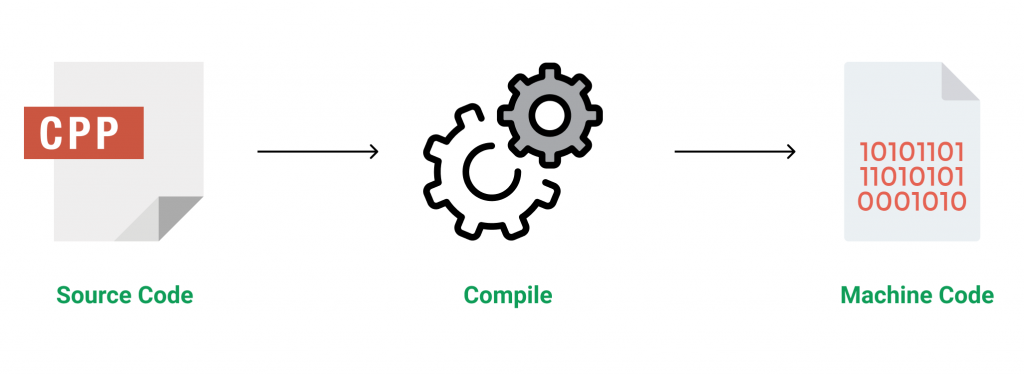
# **OOPS CONCEPT**

* The major factor of Object Oriented programming language approach is to remove some flaws of encountered in the procedural approach.
* OOPs treats data as a critical elements and does not allow it to flow freely around system. It ties data more closely to the functions that operate on it , and protects its from accidental modification from the outside functions .

# **Introduction to C++ Programming Language**

1. **C++** is a **high-level**, **general-purpose programming language** that was developed as an enhancement of the C language to include object-oriented paradigm. It is an imperative and a **compiled** language.
2. It was developed by Bjarne Stroustrup at Bell Labs in 1983 as an extension of the C programming language.
3. C++ is an object-oriented, multi-paradigm language that supports procedural, functional, and generic programming styles.
4. One of the key features of C++ is its ability to support low-level, system-level programming, making it suitable for developing operating systems, device drivers, and other system software. At the same time, C++ also provides a rich set of libraries and features for high-level application programming, making it a popular choice for developing desktop applications, video games, and other complex applications.
5. Object-Oriented Programming: C++ supports object-oriented programming, allowing developers to create classes and objects and to define methods and properties for these objects.
6. Templates: C++ templates allow developers to write generic code that can work with any data type, making it easier to write reusable and flexible code.
7. Standard Template Library (STL): The STL provides a wide range of containers and algorithms for working with data, making it easier to write efficient and effective code.
8. Exception Handling: C++ provides robust exception handling capabilities, making it easier to write code that can handle errors and unexpected situations.



***features & key-points*** of C++

1. **Simple**: It is a simple language in the sense that programs can be broken down into logical units and parts, has a rich library support and a variety of data-types.
2. **Machine Independent but Platform Dependent**: A C++ executable is not platform-independent (compiled programs on Linux won’t run on Windows), however they are machine independent.
3. **Mid-level language**: It is a mid-level language as we can do both systems-programming (drivers, kernels, networking etc.) and build large-scale user applications (Media Players, Photoshop, Game Engines etc.)
4. **Rich library support**: Has a rich library support (Both standard ~ built-in data structures, algorithms etc.) as well 3rd party libraries (e.g. Boost libraries) for fast and rapid development.
5. **Speed of execution**: C++ programs excel in execution speed. Since, it is a compiled language, and also hugely procedural. Newer languages have extra in-built default features such as garbage-collection, dynamic typing etc. which slow the execution of the program overall. Since there is no additional processing overhead like this in C++, it is blazing fast.
6. **Pointer and direct Memory-Access**: C++ provides pointer support which aids users to directly manipulate storage address. This helps in doing low-level programming (where one might need to have explicit control on the storage of variables).
7. **Object-Oriented**: One of the strongest points of the language which sets it apart from C. Object-Oriented support helps C++ to make maintainable and extensible programs. i.e. Large-scale applications can be built. Procedural code becomes difficult to maintain as code-size grows.
8. **Compiled Language**: C++ is a compiled language, contributing to its speed.

**Applications of C++:**   
C++ finds varied usage in applications such as:

* Operating Systems & Systems Programming. e.g. *Linux-based OS (Ubuntu etc.)*
* Browsers *(Chrome & Firefox)*
* Graphics & Game engines *(Photoshop, Blender, Unreal-Engine)*
* Database Engines *(MySQL, MongoDB, Redis etc.)*
* Cloud/Distributed Systems

### Here are some key points to keep in mind while working with C++:

1. Object-Oriented Programming: C++ is an object-oriented programming language, which means that it allows you to define classes and objects to model real-world entities and their behavior.
2. Strong Type System: C++ has a strong type system, which means that variables have a specific type and that type must be respected in all operations performed on that variable.
3. Low-level Access: C++ provides low-level access to system resources, which makes it a suitable choice for system programming and writing efficient code.
4. Standard Template Library (STL): The STL provides a large set of pre-written algorithms and data structures that can be used to simplify your code and make it more efficient.
5. Cross-platform Compatibility: C++ can be compiled and run on multiple platforms, including Windows, MacOS, and Linux, making it a versatile language for developing cross-platform applications.
6. Performance: C++ is a compiled language, which means that code is transformed into machine code before it is executed. This can result in faster execution times and improved performance compared to interpreted languages like Python.
7. Memory Management: C++ requires manual memory management, which can lead to errors if not done correctly. However, this also provides more control over the program’s memory usage and can result in more efficient memory usage.
8. Syntax: C++ has a complex syntax that can be difficult to learn, especially for beginners. However, with practice and experience, it becomes easier to understand and work with.

### **Advantages of C++:**

1. Performance: C++ is a compiled language, which means that its code is compiled into machine-readable code, making it one of the fastest programming languages.
2. Object-Oriented Programming: C++ supports object-oriented programming, which makes it easier to write and maintain large, complex applications.
3. Standard Template Library (STL): The STL provides a wide range of algorithms and data structures for working with data, making it easier to write efficient and effective code.
4. Platform Independent: C++ is a platform-independent language, meaning that code written in C++ can be compiled and run on a wide range of platforms, from desktop computers to mobile devices.
5. Large Community: C++ has a large, active community of developers and users, providing a wealth of resources and support for learning and using the language.

### **Disadvantages of C++:**

1. Steep Learning Curve: C++ can be challenging to learn, especially for beginners, due to its complexity and the number of concepts that need to be understood.
2. Verbose Syntax: C++ has a verbose syntax, which can make code longer and more difficult to read and maintain.
3. Error-Prone: C++ provides low-level access to system resources, which can lead to subtle errors that are difficult to detect and fix.

# **C++ Data Types:**

**How to Declare Variables?**

A typical variable declaration is of the form:

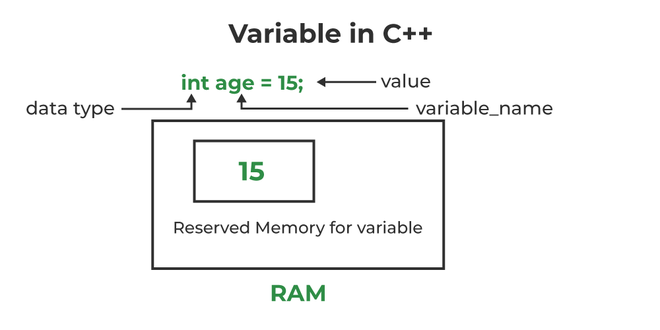
// Declaring a single variable

type variable\_name;

// Declaring multiple variables:

type variable1\_name, variable2\_name, variable3\_name;

A variable name can consist of alphabets (both upper and lower case), numbers, and the underscore ‘\_’ character. However, the name must not start with a number.



*Initialization of a variable in C++*

In the above diagram,

***datatype****: Type of data that can be stored in this variable.****variable\_name****: Name given to the variable.****value****: It is the initial value stored in the variable.*

**Examples**:

// Declaring float variable

float simpleInterest;

// Declaring integer variable

int time, speed;

// Declaring character variable

char var;

**We can also provide values while declaring the variables as given below:**

int a=50,b=100; //declaring 2 variable of integer type

float f=50.8; //declaring 1 variable of float type

char c='Z'; //declaring 1 variable of char type

**Rules For Declaring Variable**

* The name of the variable contains letters, digits, and underscores.
* The name of the variable is case sensitive (ex Arr and arr both are different variables).
* The name of the variable does not contain any whitespace and special characters (ex #,$,%,\*, etc).
* All the variable names must begin with a letter of the alphabet or an underscore(\_).
* We cannot used C++ keyword(ex float,double,class)as a variable name.

**Valid variable names:**

int x; //can be letters

int \_yz; //can be underscores

int z40;//can be letters

**Invalid variable names:**

int 89; Should not be a number

int a b; //Should not contain any whitespace

int double;// C++ keyword CAN NOT BE USED

**Difference Between Variable Declaration and Definition**

The **variable declaration** refers to the part where a variable is first declared or introduced before its first use. A **variable definition** is a part where the variable is assigned a memory location and a value. Most of the time, variable declaration and definition are done together.  
See the following C++ program for better clarification:

* C++

|  |
| --- |
| // C++ program to show difference between  // definition and declaration of a  // variable  #include <iostream>  **using** **namespace** std;    **int** main()  {      // this is declaration of variable a  **int** a;        // this is initialisation of a      a = 10;        // this is definition = declaration + initialisation  **int** b = 20;        // declaration and definition      // of variable 'a123'  **char** a123 = 'a';        // This is also both declaration and definition      // as 'c' is allocated memory and      // assigned some garbage value.  **float** c;        // multiple declarations and definitions  **int** \_c, \_d45, e;        // Let us print a variable      cout << a123 << endl;    **return** 0;  } |

**Output**

a

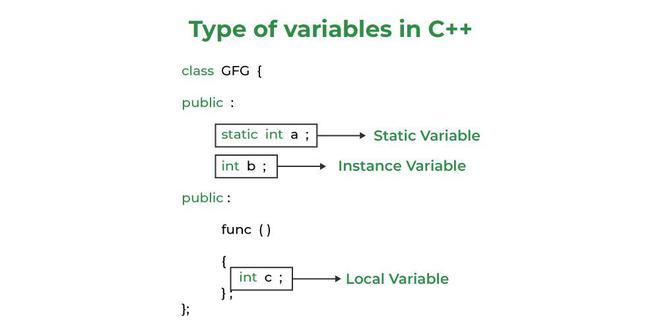
**Time Complexity:** O(1)

**Space Complexity:** O(1)

**Types of Variables**

There are three types of variables based on the scope of variables in C++

* **Local Variables**
* **Instance Variables**
* **Static Variables**



*Types of Variables in C++*

Let us now learn about each one of these variables in detail.

1. **Local Variables**: A variable defined within a block or method or constructor is called a local variable.
   * These variables are created when entered into the block or the function is called and destroyed after exiting from the block or when the call returns from the function.
   * The scope of these variables exists only within the block in which the variable is declared. i.e. we can access this variable only within that block.
   * Initialization of Local Variable is Mandatory.
2. **Instance Variables**: Instance variables are non-static variables and are declared in a class outside any method, constructor, or block.
   * As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
   * Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier then the default access specifier will be used.
   * Initialization of Instance Variable is not Mandatory.
   * Instance Variable can be accessed only by creating objects.
3. **Static Variables**: Static variables are also known as Class variables.
   * These variables are declared similarly as instance variables, the difference is that static variables are declared using the [static keyword](https://www.geeksforgeeks.org/static-keyword-cpp/) within a class outside any method constructor or block.
   * Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
   * Static variables are created at the start of program execution and destroyed automatically when execution ends.
   * Initialization of Static Variable is not Mandatory. Its default value is 0
   * If we access the static variable like the Instance variable (through an object), the compiler will show the warning message and it won’t halt the program. The compiler will replace the object name with the class name automatically.
   * If we access the static variable without the class name, the Compiler will automatically append the class name.

**Instance Variable Vs Static Variable**

* Each object will have its **own copy** of the instance variable whereas We can only have **one copy** of a static variable per class irrespective of how many objects we create.
* Changes made in an instance variable using one object will **not be reflected** in other objects as each object has its own copy of the instance variable. In the case of static, changes**will be reflected** in other objects as static variables are common to all objects of a class.
* We can access instance variables **through object references** and Static Variables can be accessed **directly using the class name.**
* The syntax for static and instance variables:

class Example

{

static int a; // static variable

int b; // instance variable

}