



# C++

Session #1

# #1

- OOP
- History of C++
- C vs C++





# OOP (Object-Oriented programming)

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



# OOP (Object-Oriented programming)

## 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.



# OOP (Object-Oriented programming)

## Class

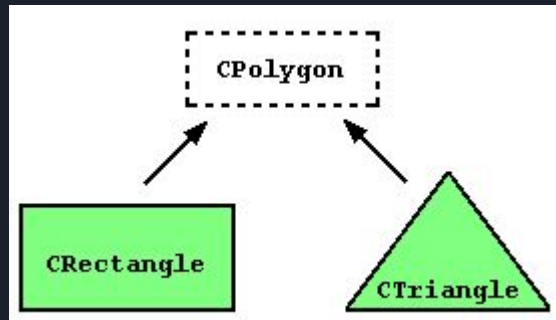
It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

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.

# OOP (Object-Oriented programming)

## Inheritance

Classes can be extended, creating new classes which retain characteristics of the base class. This process, known as inheritance, involves a base class and a derived class: The derived class inherits the members of the base class, on top of which it can add its own members.





# OOP (Object-Oriented programming)

## Polymorphism

The word polymorphism means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

Polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.



# OOP (Object-Oriented programming)

## Abstraction

Data abstraction refers to providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.

Data abstraction is a programming (and design) technique that relies on the separation of interface and implementation.

*Example: You don't fully know how your computer works. There are I/O (keyboard, screen, speakers) you can interact with and if you can add/remove RAM, your user experience will stay the same.*





# OOP (Object-Oriented programming)

## Encapsulation

Encapsulation is an Object Oriented Programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse. Data encapsulation led to the important OOP concept of data hiding.

Data encapsulation is a mechanism of bundling the data, and the functions that use them and data abstraction is a mechanism of exposing only the interfaces and hiding the implementation details from the user.



# OOP (Object-Oriented programming)

## Who is using OOP?

Well... Lot of languages:

- C++
- Java
- Python
- PHP
- JavaScript
- etc...



# History of C++

C++ is a programming language developed at AT&T Bell Laboratories by Bjarne Stroustrup in the early 1980's. The language was designed with the intent of merging the efficiency and conciseness of C with the object-oriented programming features of SIMULA-67. Since its creation, the language has evolved rapidly and several new features have been added since its initial release in 1985. The language also promises to provide support for several other useful mechanisms such as parameterized types and exception handling in the near future.



# History of C++

A formal ANSI-C++ committee (X3J16) has since been established to help develop an accurate and reliable standard for the language which should eliminate most, if not all, ambiguities in the C++ compilers and translators of today. It is expected that this committee will adopt most of the rules present in the ANSI base document *The Annotated C++ Reference Manual* as written by Ellis and Stroustrup.



# History of C++

With a few modest exceptions, C++ can be considered a superset of the C programming language. While C++ is similar to C in syntax and structure, it is important to realize that the two languages are radically different. Comparing C to C++ is like comparing checkers to chess. Although both games are played on the same board, one must realize that the game of chess cannot be played using the same strategy as that used in the game of checkers. C++ and its support for object-oriented programming provide a new methodology for designing, implementing and ease of maintaining software projects which C, a structured programming language, is unable to support.



# History of C++

Extensive libraries are available for the C programming language; consequently, a deliberate effort was made on behalf of the developers of C++ to maintain backward compatibility with C. Any major deviation from the C programming language would have meant that all the libraries available for C would have to be tediously rewritten for C++. This would have severely limited the usefulness of C++ in an environment where C libraries were used extensively.



# History of C++

## Timeline:

- 1979
  - Stroustrup starts to work on “C with Classes” in Bell Labs
- 1983
  - “C with classes” renamed C++
- 1985
  - The first edition of the “The C++ programming language”
- 1989
  - C++ 2.0 version
- 1998
  - The first ISO Standards (C++98)



# History of C++

## Timeline:

- 2003
  - C++ 03, bug fixes of C++98
- 2011
  - C++11, a major revision
- 2014
  - C++14, bug fixes and small improvement on C++
- 2017
  - C++17, latest major version





# History of C++

Throughout C++'s life, its development and evolution has been informally governed by a set of rules that its evolution should follow:

- It must be driven by actual problems and its features should be useful immediately in real world programs.
- Every feature should be implementable (with a reasonably obvious way to do so).
- Programmers should be free to pick their own programming style, and that style should be fully supported by C++.
- Allowing a useful feature is more important than preventing every possible misuse of C++.
- It should provide facilities for organising programs into well-defined separate parts, and provide facilities for combining separately developed parts.



# History of C++

- No implicit violations of the type system (but allow explicit violations; that is, those explicitly requested by the programmer).
- User-created types need to have the same support and performance as built-in types.
- Unused features should not negatively impact created executables (e.g. in lower performance).
- There should be no language beneath C++ (except assembly language).
- C++ should work alongside other existing programming languages, rather than fostering its own separate and incompatible programming environment.
- If the programmer's intent is unknown, allow the programmer to specify it by providing manual control.



# History of C++

```
1 #include <iostream>
2
3 int main() {
4     std::cout << "Hello, world!\n";
5     return 0;
6 }
```



# History of C++

The C++ language has two main components:

- A direct mapping of hardware features provided primarily by the C subset
- zero-overhead abstractions based on those mappings.

Stroustrup describes C++ as "a light-weight abstraction programming language designed for building and using efficient and elegant abstractions; and offering both hardware access and abstraction is the basis of C++. Doing it efficiently is what distinguishes it from other languages.



# History of C++

If you are familiar with Java, PHP or Python, you will find that C++ is a bit harder to use since you need to manually manage memory allocation of your program.



# C vs C++

Comparing the programming languages C and C++ is a bit like comparing a traditional typewriter with an electric typewriter. That's because C++ is a direct descendent of C, the “grandfather” of many modern programming languages, just with more under the hood. C++ boasts better efficiency and productivity; however, with more bells and whistles comes more responsibility.



## C vs C++

C is generally considered to be the foundation of many modern high-level programming languages like C# and Java. C++ language is one of those an enhanced version of the language that adds an object-oriented layer, which definitely boosts developer speed and productivity. C++ is also one of the foundation languages for the MongoDB database and the Apache HTTP server.

Most of the “kernels” available are written in C or C++, because the speed of execution and it allows developer to work closely with the hardware.



# C vs C++

## A BIT ABOUT C

C is a system programming language, whereas C++ is a general-purpose programming language commonly used in embedded systems. C is procedural, so it doesn't support classes and objects like C++ does (although, despite being object-oriented, C++ can be procedural like C, making it a bit more hybrid).

Generally, you would opt to use C over C++ if you didn't want the extra overhead of C++. However you can always just pick the features of C++ you want to use and exclude the others.





# C vs C++

## A BIT ABOUT C++

C++ is everything C is, and more. It's not new, either, and has itself been the inspiration for many languages that have come behind it like Python, Perl, and PHP. It does however add in a few modern elements that make it a step up from C.

For a C++ developer to know the language, they'll also know C—and quite a bit more, which can make it difficult to learn. C++ was created in the 1980s and has been used in the creation of desktop and web applications, although it's most popular for applications such as games, operating systems, and low-level hardware programming for a PC or server.



# C vs C++

To make it simple (1 / 3):

- C++ is directly derived from the C language. This means it shares some properties with C while also adding some improvements.
- C++ is object-oriented. This translates to productivity and organization of code, which is a boon for more complex applications. It's great for fast applications and server-side software.
- C++ is lightweight and compiled. This means that before a C++ application is launched on a PC or the server, the code is converted into a binary file, or an executable .EXE file. C++ compiled files are pretty lightweight vs. files with more overhead, like C#. With C++, you can code for any platform including Mac, Windows and Linux.



# C vs C++

To make it simple (2 / 3):

- It has benefits of both high-level and low-level programming languages. This makes it more of a mid-level language.
- The power of C++ lies in its performance and speed. This makes it ideal for complex, large applications that require a lot of speed at scale. It's super efficient where higher level languages might not be as efficient, making it a better solution for applications where performance is important. We'll get more into some of the features that enable this below, but this is a big win for the language.
- C++ plays well with other languages. Because it can interface with nearly any other language, C++ is a great option. And, almost any system can compile and run C++ code.



# C vs C++

To make it simple (3 / 3):

- Pointers equal productivity. A “pointer” is a feature of C++ (and other C-based languages) that allows developers to simplify code. A pointer represents an “address” where a piece of data exists, so you code the location of a variable, not the whole variable. Think of it this way: Instead of personally handing out newsletters to everyone in your company, you put the newsletters in a mailbox and tell everyone where the mailbox is located. Or, if you’re dealing with a large bit of data, think of a pointer like giving someone your address, rather than giving them your whole house. It’s a logic for computing—one we use every day as humans.



# C vs C++

## When Should You Use Which Language?

If you want an application that works directly with computer hardware or deals with application development, C++ is a good option. C++ programs include server-side applications, networking, gaming, and even device drivers for your PC. However, if you need to code truly tiny systems, using C will result in a little less overhead than C++.

C++ is well-rounded in terms of platforms and target applications, so if your project is focused on extremely low-level processing, then you may want to use C++. One other instance you may consider C over C++ is when you need an application to be incredibly stable, and removing the abstractions of C++ can ensure airtight code and control over every aspect. Also, if you don't have a C++ compiler on your platform, that's another common reason to go with C.

Why Learn C++?





# Why Learn C++?


Pro:

- **Scalability**

C++'s greatest strength is how scalable it could be, so apps that are very resource intensive are usually built with it. Graphics require a lot of resource, which is why the most beautiful 3D games you happily feast your eyes on are often built with C++.

- **Fast**

As a statically typed language, C++ is generally more performant than dynamically typed languages because the code is type-checked before it is executed. Java is gaining ground in terms of speed, but in the end, depending on how talented the C++ developer is, C++ can still be faster than Java.



# Why Learn C++?

Pro:

- **Community**

First of all, community size is important, because the larger a programming language community is, the more support you'd be likely to get. As you step into the programming world, you'll soon understand how vital support is, as the developer community is all about giving and receiving help. Moreover, the larger a community, the more people will be building useful tools to make development in that particular language easier. As of now, there are over 600 notable programming languages world-wide.

So, with that context in mind, let's get into the details of the C++ community.





# Why Learn C++?

Pro:

- **Control**

As mentioned before, since you have a lot of control over how your app uses resources, your app can take up very little resource. All in all, since C++ can be very performant in the right hands, enterprises often use C++ to code functions that have a critical reliance on speed and resource usage.



# Why Learn C++?

## Cons:

- **Very Complex**

Since C++ is rather lower level, the language is huge and you will need to handle a lot of complex things such as memory management and more. You also need to write a lot of code before you can get a working prototype if you're planning on building an app from scratch. Since it will be difficult to grasp how all features in C++ works, you can easily shoot yourself in the foot.

As such, since it's easy for a coding beginner to go astray when learning C++, we strongly recommend learning C++ with a mentor. In addition, C++ has a longer history with game development in general, so there are a lot of proven good practices a C++ mentor from the gaming community can teach you.



# Why Learn C++?

## Cons:

- **Not Easy To Maintain**

C++ needs a lot of code, which means you need a large team to scale a C++ app, and from a time and financial investment point of view, C++ not easy to scale. Also, since you have to do a lot of things manually with C++, it's easy for less experienced or less skilled developers to introduce errors into the code base. Java was developed because so many professional developers were making mistakes, so in terms of talent-recruiting, a very skilled and experienced C++ developer may be hard to find and also expensive to afford, which is why C++ is not so scalable.

# Why Learn C++?

Cons:

