - members are accessible from outside the class

OOP in c++

- `private` - members cannot be accessed (or viewed) from outside the class
- `protected` - members cannot be accessed from outside the class, however, they can be accessed in inherited classes. You will learn more about Inheritance later.

```
class MyClass {
  public:    // Public access specifier
    int x;   // Public attribute
  private:   // Private access specifier
    int y;   // Private attribute
};

int main() {
  MyClass myObj;
  myObj.x = 25;  // Allowed (public)
  myObj.y = 50;  // Not allowed (private)
  return 0;
}
```

```
error: y is private
```

Note: By default, all members of a class are `private` if you don't specify an access specifier:

Example

```
class MyClass {
  int x;   // Private attribute
  int y;   // Private attribute
};
```

OOP in c++

C++ Constructors

In this tutorial, we will learn about the C++ constructor and its type with the help examples.

A constructor is a special type of member function that is called automatically when an object is created.

In C++, a constructor has the same name as that of the class and it does not have a return type. For example,

```
class  Wall {
  public:
    // create a constructor
    Wall() {
      // code
    }
};
```

Here, the function `Wall()` is a constructor of the class `Wall`. Notice that the constructor

- has the same name as the class,
- does not have a return type, and
- is `public`

OOP in c++

C++ Default Constructor

A constructor with no parameters is known as a **default constructor**. In the example above, `Wall()` is a default constructor.

Example 1: C++ Default Constructor

```
// C++ program to demonstrate the use of default constructor

#include <iostream>
using namespace std;

// declare a class
class  Wall {
  private:
    double length;

  public:
    // default constructor to initialize variable
    Wall() {
      length = 5.5;
      cout << "Creating a wall." << endl;
      cout << "Length = " << length << endl;
    }
};

int main() {
  Wall wall1;
  return 0;
}
```

Output

OOP in c++

```
Creating a Wall
Length = 5.5
```

C++ Parameterized Constructor

Example 2: C++ Parameterized Constructor

```
// C++ program to calculate the area of a wall

#include <iostream>
using namespace std;

// declare a class
class Wall {
  private:
    double length;
    double height;

  public:
    // parameterized constructor to initialize variables
    Wall(double len, double hgt) {
      length = len;
      height = hgt;
    }

    double calculateArea() {
      return length * height;
    }
};

int main() {
  // create object and initialize data members
  Wall wall1(10.5, 8.6);
```

OOP in c++

```
  Wall wall2(8.5, 6.3);

  cout << "Area of Wall 1: " << wall1.calculateArea() << endl;
  cout << "Area of Wall 2: " << wall2.calculateArea();

  return 0;
}
```

Output

```
Area of Wall 1: 90.3
Area of Wall 2: 53.55
```