What is object-oriented programming?

Object-oriented programming combines a group of data attributes with functions or methods into a unit called an "object." Typically, OOP languages are class-based, which means that a class defines the data attributes and functions as a blueprint for creating objects, which are instances of the class. Popular class-based OOP languages include Java, Python, and C++. Multiple independent objects may be instantiated—or represented—from the same class and interact with each other in complex ways.

A simple example would be a class representing a person. The person class would contain attributes to represent information such as the person’s age, name, height, etc. The class definition might also contain functions such as "sayMyName" which would simply print that person’s name to the screen.

A family could be constructed by instantiating person objects from the class for each member of the family. Each person object would contain different data attributes since each person is unique.

The four basics of object-oriented programming

Object-oriented programming has four basic concepts: encapsulation, abstraction, inheritance, and polymorphism. Even if these concepts seem incredibly complex, understanding the general framework of how they work will help you understand the basics of an OOP computer program. Below, we outline these four basic principles and what they entail:

Encapsulation

Abstraction

\*Inheritance

Polymorphism

1. Encapsulation

The word, “encapsulate,” means to enclose something. Just like a pill "encapsulates" or contains the medication inside of its coating, the principle of encapsulation works in a similar way in OOP: by forming a protective barrier around the information contained within a class from the rest of the code.

In OOP, we encapsulate by binding the data and functions which operate on that data into a single unit, the class. By doing so, we can hide private details of a class from the outside world and only expose functionality that is important for interfacing with it. When a class does not allow calling code access to its private data directly, we say that it is well encapsulated.

*Example: Elaborating on the person class example from earlier, we might have private data in the class, such as "socialSecurityNumber," that should not be exposed to other objects in the program. By encapsulating this data member as a private variable in the class, outside code would not have direct access to it, and it would remain safe within that person’s object.*

*If a method is written in the person class to perform, say, a bank transaction called "bankTransaction()," that function could then access the "socialSecurityNumber" variable as necessary. The person’s private data would be well encapsulated in such a class.*

2. Abstraction

Often, it’s easier to reason and design a program when you can separate the interface of a class from its implementation, and focus on the interface. This is akin to treating a system as a “black box,” where it’s not important to understand the gory inner workings in order to reap the benefits of using it.

This process is called “abstraction” in OOP, because we are abstracting away the gory implementation details of a class and only presenting a clean and easy-to-use interface via the class’ member functions. Carefully used, abstraction helps isolate the impact of changes made to the code, so that if something goes wrong, the change will only affect the implementation details of a class and not the outside code.

*Example: Think of a stereo system as an object with a complex logic board on the inside. It has buttons on the outside to allow for interaction with the object. When you press any of the buttons, you're not thinking about what happens on the inside because you can't see it. Even though you can't see the logic board completing these functions as a result of pressing a button, it's still performing them., albeit hidden to you.*

This is the concept of abstraction, which is incredibly useful in all areas of engineering and also applied to great effect in object-oriented programming.

*Example: In OOP, we might have a class defined to represent the human body. One might define some functions as part of its publicly facing interface such as “walk()” or “eatFood().” Calling code could call these functions and remain completely oblivious to the complex inner workings of the human body and its necessary functions to perform the act of walking or eating. These details are completely hidden in the implementation of the walk() and eatFood() body functions and are, therefore, us abstracted away from the end user. In these cases, it’s not important for calling code to understand how the brain coordinates walking or how the stomach manages digesting the food, but rather simply that a human walked or ate.*

3. Inheritance

Object-oriented languages that support classes almost always support the notion of “inheritance.” Classes can be organized into hierarchies, where a class might have one or more parent or child classes. If a class has a parent class, we say it is derived or inherited from the parent class and it represents an “IS-A” type relationship. That is to say, the child class “IS-A” type of the parent class.

Therefore, if a class inherits from another class, it automatically obtains a lot of the same functionality and properties from that class and can be extended to contain separate code and data. A nice feature of inheritance is that it often leads to good code reuse since a parent class’ functions don’t need to be re-defined in any of its child classes.

Consider two classes: one being the superclass—or parent—and the other being the subclass—or child. The child class will inherit the properties of the parent class, possibly modifying or extending its behavior. Programmers applying the technique of inheritance arrange these classes into what is called an “IS-A” type of relationship.

*Example: For instance, in the animal world, an insect could be represented by an Insect superclass. All insects share similar properties, such as having six legs and an exoskeleton. Subclasses might be defined for grasshoppers and ants. Because they inherit or are derived from the Insect class, they automatically share all insect properties.*

4. Polymorphism

In OOP, polymorphism allows for the uniform treatment of classes in a hierarchy. Therefore, calling code only needs to be written to handle objects from the root of the hierarchy, and any object instantiated by any child class in the hierarchy will be handled in the same way.

Because derived objects share the same interface as their parents, the calling code can call any function in that class’ interface. At run-time, the appropriate function will be called depending on the type of object passed leading to possibly different behaviors.

*Example: Suppose we have a class called, “Animal” and two child classes, “Cat,” and “Dog.” If the Animal class has a method to make a noise, called, “makeNoise,” then, we can override the "makeNoise" function that is inherited by the sub-classes, "Cat" and "Dog," to be “meow” and “bark,” respectively. Another function can, then, be written that accepts any Animal object as a parameter and invokes its "makeNoise" member function. The noise will be different: either a “meow” or a “bark” depending on the type of animal object that was actually passed to the function.*

Advantages of OOP

Troubleshooting is easier with the OOP language

Suppose the user has no idea where the bug lies if there is an error within the code. Also, the user has no idea where to look into the code to fix the error. This is quite difficult for standard programming languages. However, when Object-Oriented Programming is applied, the user knows exactly where to look into the code whenever there is an error. There is no need to check other code sections as the error will show where the trouble lies.

It is mainly through encapsulation that makes the objects self-contained. This further helps in troubleshooting and easier collaborative development.

2. Code Reusability

One of two important concepts that are provided by Object-Oriented Programming is the concept of inheritance. Through inheritance, the same attributes of a class are not required to be written repeatedly. This avoids the issues where the same code has still to be written multiple times in a code. With the introduction of the concept of classes, the code section can be used as many times as required in the program. Through the inheritance approach, a child class is created that inherits the fields and methods of the parent class. The methods and values that are present in the parent class can be easily overridden. Through inheritance, the features of one class can be inherited by another class by extending the class. Therefore, inheritance is vital for providing code reusability and also multilevel inheritance.

Thus, Object-Oriented Programming offers the feature of class reusability where the class that is once created can be used again. In doing so, time is saved, and the need for extra coding is eliminated as similar features can be inherited.

3. Productivity

The productivity of two codes increases through the use of Object-Oriented Programming. This is because the OOP has provided so many libraries that new programs have become more accessible. Also, as it provides the facility of code reusability, the length of a code is decreased, further enhancing the faster development of newer codes and programs.

4. Data Redundancy

By the term data redundancy, it means that the data is repeated twice. This means that the same data is present more than one time. In Object-Oriented Deprogramming the data redundancy is considered to be an advantage. For example, the user wants to have a functionality that is similar to almost all the classes. In such cases, the user can create classes with similar functionaries and inherit them wherever required.

5. Code Flexibility

The flexibility is offered through the concept of Polymorphism. A scenario can be considered for a better understanding of the concept. A person can behave differently whenever the surroundings change. For example, if the person is in a market, the person will behave like a customer, or the behavior might get changed to a student when the person is in a school or any institution.

In this example, it can be observed that different behaviors are shown by the same person whenever the surroundings around the person get changed. This could explain the concept of Polymorphism and its flexibility. The developers benefit through Polymorphism in the following ways: simplicity and extensibility.

6. Solving problems

Problems can be efficiently solved by breaking down the problem into smaller pieces. If a complex problem is broken down into smaller pieces or components, it becomes a good programming practice. Considering this fact, OOPS utilizes this feature where it breaks down the code of the software into smaller pieces of the object into bite-size pieces that are created one at a time. Once the problem is broken down, these broken pieces can be used again to solve other problems. Also, the more minor codes can get replaced through the modules with the same interface having the implementation details.

7. Security

Because of the concept of data abstraction in OOPS, only a limited amount of data is shown to the user. The rest data is not exposed while exposing only the required amount of data. Therefore, it allows the maintenance of security. The concept of abstraction is used to hide the complexity from other users and demonstrate the element’s information as per the requirements. It also helps in avoiding repetitive code. Another concept provided in OOPS is the feature of encapsulation that allows the protection of the data in the classes from getting accessed by the system. All the internal contents in the class can be safeguarded. In Java, encapsulation is mainly used for restricting access to the class fields directly while setting all the fields of the class to private.

The code in the OOPS is an easy maintenance coding due to the presence of a coding base in a central way. Therefore it is easy to create procedure code that can be easily maintained.

A lot of benefits are further associated with the use of Object-Oriented Programming. Therefore the benefits of Object-Oriented Programming are:

* Users can build programs from the standard modules that communicate with each other. This prevents writing the code again from scratch, which ultimately saves the development time of the code and therefore is higher productivity of the program.
* Problems can be solved easily through OOPS because a program can be broken down into bit-sized codes or problems that can be easily solved.
* With the development of technology, the maintenance cost of the programs is reduced, and there is increased productivity.
* The upgrade of the OOP system of languages can be easily done from smaller systems to larger systems.
* Without interference, there might be multiple instances of the same object.

Java, Python, C++, Lisp, and Perl are all examples of popular object-oriented programming languages. They support programming using the classes and objects paradigm.

Five of the most popular object-oriented languages include:

* Java
* Python
* C++
* Ruby
* C#

1. Java – Java is everywhere, and it is one of the most used and in-demand languages of all time. Java’s motto is ‘write once, run anywhere,’ and that is reflected in the number of platforms it runs on and places it’s used.
2. Python – Python is general-purpose and used in many places. However, Python has a strong foothold in machine learning and data science. It’s one of the preferred languages for that new and ever-growing field.
3. C++ – C++has the speed of C with the functionality of classes and an object-oriented paradigm. It’s a compiled, reliable, and powerful language. In fact, it’s even used to build compilers and interpreters for other languages.
4. Ruby – Ruby is another general-purpose programming language. It was built for simplicity. With that said, Ruby is an incredibly powerful language. The creator of Ruby, Yukihiro “Matz” Matsumoto, has said, “Ruby is very simple in appearance, but is very complex inside, just like our human body.”
5. C# – C# is a programming language designed by Microsoft. It was designed to improve upon existing concepts in C. C# powers the Microsoft .NET framework alongside many web apps, games, desktop apps, and mobile apps.

There are other object-oriented languages that we haven’t covered above. Perl, Objective-C, Dart, Lisp, JavaScript, and PHP are all object-oriented too or support object-oriented principles.

1. Client-Server Systems

Object-oriented client-server systems provide the IT infrastructure, creating Object-Oriented Client-Server Internet (OCSI) applications. Here, infrastructure refers to operating systems, networks, and hardware. OSCI consist of three major technologies:

The Client Server

Object-Oriented Programming

The Internet

2. Object-Oriented Databases

They are also called Object Database Management Systems (ODBMS). These databases store objects instead of data, such as real numbers and integers. Objects consist of the following:

Attributes: Attributes are data that define the traits of an object. This data can be as simple as integers and real numbers. It can also be a reference to a complex object.

3. Object-Oriented Databases

These databases try to maintain a direct correspondence between the real-world and database objects in order to let the object retain its identity and integrity. They can then be identified and operated upon.

4. Real-Time System Design

Real-time systems inherent complexities that make it difficult to build them. Object-oriented techniques make it easier to handle those complexities. These techniques present ways of dealing with these complexities by providing an integrated framework, which includes schedulability analysis and behavioral specifications.

5. Simulation and Modeling System

It’s difficult to model complex systems due to the varying specification of variables. These are prevalent in medicine and in other areas of natural science, such as ecology, zoology, and agronomic systems. Simulating complex systems requires modeling and understanding interactions explicitly. Object-oriented programming provides an alternative approach for simplifying these complex modeling systems.

6. Hypertext and Hypermedia

OOP also helps in laying out a framework for hypertext. Basically, hypertext is similar to regular text, as it can be stored, searched, and edited easily. The only difference is that hypertext is text with pointers to other text as well.

Hypermedia, on the other hand, is a superset of hypertext. Documents having hypermedia not only contain links to other pieces of text and information but also to numerous other forms of media, ranging from images to sound.

7. Neural Networking and Parallel Programming

It addresses the problem of prediction and approximation of complex time-varying systems. Firstly, the entire time-varying process is split into several time intervals or slots. Then, neural networks are developed in a particular time interval to disperse the load of various networks. OOP simplifies the entire process by simplifying the approximation and prediction ability of networks.

8. Office Automation Systems

These include formal as well as informal electronic systems primarily concerned with information sharing and communication to and from people inside and outside the organization. Some examples are:

Email

Word processing

Web calendars

Desktop publishing

9. CIM/CAD/CAM Systems

OOP can also be used in manufacturing and design applications, as it allows people to reduce the effort involved. For instance, it can be used while designing blueprints and flowcharts. OOP makes it possible for the designers and engineers to produce these flowcharts and blueprints accurately.

10. AI Expert Systems

These are computer applications that are developed to solve complex problems pertaining to a specific domain, which is at a level far beyond the reach of a human brain.

It has the following characteristics:

Reliable

Highly responsive

Understandable

High-performance

Keywords: Keywords are pre-defined or reserved words in a programming language. Each keyword is meant to perform a specific function in a program. Since keywords are referred names for a compiler, they can’t be used as variable names because by doing so, we are trying to assign a new meaning to the keyword which is not allowed. You cannot redefine keywords. However, you can specify the text to be substituted for keywords before compilation by using C/C++ preprocessor directives.

asm bool catch class

const\_cast delete dynamic\_cast explicit

export false friend inline

mutable namespace new operator

private protected public reinterpret\_cast

static\_cast template this throw

true try typeid typename

using virtual wchar\_t



Identifiers: Identifiers are used as the general terminology for the naming of variables, functions and arrays. These are user-defined names consisting of an arbitrarily long sequence of letters and digits with either a letter or the underscore(\_) as a first character. Identifier names must differ in spelling and case from any keywords. You cannot use keywords as identifiers; they are reserved for special use. Once declared, you can use the identifier in later program statements to refer to the associated value. A special kind of identifier, called a statement label, can be used in goto statements.

There are certain rules that should be followed while naming c identifiers:

* They must begin with a letter or underscore(\_).
* They must consist of only letters, digits, or underscore. No other special character is allowed.
* It should not be a keyword.
* It must not contain white space.
* It should be up to 31 characters long as only the first 31 characters are significant.
* main: method name.
* a: variable name.

3.Constants: Constants are also like normal variables. But, the only difference is, their values can not be modified by the program once they are defined. Constants refer to fixed values. They are also called literals.

Constants may belong to any of the data type

Syntax:

const data\_type variable\_name; (or) const data\_type \*variable\_name;

Types of Constants:

1. Integer constants – Example: 0, 1, 1218, 12482
2. Real or Floating-point constants – Example: 0.0, 1203.03, 30486.184
3. Octal & Hexadecimal constants – Example: octal: (013 )8 = (11)10, Hexadecimal: (013)16 = (19)10
4. Character constants -Example: ‘a’, ‘A’, ‘z’
5. String constants -Example: “GeeksforGeeks”

Strings: Strings are nothing but an array of characters ended with a null character (‘\0’). This null character indicates the end of the string. Strings are always enclosed in double-quotes. Whereas, a character is enclosed in single quotes in C and C++.Declarations for String:

char string[20] = {‘g’, ’e’, ‘e’, ‘k’, ‘s’, ‘f’, ‘o’, ‘r’, ‘g’, ’e’, ‘e’, ‘k’, ‘s’, ‘\0’};

char string[20] = “geeksforgeeks”;

char string [] = “geeksforgeeks”;

when we declare char as “string[20]”, 20 bytes of memory space is allocated for holding the string value.

When we declare char as “string[]”, memory space will be allocated as per the requirement during the execution of the program.

Machine Independent

A C++ executable is not platform-independent (compiled programs on Linux won’t run on Windows), however they are machine independent.

Let us understand this feature of C++ with the help of an example. Suppose you have written a piece of code which can run on Linux/Windows/Mac OSx which makes C++ Machine Independent but the executable file of the C++ cannot run on different operating systems.

Simple

It is a simple language in the sense that programs can be broken down into logical units and parts, has rich library support, and a variety of data-types. Also, the Auto Keyword of the C++ makes life easier.

The auto keyword

The idea of the auto was to form the C++ compiler deduce the data type while compiling instead of making you declare the data type every-freaking-time. Do keep in mind that you cannot declare something without an initializer. There must be some way for the compiler to deduce your type.

High-Level Language

C++ is a High-Level Language, unlike C which is a Mid-Level Programming Language. It makes life easier to work in C++ as it is a high-level language as it is closely associated with the human-comprehensible English language.

Popular

C++ can be the base language for many other programming languages that supports the feature of object-oriented programming. Bjarne Stroustrup found Simula 67, the first object-oriented language ever, lacking simulations and decided to develop C++.

Case-sensitive

It is clear that the C++ is a case-sensitive programming language. For example, cin is used to take input from the input stream. But if the “Cin” won’t work. Other languages like HTML and MySQL are not case-sensitive language.

Compiler Based

C++ is a compiler-based language, unlike Python. That is C++ programs used to be compiled and their executable file is used to run it. Due to which C++ is a relatively faster language than Java and Python.

Dynamic Memory Allocation

When the program executes in the C++ then the variables are allocated the dynamical heap space. Inside of the functions the variables are allocated in the stack space. Many times, We are not aware in advance that how much memory is needed to store particular information in a defined variable and the size of required memory can be determined at run time.

Memory Management

C++ allows us to allocate the memory of a variable or an array in run time. This is known as Dynamic Memory Allocation.

In other programming languages such as Java and Python, the compiler automatically manages the memories allocated to variables. But this is not the case in C++.

In C++, the memory must be de-allocate dynamically allocated memory manually after it is of no use.

The allocation and deallocation of the memory can be done using the new and delete operators respectively.

Multi-threading

Multithreading is a specialized form of multitasking and multitasking is a feature that allows your system to execute two or more programs concurrently. In general, there are two sorts of multitasking: process-based and thread-based.

Process-based multitasking handles the concurrent execution of programs. Thread-based multitasking deals with the multiprogramming of pieces of an equivalent program.

A multithreaded program contains two or more parts that will run concurrently. Each a part of such a program is named a thread, and every thread defines a separate path of execution.

C++ doesn’t contain any built-in support for multithreaded applications. Instead, it relies entirely upon the OS to supply this feature.



C++ int

* The int keyword is used to indicate integers.
* Its size is usually 4 bytes. Meaning, it can store values from -2147483648 to 2147483647.
* For example,
* int salary = 85000;

2. C++ float and double

* float and double are used to store floating-point numbers (decimals and exponentials).
* The size of float is 4 bytes and the size of double is 8 bytes. Hence, double has two times the precision of float. To learn more, visit C++ float and double.
* For example,
* float area = 64.74;
* double volume = 134.64534;
* As mentioned above, these two data types are also used for exponentials.
* For example, double distance = 45E12 // 45E12 is equal to 45\*10^12

3. C++ char

* Keyword char is used for characters.
* Its size is 1 byte.
* Characters in C++ are enclosed inside single quotes ' '.
* For example,
* char test = 'h';
* Note: In C++, an integer value is stored in a char variable rather than the character itself. To learn more, visit C++ characters.

4. C++ wchar\_t

* Wide character wchar\_t is similar to the char data type, except its size is 2 bytes instead of 1.
* It is used to represent characters that require more memory to represent them than a single char.
* For example,
* wchar\_t test = L'ם' // storing Hebrew character;
* Notice the letter L before the quotation marks.
* Note: There are also two other fixed-size character types char16\_t and char32\_t introduced in C++11.

5. C++ bool

* The bool data type has one of two possible values: true or false.
* Booleans are used in conditional statements and loops (which we will learn in later chapters).
* For example,
* bool cond = false;

6. C++ void

* The void keyword indicates an absence of data. It means "nothing" or "no value".
* We will use void when we learn about functions and pointers.
* Note: We cannot declare variables of the void type.

C++ Type Modifiers

We can further modify some of the fundamental data types by using type modifiers. There are 4 type modifiers in C++. They are:

1. signed
2. unsigned
3. short
4. long

We can modify the following data types with the above modifiers:

1. int
2. double
3. char

The general rules for naming variables are:

* Names can contain letters, digits and underscores
* Names must begin with a letter or an underscore (\_)
* Names are case sensitive (myVar and myvar are different variables)
* Names cannot contain whitespaces or special characters like !, #, %, etc.
* Reserved words (like C++ keywords, such as int) cannot be used as names

Data values that stay the same every time a program is executed are known as constants. Constants are not expected to change.

Literal constants are actual values fixed into the source code. An example of this might be the character string "hello world". The data value "hello world" has been fixed into the code.

Named constants are values where a name is defined to be used instead of a literal constant. An example of this might be stating that the 'starting level' of a game is always referred to as 1.

Examples of a constant within a game might be:

* the unit of gravity
* the number of lives available for the player
* the amount of time allowed for a level in a game

A Cast operator is an unary operator which forces one data type to be converted into another data type.

C++ supports four types of casting:

* Static Cast
* Dynamic Cast
* Const Cast
* Reinterpret Cast

Static Cast: This is the simplest type of cast which can be used. It is a compile time cast.It does things like implicit conversions between types (such as int to float, or pointer to void\*), and it can also call explicit conversion functions (or implicit ones).

For e.g.

#include <iostream>

using namespace std;

int main()

{

float f = 3.5;

int a = f; // this is how you do in C

int b = static\_cast<int>(f);

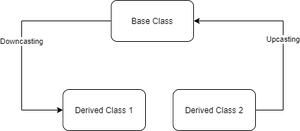
cout << b;

}

Dynamic Cast: A cast is an operator that converts data from one type to another type. In C++, dynamic casting is mainly used for safe downcasting at run time. To work on dynamic\_cast there must be one virtual function in the base class. A dynamic\_cast works only polymorphic base class because it uses this information to decide safe downcasting.

Syntax:

dynamic\_cast <new\_type>(Expression)



Downcasting: Casting a base class pointer (or reference) to a derived class pointer (or reference) is known as downcasting. In figure 1 casting from the Base class pointer/reference to the “derived class 1” pointer/reference showing downcasting (Base ->Derived class).

Upcasting: Casting a derived class pointer (or reference) to a base class pointer (or reference) is known as upcasting. In figure 1 Casting from Derived class 2 pointer/reference to the “Base class” pointer/reference showing Upcasting (Derived class 2 -> Base Class).

const\_cast

const\_cast is used to cast away the constness of variables. Following are some interesting facts about const\_cast.

1. const\_cast can be used to change non-const class members inside a const member function. Consider the following code snippet. Inside const member function fun(), ‘this’ is treated by the compiler as ‘const student\* const this’, i.e. ‘this’ is a constant pointer to a constant object, thus compiler doesn’t allow to change the data members through ‘this’ pointer. const\_cast changes the type of ‘this’ pointer to ‘student\* const this’.
2. const\_cast can be used to pass const data to a function that doesn’t receive const. For example, in the following program fun() receives a normal pointer, but a pointer to a const can be passed with the help of const\_cast.
3. It is undefined behavior to modify a value which is initially declared as const. Consider the following program. The output of the program is undefined. The variable ‘val’ is a const variable and the call ‘fun(ptr1)’ tries to modify ‘val’ using const\_cast.
4. const\_cast is considered safer than simple type casting. It’safer in the sense that the casting won’t happen if the type of cast is not same as original object. For example, the following program fails in compilation because ‘int \*’ is being typecasted to ‘char \*’

reinterpret\_cast is a type of casting operator used in C++.

It is used to convert a pointer of some data type into a pointer of another data type, even if the the data types before and after conversion are different.

It does not check if the pointer type and data pointed by the pointer is same or not.

data\_type \*var\_name =

reinterpret\_cast <data\_type \*>(pointer\_variable);

Return Type

It doesn’t have any return type. It simply converts the pointer type.

Parameters

It takes only one parameter i.e., the source pointer variable (p in above example).

Purpose for using reinterpret\_cast

reinterpret\_cast is a very special and dangerous type of casting operator. And is suggested to use it using proper data type i.e., (pointer data type should be same as original data type).

* It can typecast any pointer to any other data type.
* It is used when we want to work with bits.
* If we use this type of cast then it becomes a non-portable product. So, it is suggested not to use this concept unless required.
* It is only used to typecast any pointer to its original type.
* Boolean value will be converted into integer value i.e., 0 for false and 1 for true.

Explicit C++ type Casting:

The word “explicit” means ‘open’ or ‘clear’. In explicit C++ type casting, the data type in which the value is to be converted is clearly specified in the program. It is done by cast operator. The cast operator is unary operator. It converts the value of an expression into a value of the type specified.

The general form of the C++ type casting operator is as follows;

(type)expression

Where

(type): it indicates one of the C++ data type (or a user-defined data type) to which the value of the expression is to be converted.

Expression: it indicates a constant value, variable or an expression whose data type is to be converted.

For example, to convert a floating-point value 2.322 into an integer value, the statement are written as;

int x;

x=(int)2.322;

cout<<x;

After executing the above statements, the floating-point value will be converted into integer. The value assigned to variable ‘x’. the decimal portion will be truncated( or removed. The value 2 will be displayed on the screen.

Example: write a program that explains the Explicit C++ type casting:

#include <iostream>

using namespace std;

int main()

{

int a, b;

a=15;

b=2;

cout<<a/b<<endl;

cout<<a/float(b)<<endl;

}

Implicit C++ Type Casting:

The word “implicit” means ‘understood’ or ‘embedded’. In implicit C++ type casting, the data type in which the value is to be converted is not specified in the program. It is automatically done by the C++ compiler.

When constant values and variables of different types are mixed in an expression, they are converted into the same type. The compiler will convert all operands of lower order data type into higher order data type.



In the above table, ‘long double’ data type has the highest order, while ‘char’ data type has the lowest order. For example, if an expression has highest order operand of data type ‘int’, then all ‘char’ and ‘short int’ will be converted into ‘int’.

Similarly, in the assignment statement when a value of one data type is assigned to a variable of another data type, the type of the value being assigned is converted into the data type of variable to the left side of assignment operator. For example;

int x;

float y= 49.723;

x=y;

The data type of the variable ‘x’ is ‘int’. when the value of ‘y’ is assigned to ‘x’, it is converted into integer type and value 49 will be assigned to ‘x’. the decimal portion of ‘y’ will be truncated.

Example: write a program that separated the integral and fractional parts of a given real number and displays the result on screen using implicit C++ type casting:

#include <iostream>

using namespace std;

int main()

{

float a= 15.58971, b;

int n;

b=a-n;

cout<<" Real number is :"<<a<<endl;

cout<<" Integral part = "<<n<<endl;

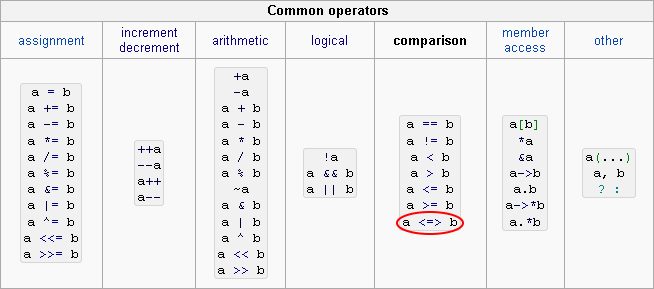
cout<<" Fractional part = "<<b<<endl;

}

Operators are symbols that perform operations on variables and values. For example, + is an operator used for addition, while - is an operator used for subtraction.

Operators in C++ can be classified into 6 types:

* Arithmetic Operators
* Assignment Operators
* Relational Operators
* Logical Operators
* Bitwise Operators
* Other Operators



The precedence of an operator specifies how "tightly" it binds two expressions together. For example, in the expression 1 + 5 \* 3, the answer is 16 and not 18 because the multiplication ("\*") operator has a higher precedence than the addition ("+") operator. Parentheses may be used to force precedence, if necessary. For instance: (1 + 5) \* 3 evaluates to 18.

When operators have equal precedence their associativity decides how the operators are grouped. For example "-" is left-associative, so 1 - 2 - 3 is grouped as (1 - 2) - 3 and evaluates to -4. "=" on the other hand is right-associative, so $a = $b = $c is grouped as $a = ($b = $c).

## C++ if Statement

The syntax of the if statement is:

if (condition) {

// body of if statement

}

The if statement evaluates the condition inside the parentheses ( ).

* If the condition evaluates to true, the code inside the body of if is executed.
* If the condition evaluates to false, the code inside the body of if is skipped.

## C++ if...else

The if statement can have an optional else clause. Its syntax is:

if (condition) {

// block of code if condition is true

}

else {

// block of code if condition is false

}

The if..else statement evaluates the condition inside the parenthesis.

If the condition evaluates true,

* the code inside the body of if is executed
* the code inside the body of else is skipped from execution

If the condition evaluates false,

* the code inside the body of else is executed
* the code inside the body of if is skipped from execution

## C++ if...else...else if statement

The if...else statement is used to execute a block of code among two alternatives. However, if we need to make a choice between more than two alternatives, we use the if...else if...else statement.

The syntax of the if...else if...else statement is:

if (condition1) {

// code block 1

}

else if (condition2){

// code block 2

}

else {

// code block 3

}

Here,

* If condition1 evaluates to true, the code block 1 is executed.
* If condition1 evaluates to false, then condition2 is evaluated.
* If condition2 is true, the code block 2 is executed.
* If condition2 is false, the code block 3 is executed.

## C++ for loop

The syntax of for-loop is:

for (initialization; condition; update) {

// body of-loop

}

Here,

* initialization - initializes variables and is executed only once
* condition - if true, the body of for loop is executed  
  if false, the for loop is terminated
* update - updates the value of initialized variables and again checks the condition

## C++ while Loop

The syntax of the while loop is:

while (condition) {

// body of the loop

}

Here,

* A while loop evaluates the condition
* If the condition evaluates to true, the code inside the while loop is executed.
* The condition is evaluated again.
* This process continues until the condition is false.
* When the condition evaluates to false, the loop terminates.

## C++ do...while Loop

The do...while loop is a variant of the while loop with one important difference: the body of do...while loop is executed once before the condition is checked.

Its syntax is:

do {

// body of loop;

}

while (condition);

Here,

* The body of the loop is executed at first. Then the condition is evaluated.
* If the condition evaluates to true, the body of the loop inside the do statement is executed again.
* The condition is evaluated once again.
* If the condition evaluates to true, the body of the loop inside the do statement is executed again.
* This process continues until the condition evaluates to false. Then the loop stops.

## Types of Arrays in C++

There are 3 types of an array in C++ :

* One-dimensional array
* Two-dimensional array
* Multidimensional array

### One-Dimensional Array:

In this type of array, it stores elements in a single dimension. And, In this array, a single specification is required to describe elements of the array.

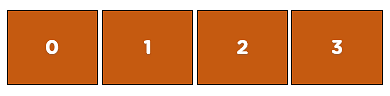


Fig: One-dimensional array

The diagram above shows that it arranged all the elements row-wise in a single dimension, one after the other.

Now, take an example of a one-dimensional array.

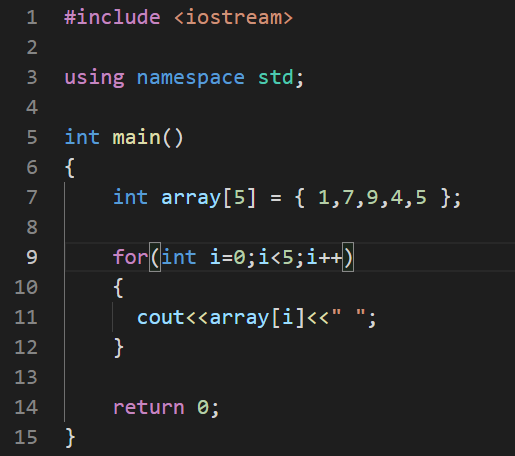
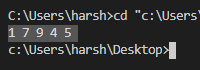


Fig: Example of a one-dimensional array

In this example, you are printing the elements 1,7,9,4,5 using for loop, and depicted below is the output of this example.



### Two-Dimensional Array:

In this type of array, two indexes describe each element, the first index represents a row, and the second index represents a column.

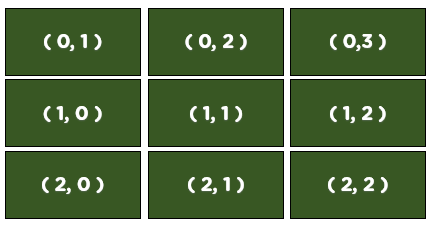


                                Fig: Two-dimensional array

As you can see, the elements are arranged row-wise and column-wise; in a two-dimensional array, there are i number of rows and j number of columns. The above figure is a representation of a 3 x 3 matrix, which means there are three rows and three columns in the array.

Now let’s look at an example of a two-dimensional array.

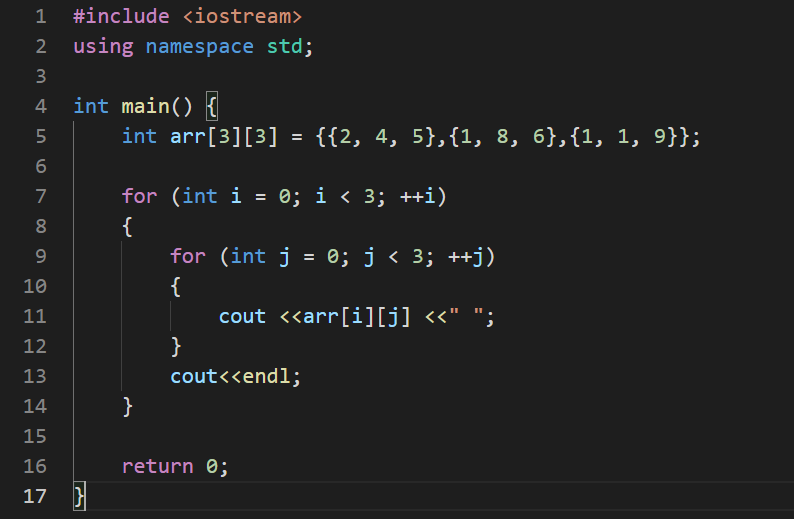


                                   Fig: Example of a two-dimensional array

In this example, you are printing a two-dimensional array of three rows and three columns; you need to use two for [loops.](https://www.simplilearn.com/tutorials/cpp-tutorial/cpp-for-loop) The first loop, i.e., i loop, runs for the row from 0 to 3, and the second loop, i.e., j loop, runs for the column from 0 to 3.

And below is the output of this example.

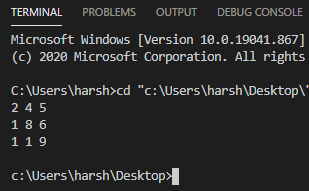


Fig: Output

### Multidimensional Array:

The simplest example of a multidimensional array is a 2-d array; a two-dimensional array also falls under the category of a multidimensional array. This array can have any number of dimensions.

The syntax for declaring a multidimensional array is:

Syntax:

                Datatype array\_name [size 1][size 2] . . . . . [size n];

Here size1 size2 up to so on size n describes the number of dimensions; in the case of a 2-d array, there are only two dimensions, a multidimensional array can have any number of dimensions.

Example:

                     int array[5][10][4];

#### ****String Manipulation With C++ String Class****

Let’s now see how to do the same thing using C++ string class

* **Concatenation** – Combines two string into one. Simply use the ***+ operator***. See example below
* **String length** – Use***length()*** method
* **Searching strings** – Accessing a character within a string. Use the **find()** method
* **Substrings** – Returning part of a string. Use the **substr()** method.
* **Replacing** – Replacing part of a string. Use the**replace()** method
* **Insertion** – Inserting character(s) into a string. Using the***insert()*** method
* **Erase** – Removing part of a string. Use the**erase()** method.

Unified Modeling Language (UML) is a general purpose modelling language. The main aim of UML is to define a standard way to visualize the way a system has been designed. It is quite similar to blueprints used in other fields of engineering.

UML is not a programming language, it is rather a visual language. We use UML diagrams to portray the behavior and structure of a system. UML helps software engineers, businessmen and system architects with modelling, design and analysis. The Object Management Group (OMG) adopted Unified Modelling Language as a standard in 1997. Its been managed by OMG ever since. International Organization for Standardization (ISO) published UML as an approved standard in 2005. UML has been revised over the years and is reviewed periodically.

Do we really need UML?

Complex applications need collaboration and planning from multiple teams and hence require a clear and concise way to communicate amongst them.

Businessmen do not understand code. So UML becomes essential to communicate with non programmers essential requirements, functionalities and processes of the system.

A lot of time is saved down the line when teams are able to visualize processes, user interactions and static structure of the system.

UML is linked with object oriented design and analysis. UML makes the use of elements and forms associations between them to form diagrams. Diagrams in UML can be broadly classified as:

Structural Diagrams – Capture static aspects or structure of a system. Structural Diagrams include: Component Diagrams, Object Diagrams, Class Diagrams and Deployment Diagrams.

Behavior Diagrams – Capture dynamic aspects or behavior of the system. Behavior diagrams include: Use Case Diagrams, State Diagrams, Activity Diagrams and Interaction Diagrams.

Structural UML Diagrams –

1. Class Diagram – The most widely use UML diagram is the class diagram. It is the building block of all object oriented software systems. We use class diagrams to depict the static structure of a system by showing system’s classes,their methods and attributes. Class diagrams also help us identify relationship between different classes or objects.
2. Composite Structure Diagram – We use composite structure diagrams to represent the internal structure of a class and its interaction points with other parts of the system. A composite structure diagram represents relationship between parts and their configuration which determine how the classifier (class, a component, or a deployment node) behaves. They represent internal structure of a structured classifier making the use of parts, ports, and connectors. We can also model collaborations using composite structure diagrams. They are similar to class diagrams except they represent individual parts in detail as compared to the entire class.
3. Object Diagram – An Object Diagram can be referred to as a screenshot of the instances in a system and the relationship that exists between them. Since object diagrams depict behaviour when objects have been instantiated, we are able to study the behaviour of the system at a particular instant. An object diagram is similar to a class diagram except it shows the instances of classes in the system. We depict actual classifiers and their relationships making the use of class diagrams. On the other hand, an Object Diagram represents specific instances of classes and relationships between them at a point of time.
4. Component Diagram – Component diagrams are used to represent the how the physical components in a system have been organized. We use them for modelling implementation details. Component Diagrams depict the structural relationship between software system elements and help us in understanding if functional requirements have been covered by planned development. Component Diagrams become essential to use when we design and build complex systems. Interfaces are used by components of the system to communicate with each other.
5. Deployment Diagram – Deployment Diagrams are used to represent system hardware and its software.It tells us what hardware components exist and what software components run on them.We illustrate system architecture as distribution of software artifacts over distributed targets. An artifact is the information that is generated by system software. They are primarily used when a software is being used, distributed or deployed over multiple machines with different configurations.
6. Package Diagram – We use Package Diagrams to depict how packages and their elements have been organized. A package diagram simply shows us the dependencies between different packages and internal composition of packages. Packages help us to organise UML diagrams into meaningful groups and make the diagram easy to understand. They are primarily used to organise class and use case diagrams.

Behavior Diagrams –

1. State Machine Diagrams – A state diagram is used to represent the condition of the system or part of the system at finite instances of time. It’s a behavioral diagram and it represents the behavior using finite state transitions. State diagrams are also referred to as State machines and State-chart Diagrams . These terms are often used interchangeably.So simply, a state diagram is used to model the dynamic behavior of a class in response to time and changing external stimuli.
2. Activity Diagrams – We use Activity Diagrams to illustrate the flow of control in a system. We can also use an activity diagram to refer to the steps involved in the execution of a use case. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram.An activity diagram focuses on condition of flow and the sequence in which it happens. We describe or depict what causes a particular event using an activity diagram.
3. Use Case Diagrams – Use Case Diagrams are used to depict the functionality of a system or a part of a system. They are widely used to illustrate the functional requirements of the system and its interaction with external agents(actors). A use case is basically a diagram representing different scenarios where the system can be used. A use case diagram gives us a high level view of what the system or a part of the system does without going into implementation details.
4. Sequence Diagram – A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place.We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.
5. Communication Diagram – A Communication Diagram(known as Collaboration Diagram in UML 1.x) is used to show sequenced messages exchanged between objects. A communication diagram focuses primarily on objects and their relationships. We can represent similar information using Sequence diagrams,however, communication diagrams represent objects and links in a free form.
6. Timing Diagram – Timing Diagram are a special form of Sequence diagrams which are used to depict the behavior of objects over a time frame. We use them to show time and duration constraints which govern changes in states and behavior of objects.
7. Interaction Overview Diagram – An Interaction Overview Diagram models a sequence of actions and helps us simplify complex interactions into simpler occurrences. It is a mixture of activity and sequence diagrams.

Class: A class in C++ is the building block that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object.

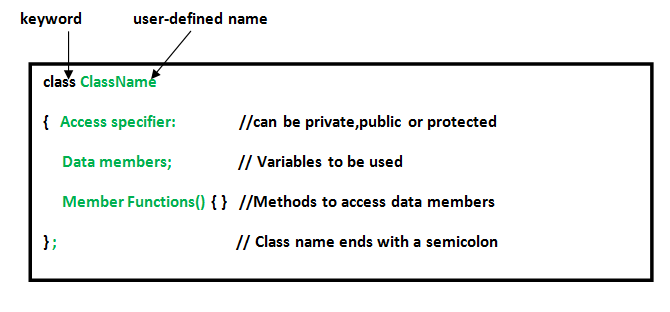
For Example: Consider the Class of Cars. There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

A Class is a user defined data-type which has data members and member functions.

Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.

In the above example of class Car, the data member will be speed limit, mileage etc and member functions can be apply brakes, increase speed etc.

A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.



An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

Declaring Objects: When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

Syntax:

ClassName ObjectName;

Accessing data members and member functions: The data members and member functions of class can be accessed using the dot(‘.’) operator with the object. For example if the name of object is obj and you want to access the member function with the name printName() then you will have to write obj.printName() .

Accessing Data Members

The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object. Accessing a data member depends solely on the access control of that data member.

This access control is given by Access modifiers in C++. There are three access modifiers : public, private and protected.