Introduction, Variables and Operators

INDEX

Table of Contents

[Introduction to C++ 2](#_Toc142430709)

[Preprocessor Directives in C++ 3](#_Toc142430710)

[Basic Input/Output in C++ 5](#_Toc142430711)

[C/C++ Preprocessors 7](#_Toc142430712)

[Comments in C++ 10](#_Toc142430713)

[Data Types and Variables in C++ 11](#_Toc142430714)

[Variable 11](#_Toc142430715)

[Data Type 11](#_Toc142430716)

[Qualifiers, Modifiers & Specifiers 12](#_Toc142430717)

[Declaration, Definition & Initialization 18](#_Toc142430718)

[Scope of Variables 19](#_Toc142430719)

[sizeof Operator in C++ 20](#_Toc142430720)

[Literals in C++ 23](#_Toc142430721)

[Type Casting in C++ 24](#_Toc142430722)

[Escape Sequences in C++ 27](#_Toc142430723)

[IO Manipulation 28](#_Toc142430724)

[Floating point Default Format 33](#_Toc142430725)

[Floating point Manipulating Default Format 34](#_Toc142430726)

[Operators in C++ 35](#_Toc142430727)

[Operator Precedence: 39](#_Toc142430728)

# Introduction to C++

To create an object file and a binary file in C++, you need to follow these steps:

main.cpp

A screen shot of a computer

Description automatically generated

Compile the source code to generate an object file:

A black and grey rectangular object

Description automatically generated

Link the object file with C++ standard libraries to create the binary executable

A black and grey background

Description automatically generated

Run the binary executable



# Preprocessor Directives in C++

A preprocessor directive is a statement which gets processed by the C++ preprocessor before compilation.A diagram of a process

Description automatically generatedFor basic programming in C++, we only need to understand the **#include** and the **#define** directives.

**#include** is a preprocessor directive used to include external files or libraries into your code. It allows you to use code from other files in your current source code, which enables code reusability and modularity.

The syntax for #define is as follows:

A black and grey background

Description automatically generated

Or

A black and grey background

Description automatically generated

**<header\_file>**: This form is used to include standard library header files.

**"header\_file"**: This form is used to include user-defined header files or files located in the same directory as the source file. The compiler will first search for the header file in the current directory and then in the standard system directories if it's not found in the current directory.

For example, if you want to use functionalities from the iostream library, you can include it in your C++ code using #include <iostream> like this:

A screen shot of a computer

Description automatically generated

#define is a preprocessor directive used to define constants or macros. It allows you to give a name to a specific value or expression, making it easier to use and maintain throughout your code.

The syntax for #define is as follows:

A black and grey rectangular object

Description automatically generated

When the preprocessor encounters a #define directive, it replaces all occurrences of identifier with the corresponding value in the source code before the actual compilation takes place. This process is known as macro substitution.

Here's an example of using #define to define a constant:

A screenshot of a computer

Description automatically generated

In this example, #define PI 3.14159 defines a constant named "PI" with the value 3.14159. Whenever the preprocessor encounters the identifier "PI" in the code, it will replace it with the value 3.14159.

# Basic Input/Output in C++

Header files available in C++ for Input/Output operations are:

1. **iostream**: iostream stands for standard input-output stream. This header file contains definitions of objects like cin, cout, cerr, etc.
2. **iomanip**: iomanip stands for input-output manipulators. The methods declared in these files are used for manipulating streams. This file contains definitions of setw, setprecision, etc.
3. **fstream**: This header file mainly describes the file stream. This header file is used to handle the data being read from a file as input or data being written into the file as output.

Standard output stream (cout) & Standard input stream (cin):

A computer screen shot of a black screen

Description automatically generated

Un-buffered standard error stream (cerr):

A screenshot of a computer program

Description automatically generated

buffered standard error stream (clog):

A computer screen shot of a program code

Description automatically generated

# C/C++ Preprocessors

In C and C++, the preprocessor is a separate phase of the compilation process that occurs before the actual compilation of the source code. It is responsible for performing various text manipulations on the source code before it is passed to the compiler. Preprocessor directives start with the # symbol and are processed before the code is compiled.

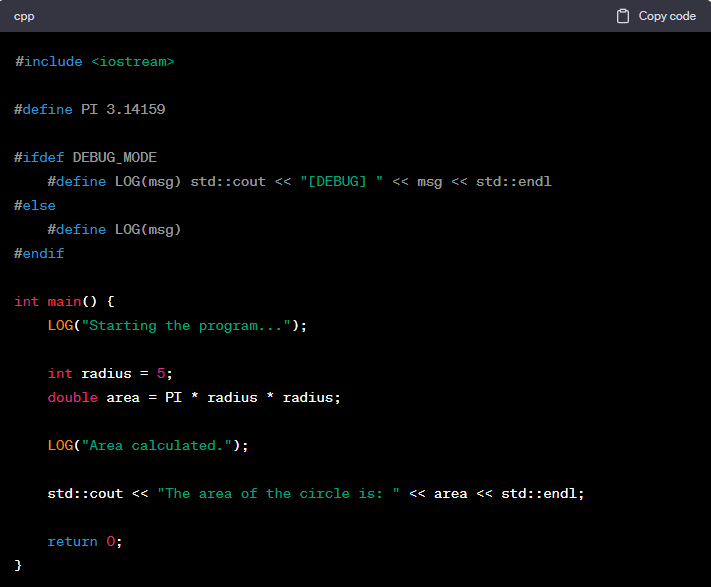
A screenshot of a computer program

Description automatically generated

Here are some common preprocessor directives and their functionalities:

1. **#include:** Used to include header files into the source code. It allows you to use functions and features from other files in your program.
2. **#define:** Used to define constants or macros. It allows you to give a name to a specific value or expression, making it easier to use and maintain throughout your code.
3. **#ifdef, #ifndef, #else, #endif:** Used for conditional compilation. These directives allow you to include or exclude blocks of code based on certain conditions.
4. **#if, #elif, #else, #endif:** Used for more complex conditional compilation. These directives allow you to evaluate expressions to include or exclude blocks of code.
5. **#pragma:** Used to provide compiler-specific instructions or control certain compilation behaviors. Pragmas are implementation-specific and may vary between different compilers.
6. **#error:** Used to generate a compilation error with a custom error message. It is often used to enforce certain conditions in the code.
7. **#warning:** Used to generate a compilation warning with a custom warning message. It is helpful for flagging potential issues in the code.

For example, here's a simple C++ code that uses some preprocessor directives:



OP



OP: if we define DEBUG\_MODE

A blue background with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

# Comments in C++

comments are text annotations within the source code that provide explanatory or informative notes to programmers. Comments are ignored by the compiler and do not affect the program's functionality.

A screenshot of a computer

Description automatically generated

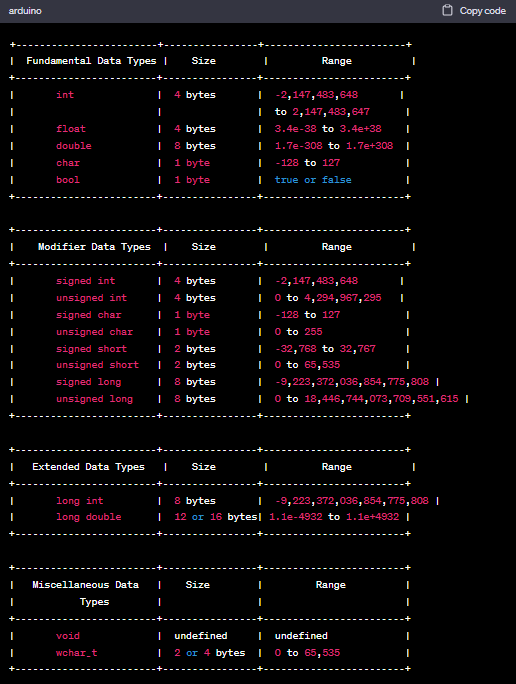
# Data Types and Variables in C++

Variable: A variable is a named storage location in a computer's memory that holds a value.

A black background with white text

Description automatically generated

Data Type: A data type in programming defines the type of data that a variable can hold. **It specifies the size and format of the data, which determines how the data is stored in memory** and how it can be manipulated.



wchar\_t: It is a wide characters, it supports Unicode, Unicode is extended characters range, with Unicode you can store local language also like Chinese or geek characters.

### Qualifiers, Modifiers & Specifiers

A screenshot of a computer

Description automatically generated

Qualifiers**:**

**Const:** The const keyword in C++ is used to declare constants, which are values that **cannot be modified after initialization**, attempting to modify it will result in a **compilation error**.

There are a certain set of rules for the declaration and initialization of the constant variables:

* The const variable cannot be left un-initialized at the time of the declaration.
* It cannot be assigned value anywhere in the program.

A black screen with a black border

Description automatically generated

There are two primary ways the const keyword can be used:

1. **Constant Variables**: In this example, x is a constant variable of type int, and its value is initialized to 10. Once initialized, you cannot change the value of x throughout the program.



1. **Constant Pointers**: In this example, **ptr is a pointer to a constant integer**. It can point to an integer (y in this case), but **it cannot modify the value it points to**. The **pointer ptr can be changed** to point to another constant integer, but it cannot modify the integer it currently points to.

A black and grey background

Description automatically generated

Here, **ptr2 is a constant pointer to an integer**. It can point to an integer (z in this case), but it **cannot be changed to point to another integer**. However, it can modify the value of the integer it points to (z).

A black and grey background

Description automatically generated

Here, **ptr is a constant pointer to a constant integer**. We have a constant integer variable x with the value 10. Once initialized, the value of x cannot be modified throughout the program and the pointer itself is constant and cannot be changed to point to another memory address.

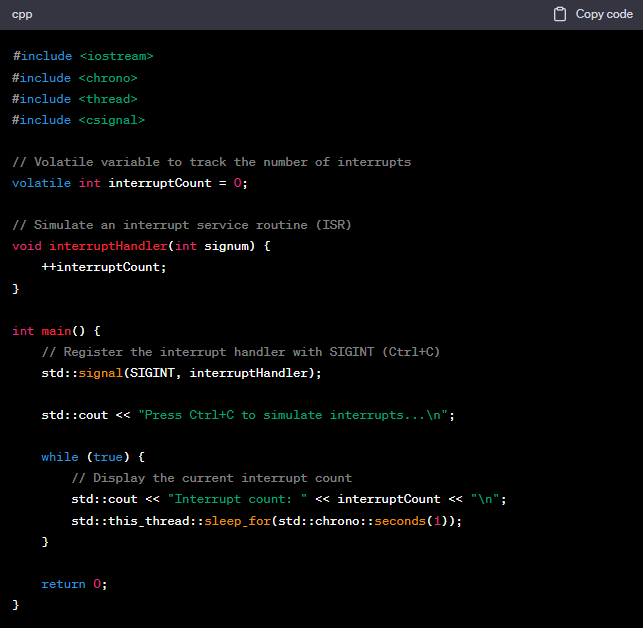
A black and grey background

Description automatically generated

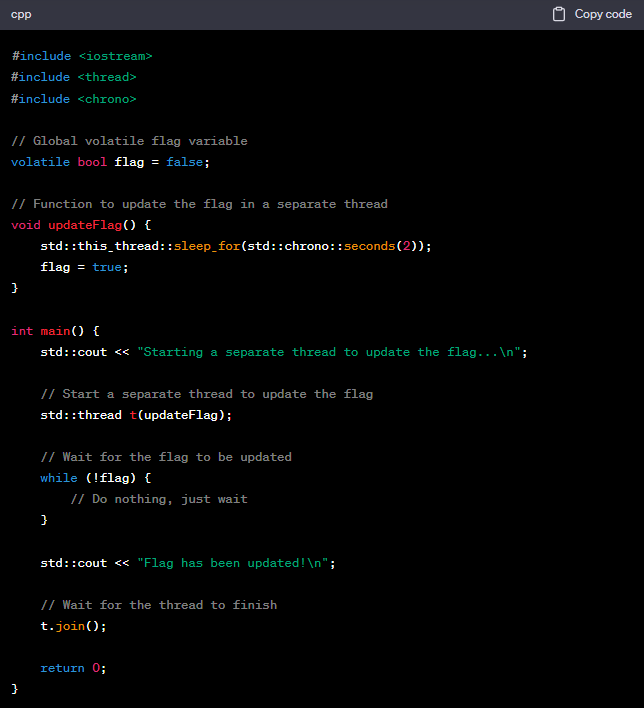
1. **Const method: TODO**

**Volatile**: Volatile variables can be read or written by external factors such as hardware, other threads, or interrupts.

Consider a scenario where we have a simple program that involves an interrupt service routine (ISR) that updates a variable representing the number of times the interrupt has occurred. The main program then reads and displays the value of this variable.



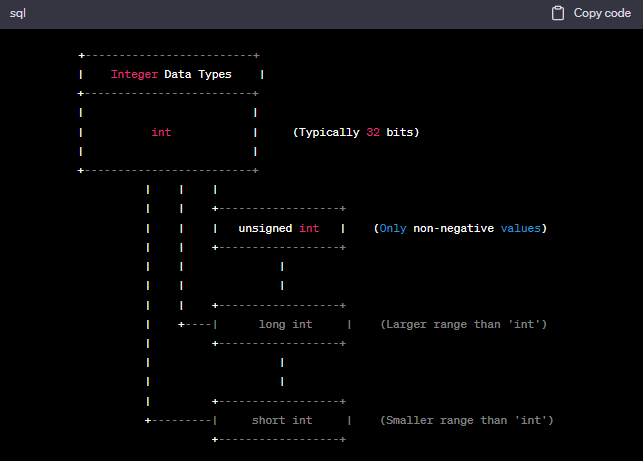
A thread-based example that demonstrates the significance of using the volatile keyword when working with shared variables in a multi-threaded environment. Here we update a shared flag variable in a separate thread, and the main thread waits for the flag to be updated:



The **volatile** keyword is important here to ensure that the main thread correctly observes the updated value of **flag** set by the separate thread. Without **volatile**, the compiler might optimize the read operation inside the loop and keep using a cached value of **flag**, leading to an infinite loop.

By using volatile, we ensure that the main thread reads the latest value of flag from memory, reflecting the updates made by the separate thread.

Modifiers:Used to alter the characteristics of data types or variables, typically by changing their size, range, or storage behavior.

****

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

Declaration, Definition & Initialization**:**

1. **Declaration:** A declaration **introduces the existence of a variable** to the compiler **without allocating memory** for it or assigning an initial value. It informs the compiler about the variable's name and data type.
2. **Definition:** A definition not only **introduces the existence of a variable** but also **allocates memory to store its value**. It reserves a memory space for the variable to hold data of the specified type.

In the case of **int x;** the line of code both declares and defines the variable x

* It is a declaration because it tells the compiler that there is a variable named x of type int.
* It is also a definition because it allocates memory for the variable x to store an integer value.

1. **Initialization:** Initialization sets the initial value of the variable after it has been defined and allocated memory. It assigns a specific value to the variable, so it's no longer in an undefined state.

**A screenshot of a computer

Description automatically generated**

Here's a breakdown of the line **extern int x;**

**extern:** The extern keyword informs the compiler that the variable is **declared elsewhere** and should not allocate memory for it in the current file.

**int**: The data type of the variable is int, indicating that x is an integer variable.

**x**: x is the name of the variable being declared.

Scope of Variables**:**

**A screenshot of a computer program

Description automatically generated**

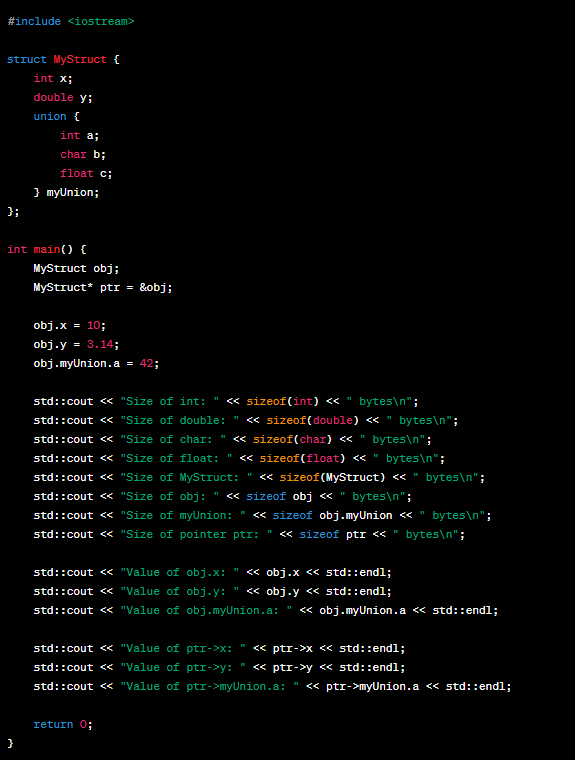
# sizeof Operator in C++

**sizeof** is an operator in C++ that allows you to determine the size in bytes, of a data type or an object. Internally, **sizeof** is evaluated during the **compilation phase** rather than at runtime. Compiler already knows their sizes based on the target architecture and compiler settings. So, when the compiler encounters the sizeof operator, it performs the following actions:

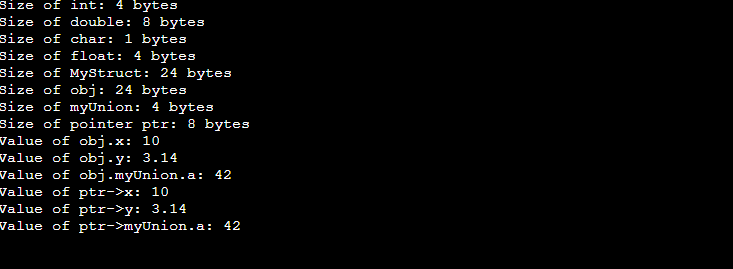
1. For **built-in data types** like int and double, the compiler knows their sizes. It **directly replaces** sizeof(int) with the size of an int data type.
2. For **user-defined types** like MyStruct, the compiler calculates the size by summing up the sizes of its members. In this case, MyStruct has an int, a double and an union. The compiler calculates the size of MyStruct based on the sizes of these members (considering alignment requirements and padding) and replaces sizeof(MyStruct) with the result.

sizeof **obj.myUnion** will be 4 bytes, which is the size of the union (int a) since it is the largest member.

sizeof(x++): x++ expression never evaluated because sizeof is compile time operator. In other words, at the time of compilation sizeof(x++) is replaced by 4. So, x++ never evaluates.



OP



# Literals in C++

Literals in C++ are fixed values that are directly written in the source code.

1. **Numeric Literals**: Represent numeric values.

* Integer Literals: Whole numbers without a fractional part.
* Decimal: Base 10 (e.g., 123).
* Octal: Base 8 (e.g., 075).
* Hexadecimal: Base 16 (e.g., 0x1A).

1. **Floating-Point** Literals: Numbers with a fractional part (e.g., 3.14).
2. **Character Literals**: Represent single characters enclosed in single quotes (e.g., 'A').
3. **String Literals**: Represent sequences of characters enclosed in double quotes (e.g., "Hello").
4. **Boolean Literals**: Boolean literals are written directly as true or false without the need for any special syntax or keywords.

A computer screen shot of a program code

Description automatically generated

OP

A black background with a black square

Description automatically generated

# Type Casting in C++

Type casting in C++ is the process of converting one data type to another.

There are two types of type casting in C++:

1. **Implicit Type Casting (Automatic Type Conversion)**:

* This type of type casting is done automatically by the compiler.
* It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float, suppose long long:8 byte & float: 4 byte).

A black background with white and yellow lines

Description automatically generated

A screenshot of a computer program

Description automatically generated

Since unsigned int cannot represent negative values, the sign is lost during the explicit type casting. The result in unsignedValue will be some unexpected positive value due to the unsigned integer overflow.

A black screen with white text

Description automatically generated

1. **Explicit Type Casting (Manual Type Conversion)**:

* **Conversion by assignment**:

This is c style conversion.

A black screen with white text

Description automatically generated

OP

A black screen with white text

Description automatically generated

* **Conversion using cast operator** (Static Cast)

Why better than c style? It does validity check before doing the conversion. It will check you can convert or not.

A computer screen shot of a program code

Description automatically generated

OP

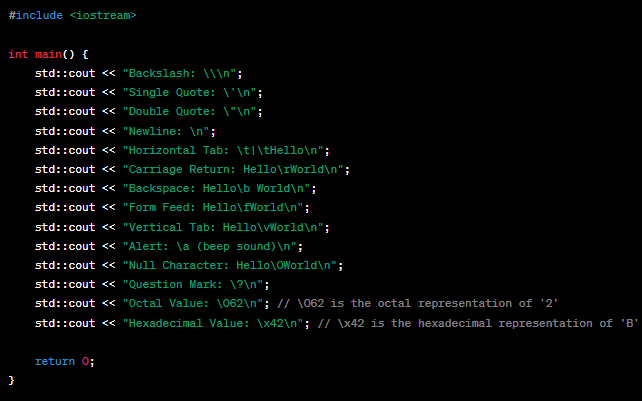
A screenshot of a computer

Description automatically generated

There are many more like Dynamic Cast, Const Cast, Reinterpret Cast.

Escape Sequences in C++

**List of Escape Sequences**



OP

A computer screen shot of a black background

Description automatically generated

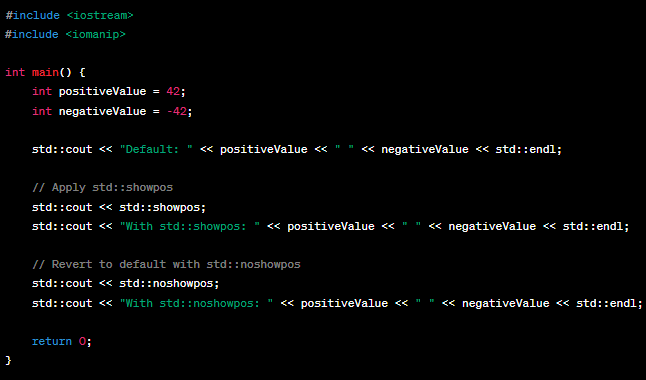
# IO Manipulation

Input/Output (IO) Manipulation in C++ refers to the techniques used to control the formatting and presentation of data when reading input from the user or displaying output to the console.

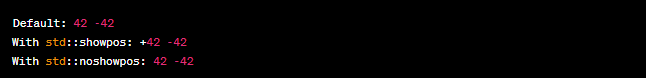
IO Manipulation is facilitated using the **<iomanip>** header.

Some important manipulators in <ios> are:

1. **showpos**: By default, positive numeric values do not show the plus sign when printed. It forces to show a positive sign on positive numbers.
2. **noshowpos**: It forces not to write a positive sign on positive numbers.



OP

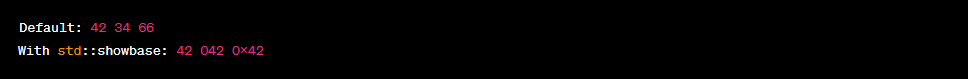


1. **showbase**: used to control the display of the base prefix for integral values (integers) when they are printed. By default, integral values are displayed without any base prefix (e.g., 10, 42).

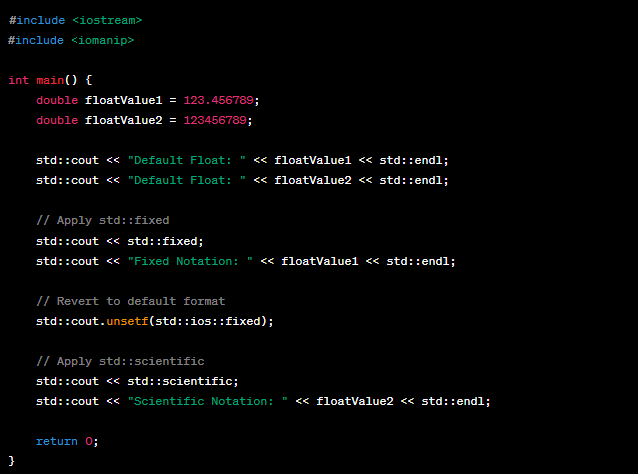
A black background with white text

Description automatically generated

OP



1. **fixed**: It displayed in fixed notation with all its decimal places. (no power (or e) uses)
2. **scientific**: It uses scientific floating-point notation. (we will use e)

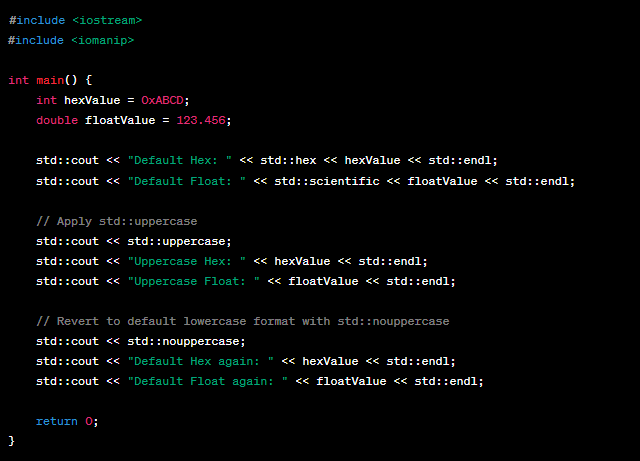


OP

A black background with a black square

Description automatically generated

1. **uppercase**: Used to control the case of hexadecimal and floating-point values when they are printed. By default, hexadecimal values are displayed in lowercase, and floating-point values are displayed in lowercase exponential notation.
2. **nouppercase**: Used to revert the case of hexadecimal and floating-point values back to lowercase.

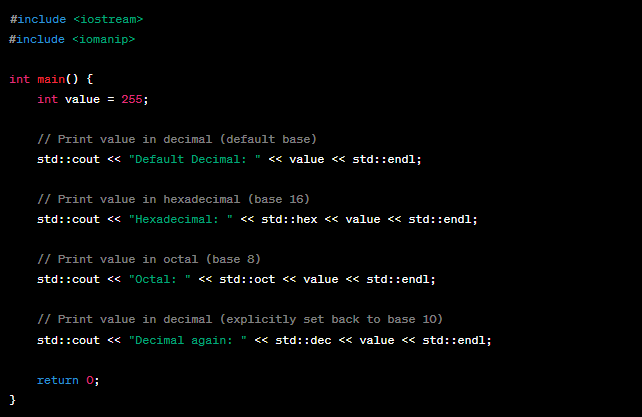


OP

A black background with a black square

Description automatically generated

1. **hex, oct, dec**: The IO Manipulators std::hex, std::oct, and std::dec are used to change the base in which integral values are printed: hexadecimal (base 16), octal (base 8), and decimal (base 10), respectively.

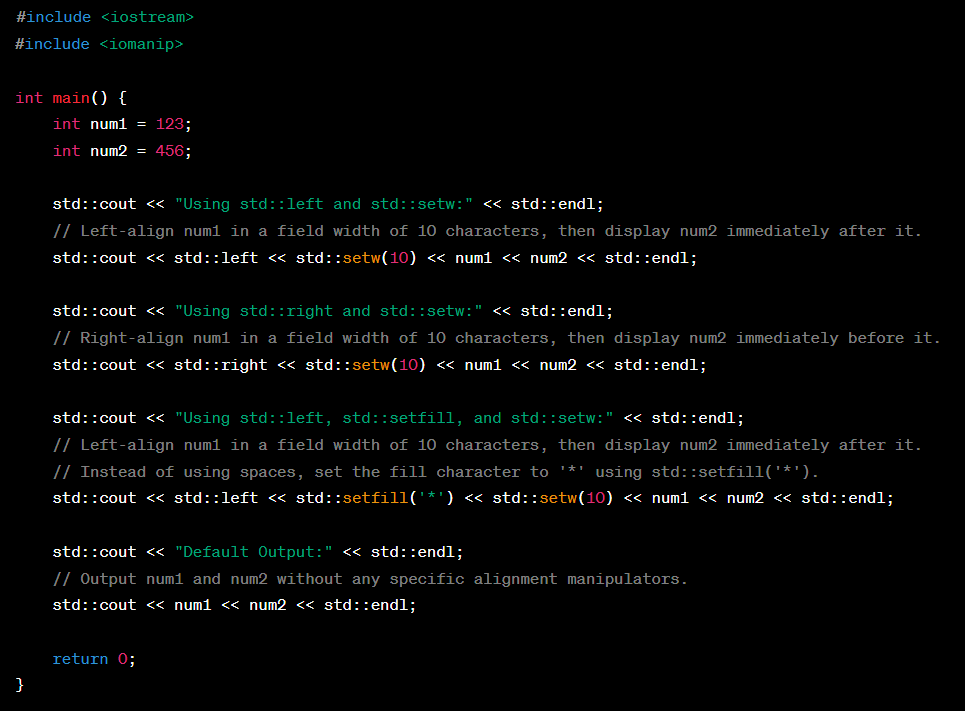


OP

A black background with a black square

Description automatically generated

1. **left, right, setfill, & setw**:

****

OP

**A black screen with white text

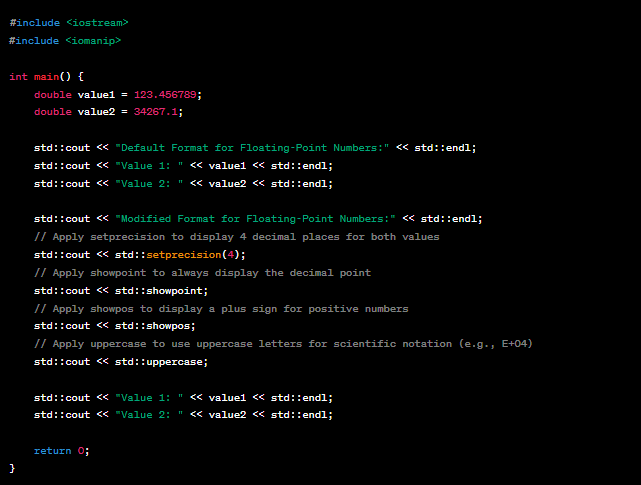
Description automatically generated**

# Floating point Default Format

A white paper with writing on it

Description automatically generated

# Floating point Manipulating Default Format

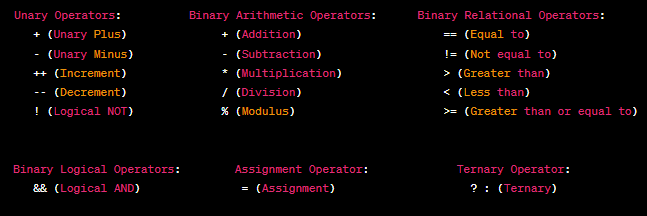


OP

A black screen with colorful text

Description automatically generated

# Operators in C++



Comparison operator: <=> operator is known as the "Three-Way Comparison Operator". The <=> operator returns one of three values:

1. It returns a **negative** value if the left operand is less than the right operand.
2. It returns **zero** if the left operand is equal to the right operand.
3. It returns a **positive** value if the left operand is greater than the right operand.

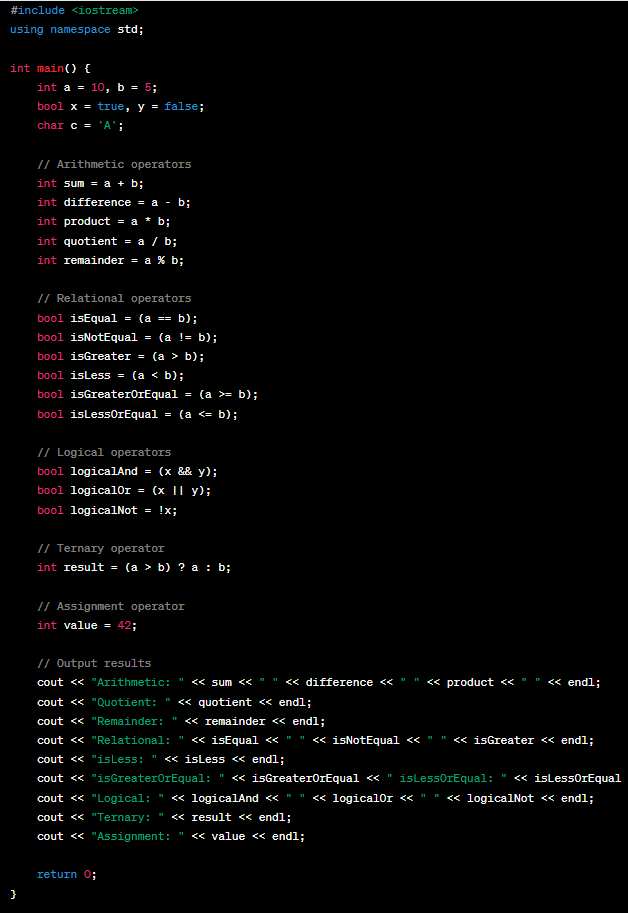
A computer screen with text and numbers

Description automatically generated

OP



Example of operators

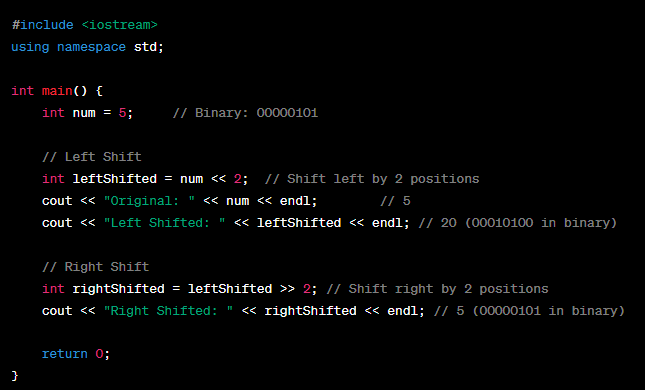


OP

A black screen with white text

Description automatically generated

Bitwise left and right



OP

A black background with a black square

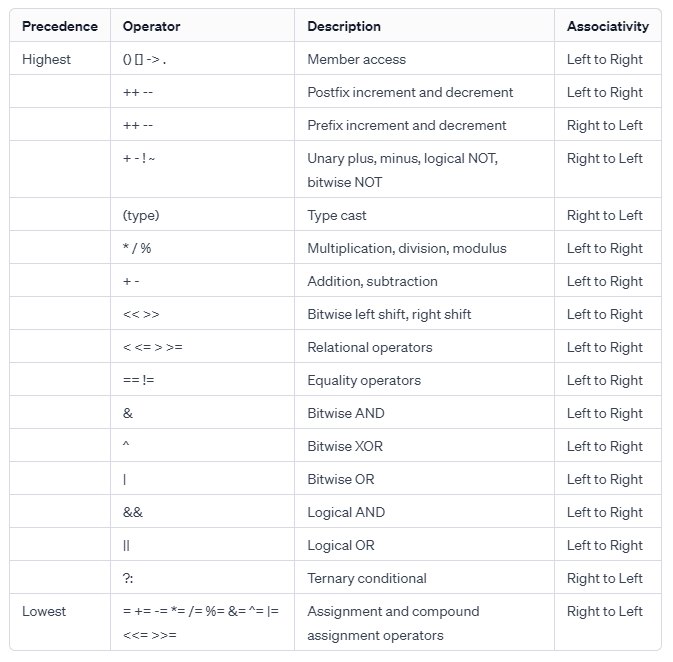
Description automatically generated with medium confidence

|  |  |
| --- | --- |
|  | **Note**:We should not use left shift, right shift, or bitwise operator on negative number because, behavior is undefined. |

|  |  |
| --- | --- |
| A white paper with writing on it  Description automatically generated | A white paper with writing on it  Description automatically generated |

## Operator Precedence:

If you don’t want to use this table then you can use brackets, which makes code more readable.



When does Associativity work?

Suppose we have two operators have same precedence in a single expression then we use associativity to process the expression in respective manner.