**Learn C++**

**C++ Literals**

1. Integers
2. Floating-point Literals
3. Characters
4. Escape Sequences
5. String Literals

**1. Integers**

There are three types of integer literals in C programming:

1. decimal (base 10)
2. octal (base 8)
3. hexadecimal (base 16)

For example:

* Decimal: 0, -9, 22 etc
* Octal: 021, 077, 033 etc
* Hexadecimal: 0x7f, 0x2a, 0x521 etc

**2. Floating-point Literals**

* 2.9
* 314159E-5 = 3.14159

Note : E-5 = 10-5

**4. Escape Sequences**

\b Backspace

\f Form feed

\n Newline

\r Return

\t Horizontal tab

\v Vertical tab

\\ Backslash

\' Single quotation mark

\" Double quotation mark

\? Question mark

\0 Null Character

**C++ Constants ( two ways )**

const int lol= 299792458;

#define lol299792458.

**C++ Data Types**

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Meaning** | **Size (in Bytes)** |
| **int** | **Integer** | **2 or 4** |
| **float** | **Floating-point** | **4** |
| **double** | **Double Floating-point** | **8** |
| **char** | **Character** | **1** |
| **wchar\_t** | **Wide Character** | **2** |
| **bool** | **Boolean** | **1** |
| **void** | **Empty** | **0** |

Int = ( -2147483648 to 2147483647)

INT\_MAX = 2147483647 ( #include <limits.h> )

INT\_MIN = -2147483648 ( #include <limits.h> )

double distance = 45E12 // 45E12 is equal to 45\*1012

**C++ wchar\_t**

It is used to represent characters that require more memory to represent them than a single char.

wchar\_t test = L'ם' // storing Hebrew character;

**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

Ref : <https://www.programiz.com/cpp-programming/data-types>

**C++ Type Conversion**

There are two types of type conversion in C++.

1. Implicit Conversion
2. Explicit Conversion (also known as Type Casting)

**Implicit Conversion**

The type conversion that is done automatically by the compiler is known as **implicit type conversion**. This type of conversion is also known as **automatic conversion**.

**Example - 1**

int x = 9;

double y;

y = x;

**Example - 2**

int x;

double y = 10.56;

x = y;

**C++ Explicit Conversion**

When the user manually changes data from one type to another, this is known as **explicit conversion**. This type of conversion is also known as **type casting**.

There are three major ways in which we can use explicit conversion in C++.

1. C-style type casting (also known as cast notation)
2. Function notation (also known as old C++ style type casting)
3. Type conversion operators

**C-style type casting (also known as cast notation)**

int x = 26;

double y;

y = (double)x;

**Function-style Casting**

int x = 26;

double y;

y = double(x);

**Type Conversion Operators**

Besides these two type castings, C++ also has four operators for type conversion. They are known as type conversion operators.

1. static\_cast
2. dynamic\_cast
3. const\_cast
4. reinterpret\_cast

**C++ Operators**

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

1. Arithmetic Operators
2. Assignment Operators
3. Relational Operators
4. Logical Operators
5. Bitwise Operators
6. Other Operators

**Arithmetic Operators: + - \* / %**

**Relational Operators: == != > < >= <=**

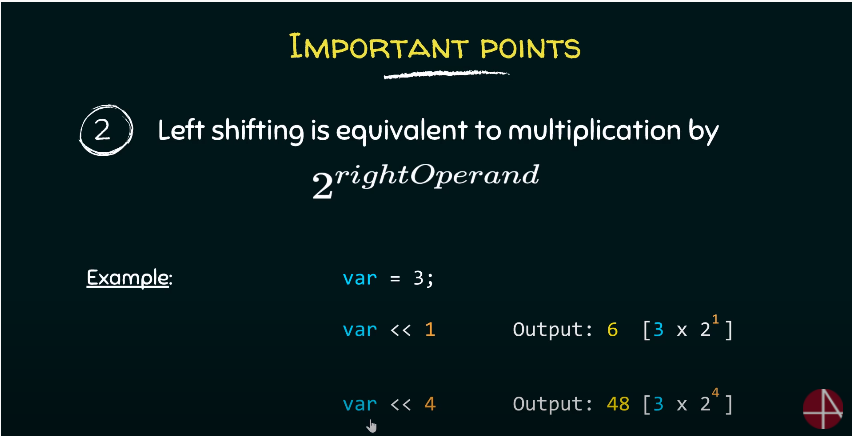
**C++ Logical Operators: && || !**

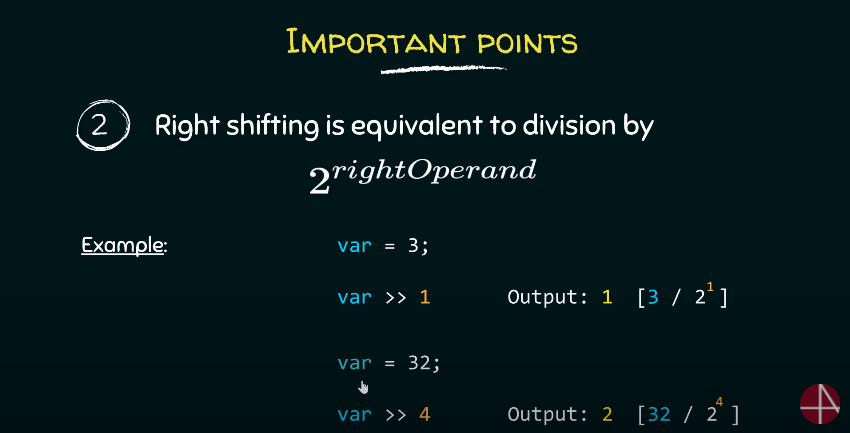
**C++ Bitwise Operators**

|  |  |
| --- | --- |
| Operator | Description |
| **&** | **Binary AND** |
| **|** | **Binary OR** |
| **^** | **Binary XOR** |
| **~** | **Binary One's Complement** |
| **<<** | **Binary Shift Left** |
| **>>** | **Binary Shift Right** |

|  |  |  |
| --- | --- | --- |
| **Binary AND ( & )**  **0 1 1 1**  **0 1 0 0**  **--------**  **0 1 0 0** | **Binary OR ( | )**  **0 1 1 1**  **0 1 0 0**  **--------**  **0 1 1 1** | **Binary NOT ( ~ ) /**  **Binary One's Complement**  **0 1 0 0**  **--------**  **1 0 1 1** |

|  |  |
| --- | --- |
| **Left Shift ( << )**  3 << 1  **0000 0011**  After shifting 1 bit  **0000 0110**  NB: Trailing position filled with Zero. | **Right Shift ( >> )**  3 >> 1  **0000 0011**  After shifting 1 bit  **0000 0001**  NB: Leading position filled with Zero. |

****

****

**Bitwise XOR ( ^ )**

1 1 = 0

0 0 = 0

Else 1

**0 1 1 1**

**0 1 0 0**

**--------**

**0 0 1 1**

**Others Operators**

Apart from the operators discussed above, there are a few other operators, such as sizeof ? . & etc., that cannot be neatly classified into one or another type. We will learn more about these operators in later tutorials.

**Comments**

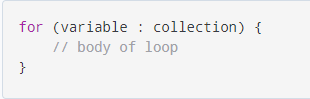
As a general rule of thumb, use comments to explain Why you did something rather than How you did something, and you are good.

**Flow Control**

There are 3 types of loops in C++

1. for loop
2. while loop
3. do...while loop

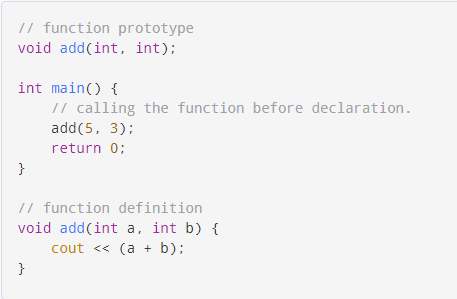
In C++11, a new range-based for loop was introduced to work with collections such as arrays and vectors. Its syntax is:



There are two types of function

1. Standard Library Functions: **Predefined in C++**
2. User-defined Function: **Created by users**

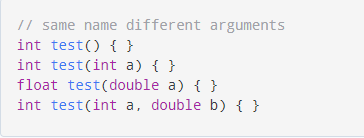
If we want to write function after main we need to define the function before..



**C++ Function Overloading**

In C++, two functions can have the same name if the **number and/or type of arguments** passed is different.

These functions having the same name but different arguments are known as overloaded functions. For example:



Notice that the return types of all these 4 functions are not the same. **Overloaded functions may or may not have different return types but they must have different arguments**.

**Working of default arguments**

Once we provide a default value for a parameter, all subsequent parameters must also have default values. For example,

// Invalid

void add(int a, int b = 3, int c, int d);

// Invalid

void add(int a, int b = 3, int c, int d = 4);

// Valid

void add(int a, int c, int b = 3, int d = 4);

**C++ Storage Class**

Every variable in C++ has two features: type and storage class.

And, storage class controls two different properties of a variable: lifetime (determines how long a variable can exist) and scope (determines which part of the program can access it).

Depending upon the storage class of a variable, it can be divided into 4 major types:

1. Local variable
2. Global variable
3. Static local variable
4. Register Variable
5. Thread Local Storage

**Global variable:** Its life ends only when the program ends.

**Static local variable:** A static local variable exists only inside a function where it is declared (similar to a local variable) but its lifetime starts when the function is called and ends only when the program ends.

**Register Variable (Deprecated in C++11) :** If a program encounters a register variable, it stores the variable in the processor's register rather than memory if available. This makes it faster than the local variables.

However, this keyword was deprecated in C++11 and should not be used.

**Difference Between a Pointer and a Reference C++**

A pointer in C++ is a variable that holds the memory address of another variable.

A reference is an alias for an already existing variable. Once a reference is initialized to a variable, it cannot be changed to refer to another variable. Hence, a reference is similar to a const pointer.

**Choose auto x :** when you want to work with copies.

**Choose auto &x :**  when you want to work with original items and may modify them.

**Choose auto const &x :**  when you want to work with original items and will not modify them.

**Array & String**

In C++, the size and type of arrays cannot be changed after its declaration.

**The function parameter int m[5] converts to int\* m;** This points to the same address pointed by the array marks. This means that when we manipulate m[5] in the function body, **we are actually manipulating the original array marks.**

**We can skip the first array while using it as a parameter. It will automatically point the initial array.**

One dimensional array: **int m[] / int m[2]**

Multidimensional arrays: **int [][2] / int [3][2]**

**How to define a C-string?**

char str[] = "C++";

char str[4] = "C++";

char str[] = {'C','+','+','\0'};

char str[4] = {'C','+','+','\0'};

**cin** takes string without space but **cin.get(str, 100); / getline(cin, str);**  takes string with space.

char str[100];

cin.get(str, 100);

// Declaring a string object

string str;

getline(cin, str);

**Enum**

An enumeration is a user-defined data type that consists of integral constants

enum season { spring, summer, autumn, winter };

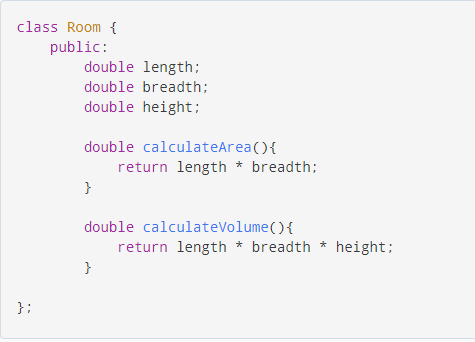
Enum is a good choice to work with flags.

<https://www.programiz.com/cpp-programming/enumeration>

**Class & Objects**

A class is a blueprint for the object.

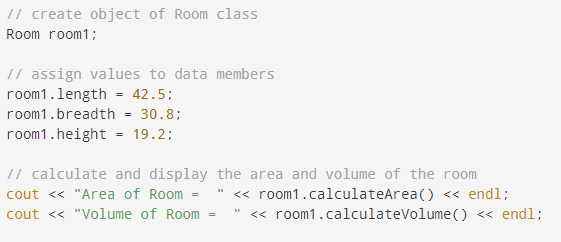
We can think of a class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows, etc. Based on these descriptions we build the house. House is the object.



**Object**

**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, we need to create objects.



Note the use of the keyword public in the program. This means the **members are public** and can be accessed anywhere from the program.

As per our needs, we can also create **private members** using the private keyword. The private members of a class can only be accessed from within the class.

**Constructors**

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

In C++, a constructor with parameters is known as a parameterized constructor. This is the preferred method to initialize member data.

The copy constructor in C++ is used to copy data of one object to another.

**We can pass objects to function like others types(String, Int ... ).**

**Operator Overloading => Have to check later**

**C++ Pointers => Have to make a separate note for pointers.**

**C++ OOP => Have to make a separate note for OOP.**