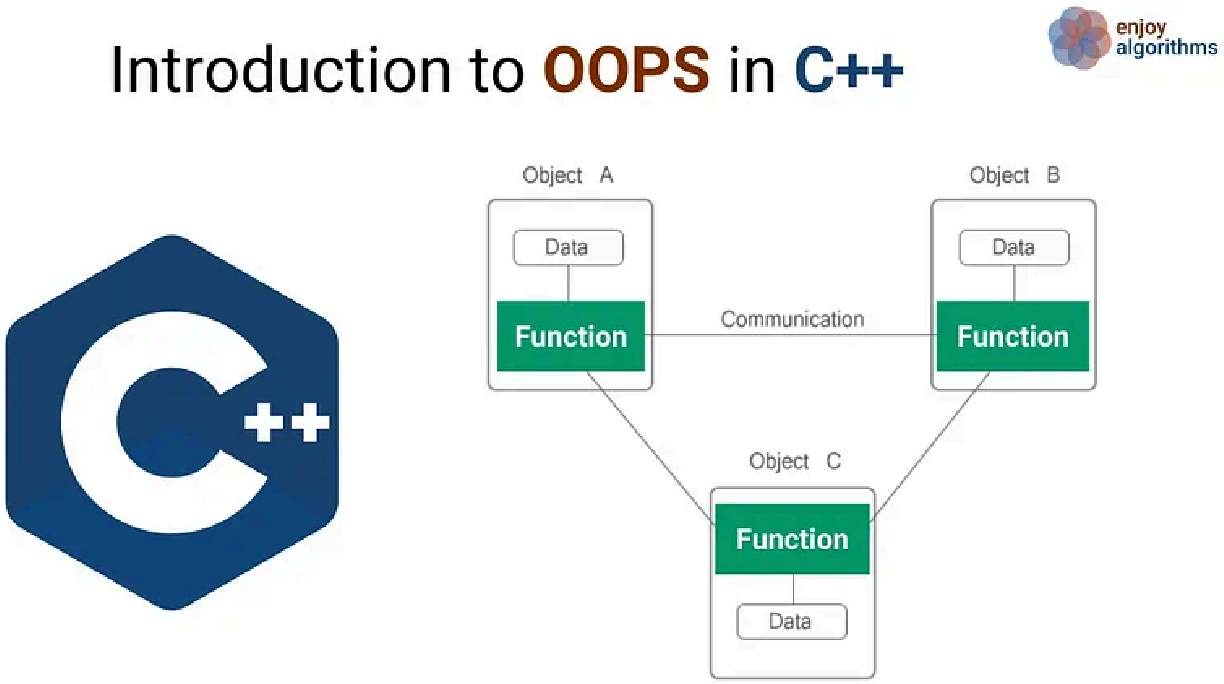
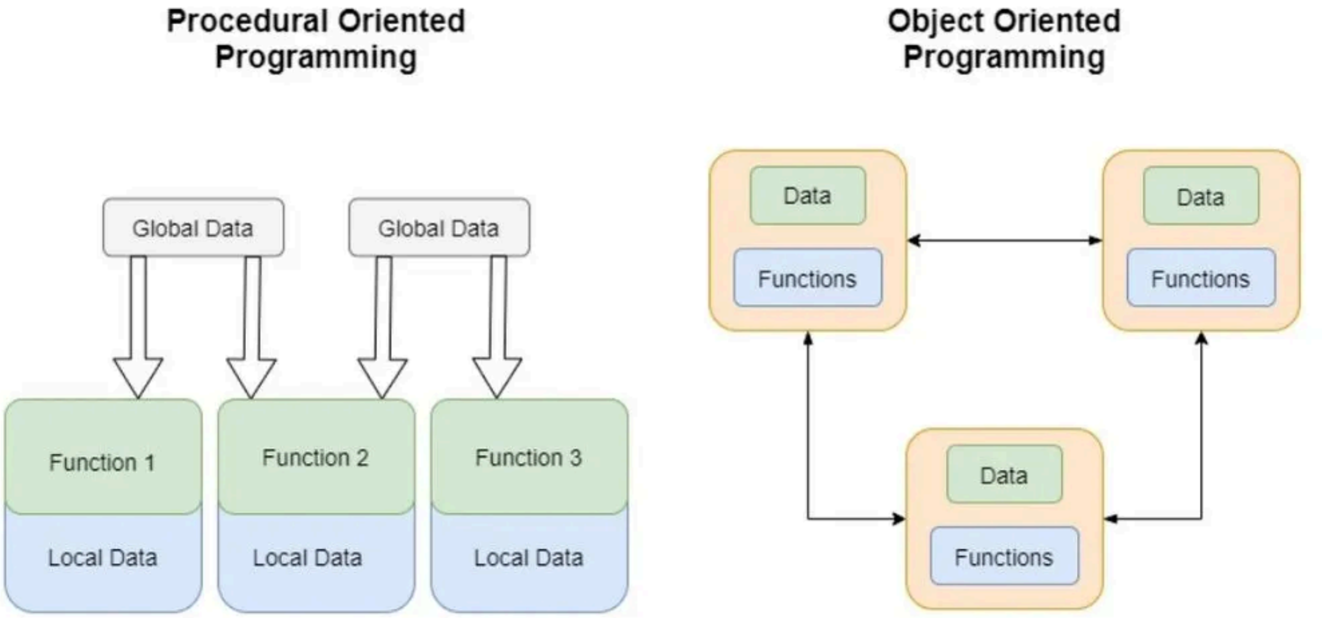
Introduction to OOPS in C++



Why the idea of “OOPS” is essential?

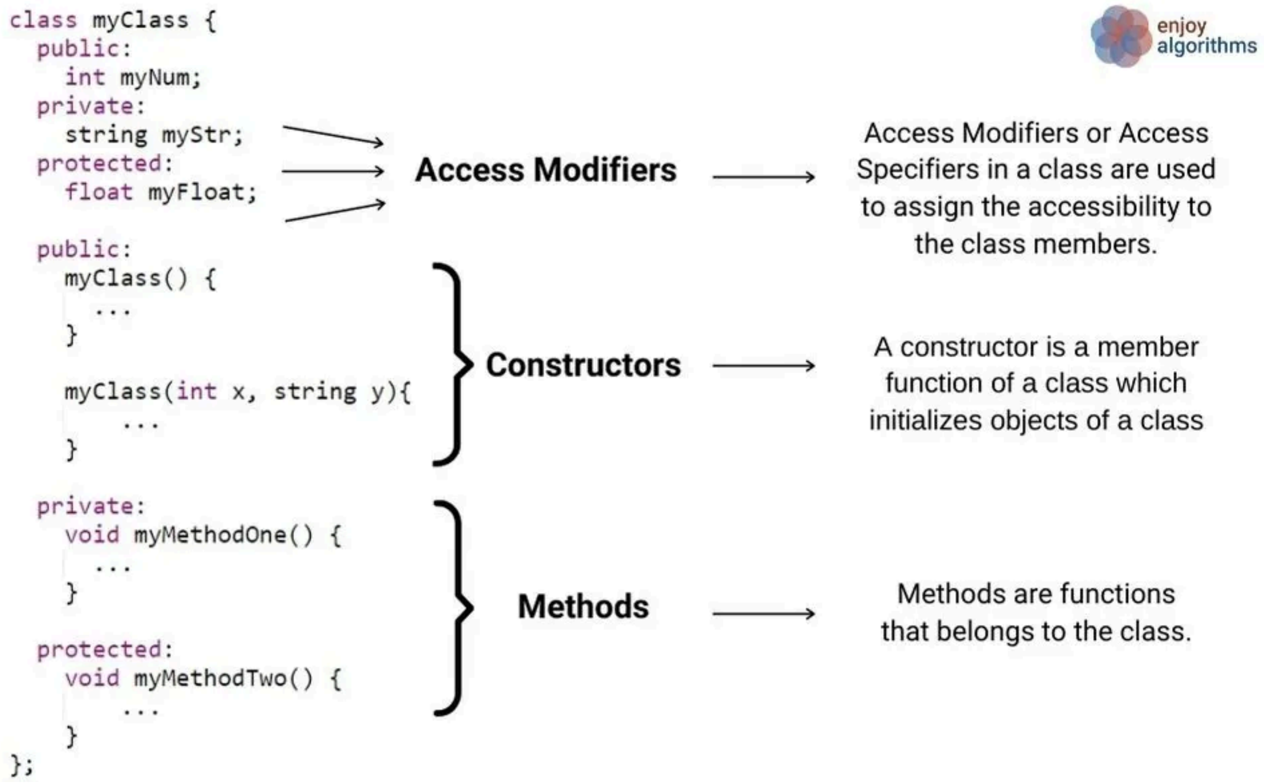
Object-Oriented Programming binds together the data and the methods in the form of an object and selectively exposes the data to other objects. It primarily revolves around classes and objects — definition, instantiation, relationship, communication, etc. On the other hand, the building block of procedural programming is procedures or functions that perform data operations. Explore and Think.



OOPS brings the concept of class, a collection of data and methods that relies on the operation of an object. The idea of class and object brings the dynamics within a code and, most importantly, makes the code reusable, unlike procedural language. OOPS is important in modern software development as it introduces many features such as Inheritance, Encapsulation, Abstraction, Polymorphism.

Class in C++

Why the idea of “class” is essential in OOPs? Here is an analogy to understand — The world around us has a lot more structure than just numbers, strings, etc. There are cars, people, trees, all kinds of things that are the structure involved in real-world design. So if we don’t want to be limited, then: how can we group values in a meaningful fashion to define such structure and define methods to use them?



```
class myClass {
  public:
    int myNum;
  private:
    string myStr;
  protected:
    float myFloat;

  public:
    myClass() {
      ...
    }

    myClass(int x, string y){
        ...
    }

  private:
    void myMethodOne() {
      ...
    }

  protected:
    void myMethodTwo() {
        ...
    }
};
```

Access Modifiers or Access Specifiers in a class are used to assign the accessibility to the class members.

A constructor is a member function of a class which initializes objects of a class

Methods are functions that belongs to the class.

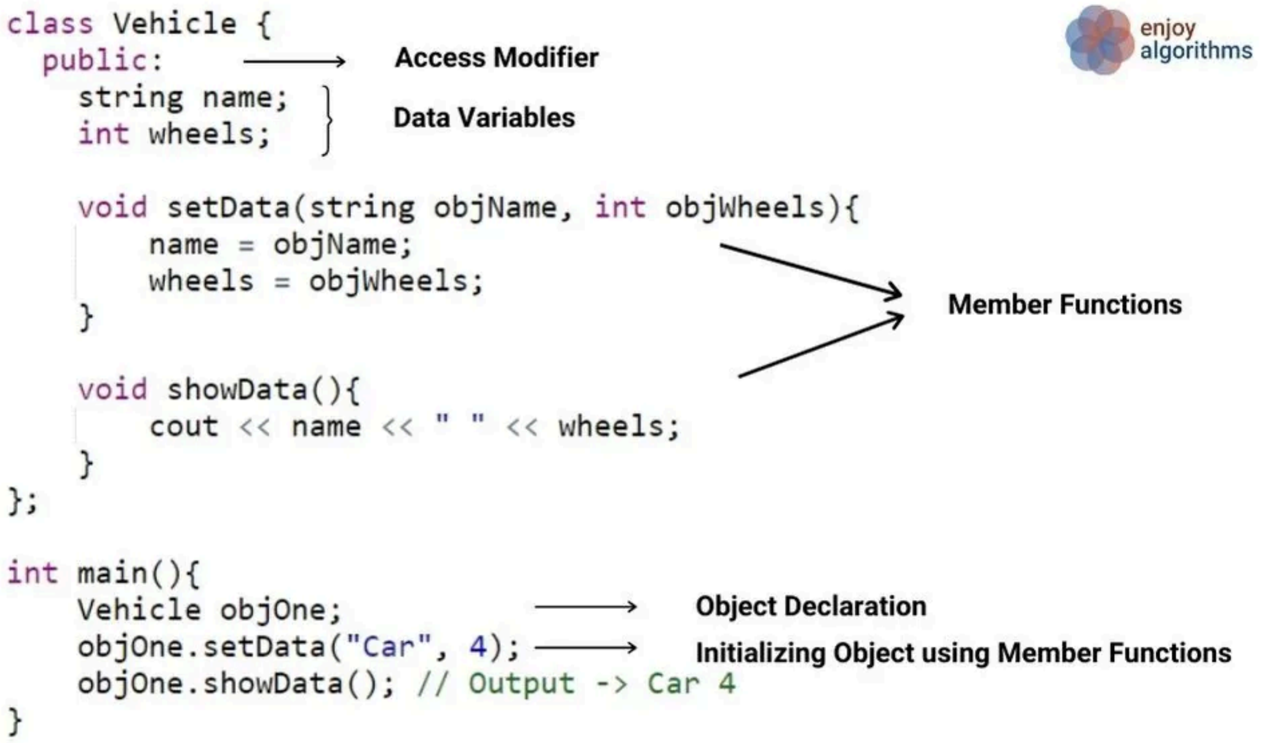
The idea is simple — If we want to solve real-world problems, we'd like to mimic the real world's structure with values that exhibit the same structure. The idea of "Class" helps us define structures similar to the real world and build scalable and reusable software. A class definition provides us two main capabilities:

1. It allows us to different group types of values to represent some more complex structures.
2. It gives us the chance to represent the world through values and animate that world-representation through interactions (method calls).

Objects in C++

Objects are instances of classes created with specific data, where a class is an abstract blueprint used to create more specific, concrete objects. Each object contains data and methods to manipulate the data.

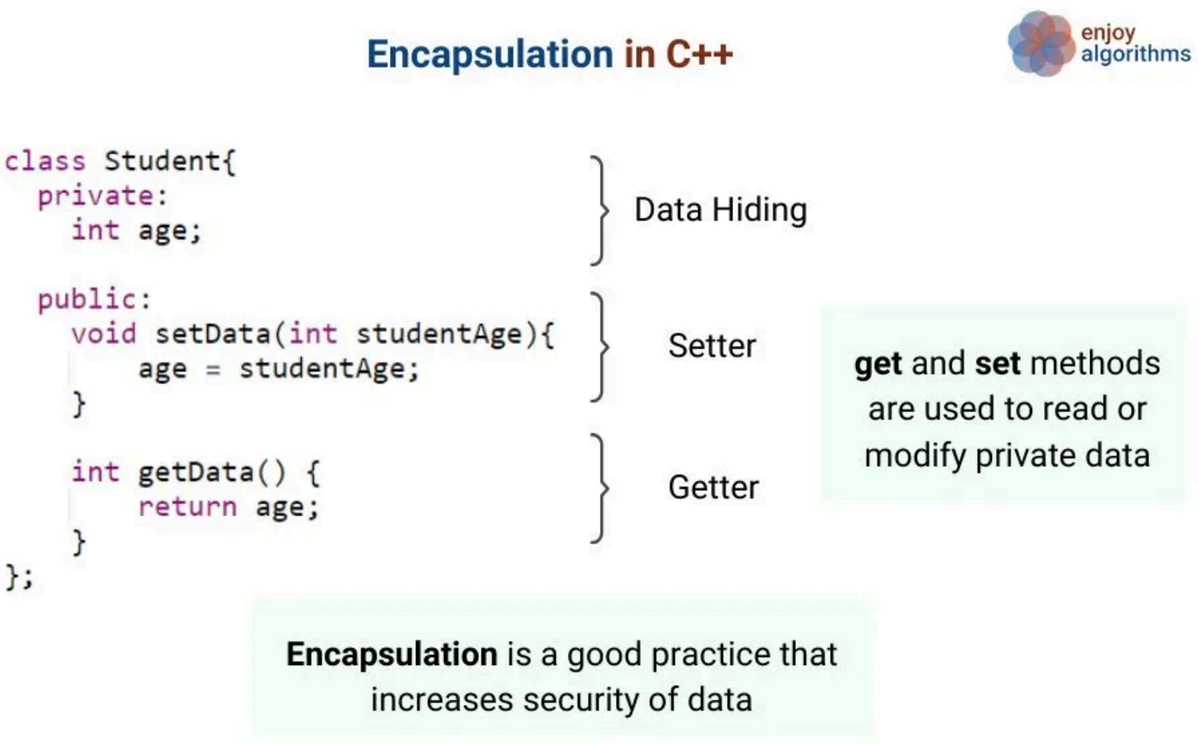
- Critical idea 1: When a class is defined, no memory is allocated, but memory is allocated when an object is created.
- Critical idea 2: The objects interact by sending messages to one another. Objects can also interact without knowing the details of each other's data or code.



```
class Vehicle {
  public:
    string name;
    int wheels;

    void setData(string objName, int objWheels){
        name = objName;
        wheels = objWheels;
    }

    void showData(){
        cout << name << " " << wheels;
    }
};

int main(){
    Vehicle objOne;
    objOne.setData("Car", 4);
    objOne.showData(); // Output -> Car 4
}
```

Encapsulation in C++

On large-scale software projects, we often need to take care of the accessibility and readability of our data. To achieve this, we use the concept of encapsulation.



Encapsulation in C++

```
class Student{
  private:
    int age;

  public:
    void setData(int studentAge){
        age = studentAge;
    }

    int getData() {
        return age;
    }
};
```

get and **set** methods are used to read or modify private data

Encapsulation is a good practice that increases security of data

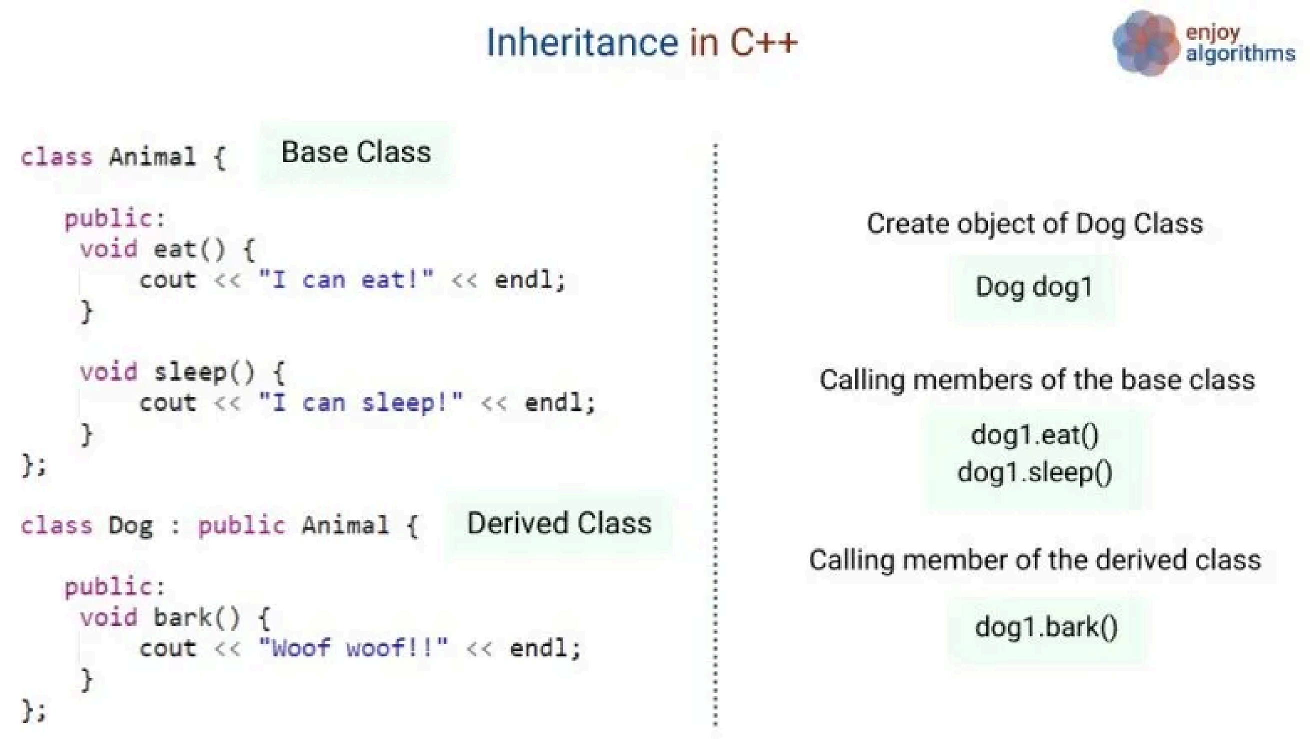
- It hides implementation details of an object from the class users and protects an object's data. Technically, the class data are hidden from other classes and can be accessed only through the member function of the class in which it is declared.

- It combines data members and functions to make our code cleaner and easy to read. In simple words, it provides an abstraction between an object and users of the object.

- Encapsulation can be achieved by declaring all the class variables as private and writing public methods in the class to set and get the values of variables.

Inheritance in C++

On large-scale software projects that involve millions of code lines, we often need to re-use existing code or extend functionalities according to requirements. To achieve this, we use the concept of Inheritance.



Inheritance in C++

```
class Animal {

   public:
    void eat() {
        cout << "I can eat!" << endl;
    }

    void sleep() {
        cout << "I can sleep!" << endl;
    }
};

class Dog : public Animal {

   public:
    void bark() {
        cout << "Woof woof!!" << endl;
    }
};
```

Base Class

Derived Class

Create object of Dog Class

Dog dog1

Calling members of the base class

dog1.eat()
dog1.sleep()

Calling member of the derived class

dog1.bark()

Inheritance is a process in which one object acquires all the properties and behaviors of its parent object.

- It enables code reusability and saves time. Inheritance is used to declare characteristics of classes inheriting it.

- It is essential for extensibility. After creating one superclass, you can use its features to create several sub-classes that inherit some features from their parent class and have few additional features.

- Inheritance supports the code reusability in the OOPS. How? Think!

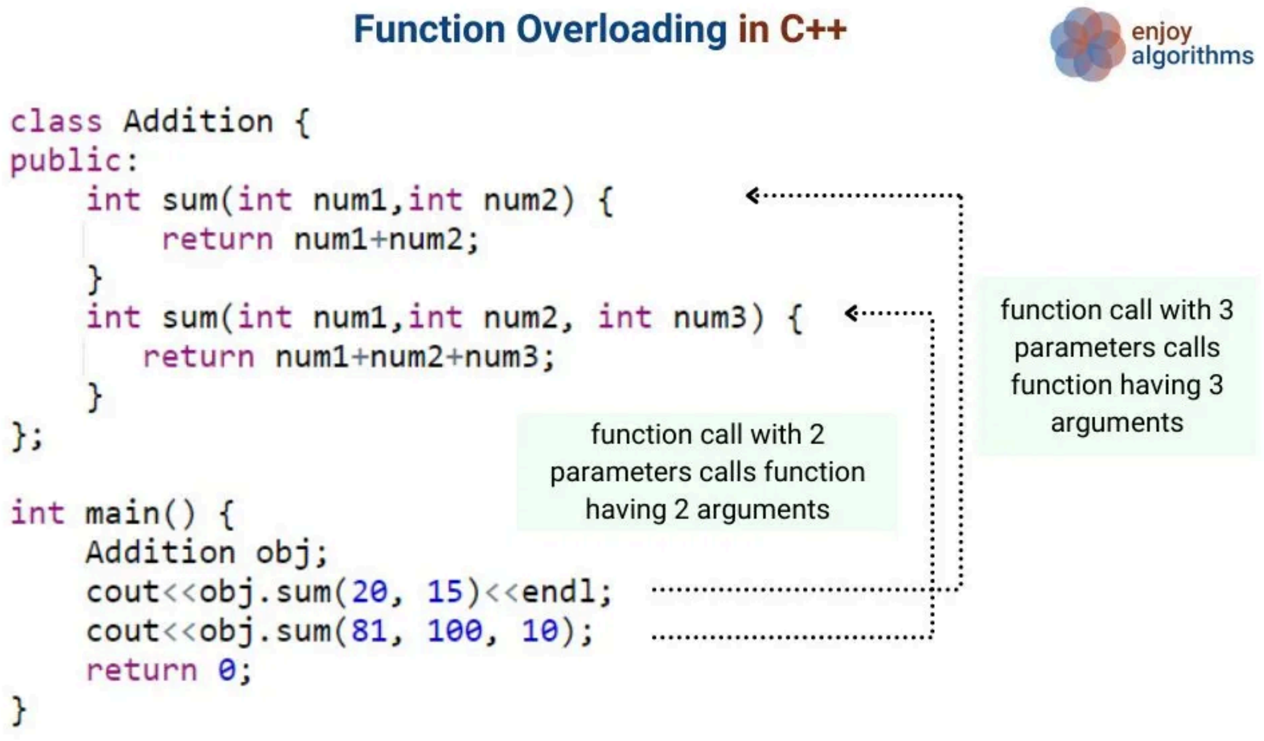
Polymorphism in C++

Why is polymorphism important in OOPS? Here are some critical reasons to think and explore:

- It performs a single action in different ways and presents the common interface to access similar structures.
- It is crucial for building modular & extensible applications. In other words, polymorphism supports code reusability.
- Instead of complex conditional statements describing the different courses of action, we can create interchangeable objects that we select based on our needs. So one benefit of polymorphism is — code working with the different classes does not need to know which class it uses.
- It makes the class responsible for its code and data. This is the opposite of the old way of doing things where code was separate from the data.

Function overloading in C++

Function overloading is a C++ programming feature that allows us to have more than one function having the same name but a different parameter list. It means the data type and sequence of the parameters, for example, the parameters list of a function **myfun(int a, int b)** is **(int, float)** which is different from the function myfun(float a, int b) parameter list **(float, int).**



Function Overloading in C++

```
class Addition {
public:
    int sum(int num1,int num2) {
        return num1+num2;
    }
    int sum(int num1,int num2, int num3) {
       return num1+num2+num3;
    }
};

int main() {
    Addition obj;
    cout<<obj.sum(20, 15)<<endl;
    cout<<obj.sum(81, 100, 10);
    return 0;
}
```

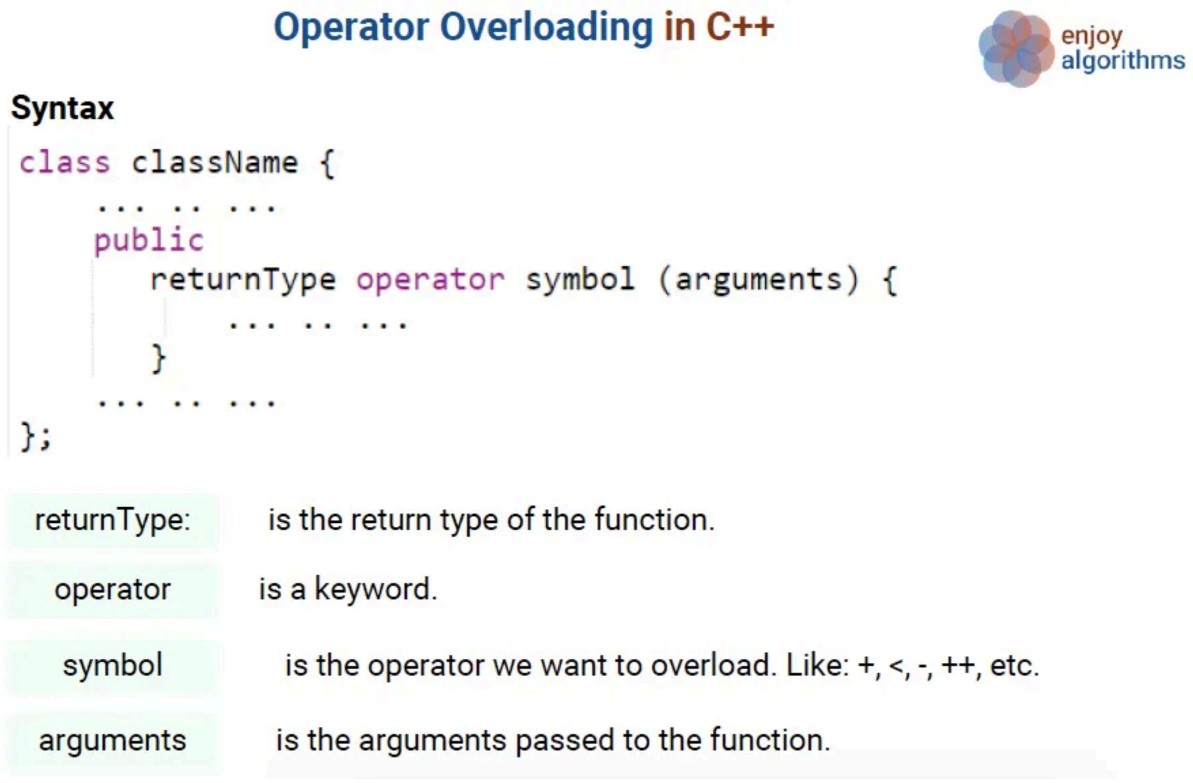
function call with 3 parameters calls function having 3 arguments

function call with 2 parameters calls function having 2 arguments

The main advantage of function overloading is to improve the code readability and allows code reusability. In the code below, we can see how we are able to have more than one function for the same task(addition) with different parameters. This allows us to add two integer numbers and three integer numbers, and if we want, we could have some more functions with the same name and four or five arguments.

Operator overloading in C++

Operator overloading means that the operation performed by the operator depends on the type of operands provided to the operator method. It is similar to function overloading, where we have many versions of the same function differentiated by their parameter lists.

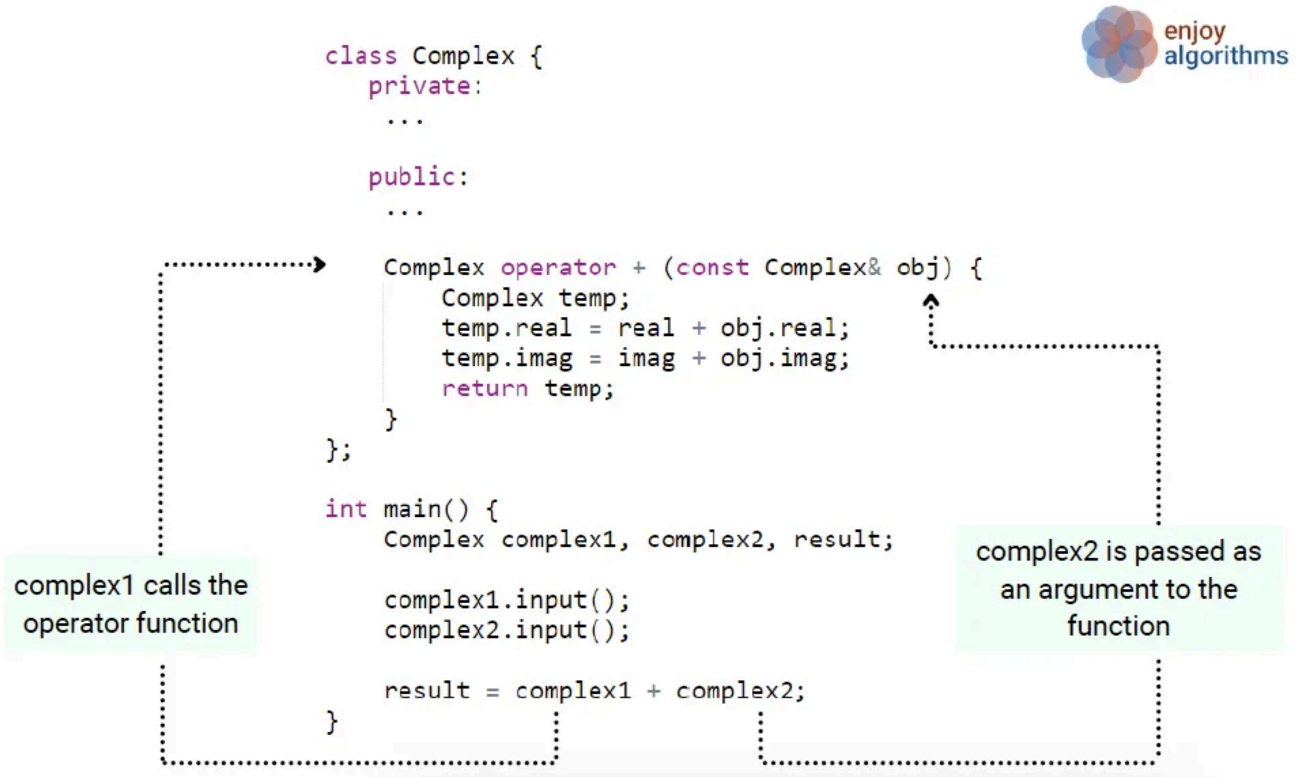


Operator Overloading in C++

Syntax

```
class className {
    ... .. ...
    public
       returnType operator symbol (arguments) {
           ... .. ...
       }
    ... .. ...
};
```

returnType:	is the return type of the function.
operator	is a keyword.
symbol	is the operator we want to overload. Like: +, <, -, ++, etc.
arguments	is the arguments passed to the function.



```
class Complex {
   private:
    ...

   public:
    ...

    Complex operator + (const Complex& obj) {
        Complex temp;
        temp.real = real + obj.real;
        temp.imag = imag + obj.imag;
        return temp;
    }
};

int main() {
    Complex complex1, complex2, result;

    complex1.input();
    complex2.input();

    result = complex1 + complex2;
}
```

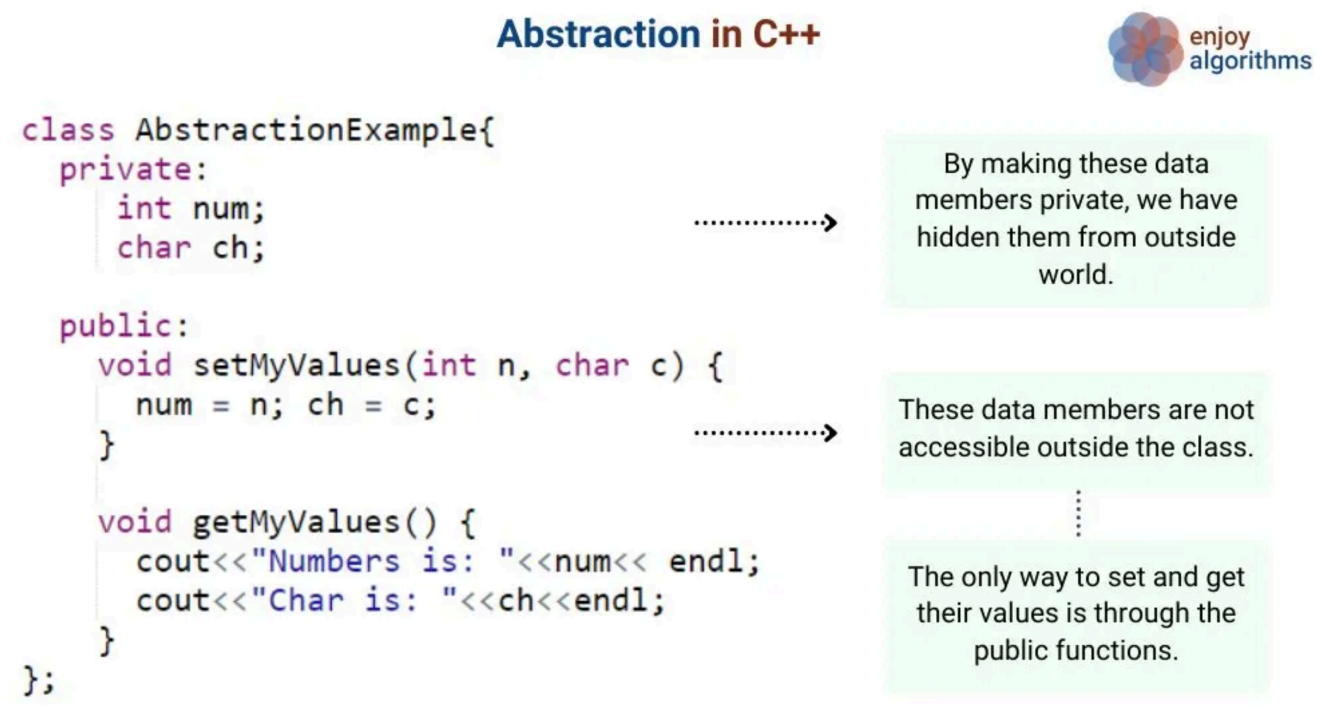
complex1 calls the operator function

complex2 is passed as an argument to the function

C++ lets us extend operator overloading to user-defined types (classes) i.e., a programmer can provide their own operator to a class by overloading the built-in operator to perform some specific computation when the operator is used on objects of that class.

Abstraction in C++

Abstraction means providing only the essential or relevant information and hiding the background or technical implementation details. For example: to drive a car, one only needs to know the driving process and not the mechanics of the car engine.



Abstraction in C++

```
class AbstractionExample{
  private:
     int num;
     char ch;

  public:
    void setMyValues(int n, char c) {
      num = n; ch = c;
    }

    void getMyValues() {
      cout<<"Numbers is: "<<num<< endl;
      cout<<"Char is: "<<ch<<endl;
    }
};
```

By making these data members private, we have hidden them from outside world.

These data members are not accessible outside the class.

The only way to set and get their values is through the public functions.

In an object-oriented language, data abstraction is implemented through classes that define properties of an object like data and functions. OOPS has the mechanism of using class functions through related objects without going into details about how the functions are written. Abstraction makes the modifications easier as the interface to access the function remains the same. For example, if the type and working of the car engine change, it does not change the driving process.

Advantages of data abstraction:

- It helps the user to avoid writing low-level code.
- It helps to prevent code duplication and increases its reusability.
- It helps to change the internal implementation of class independently without affecting the user.
- It helps increase the security of an application or program as only essential details are provided to the user.

Enjoy learning, Enjoy OOPS!