Data Types

Data types specify the type of data that a variable can store. Whenever a variable is defined in C++, the compiler allocates some memory for that variable based on the data type with which it is declared as every data type requires a different amount of memory.

In C++, data types are classified into the following types:

S. No.	Type	Description	Data Types
1	Basic Data Types	Built-in or primitive data types that are used to store simple values.	int, float, double, char, bool, void
2	Derived Data Types	Data types derived from basic types.	array, pointer, reference, function
3	User Defined Data Types	Custom data types created by the programmer according to their need.	class, struct, union, typedef, using

Basic Data Types

The data type specifies the size and type of information the variable will store:

Data Type	Size	Description
`boolean`	1 byte	Stores true or false values
`char`	1 byte	Stores a single character/letter/number, or ASCII values
`int`	2 or 4 bytes	Stores whole numbers, without decimals
`float`	4 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 6-7 decimal digits
`double`	8 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits

Numeric Types

Use *int* when you need to store a whole number without decimals, like 35 or 1000, and *float* or *double* when you need a floating-point number (with decimals), like 9.99 or 3.14515.

```
int myNum = 1000;
cout << myNum;

float myNum = 5.75;
cout << myNum;

double myNum = 19.99;
cout << myNum;
```

float vs double

The **precision** of a floating-point value indicates how many digits the value can have after the decimal point. The precision of float is only six or seven decimal digits, while double variables have a precision of about 15 digits. Therefore, it is safer to use double for most calculations.

Scientific Numbers

A floating-point number can also be a scientific number with an "e" to indicate the power of 10

```
float f1 = 35e3;
double d1 = 12E4;
cout << f1;
cout << d1;
```

Boolean Types

A boolean data type is declared with the *bool* keyword and can only take the values true or false. When the value is returned, true = 1 and false = 0.

```
bool isCodingFun = true;
bool isFishTasty = false;
```

```
cout << isCodingFun;      // Outputs 1 (true)
cout << isFishTasty;      // Outputs 0 (false)
```

Character Types

The *char* data type is used to store a **single** character. The character must be surrounded by *single quotes*, like 'A' or 'c'

```
char myGrade = 'B';
cout << myGrade;
```

String Types

The *string* type is used to store a sequence of characters (text). String values must be surrounded by *double quotes*:

```
string greeting = "Hello";
cout << greeting;
```

Note: To use strings, you must include an additional header file in the source code, the <string> library:

```
// Include the string library
#include <string>

// Create a string variable
string greeting = "Hello";

// Output string value
cout << greeting;
```

Operators

Operators are used to perform operations on variables and values. In the example below, we use the + **operator** to add together two values:

```
int x = 100 + 50;
```

Although the + operator is often used to add together two values, like in the example above, it can also be used to add together a variable and a value, or a variable and another variable:

```
int sum1 = 100 + 50;          // 150 (100 + 50)
int sum2 = sum1 + 250;        // 400 (150 + 250)
int sum3 = sum2 + sum2;       // 800 (400 + 400)
```

Operators are divided into the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators

Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Assignment Operators

Assignment operators are used to assign values to variables. In the example below, we use the **assignment** operator (=) to assign the value **10** to a variable called **x**:

```
int x = 10;
```

The **addition assignment** operator (+=) adds a value to a variable:

```
int x = 10;
x += 5;
```

A list of all assignment operators:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison Operators

Comparison operators are used to compare two values (or variables). This is important in programming, because it helps us to find answers and make decisions.

The return value of a comparison is either 1 or 0, which means **true** (1) or **false** (0). These values are known as **Boolean values**.

In the following example, we use the **greater than** operator (>) to find out if 5 is greater than 3:

```
int x = 5;
int y = 3;
cout << (x > y);          // returns 1 (true) because 5 is greater than 3
```

A list of all comparison operators:

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical Operators

As with comparison operators, you can also test for true (1) or false (0) values with logical operators. Logical operators are used to determine the logic between variables or values:

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Math

C++ has many functions that allows you to perform mathematical tasks on numbers.

Max and min

The max(*x*,*y*) function can be used to find the highest value of *x* and *y*:

```
cout << max(5, 10);
```

And the min(*x*,*y*) function can be used to find the lowest value of *x* and *y*:

```
cout << min(5, 10);
```

<cmath> Library

Other functions, such as sqrt (square root), round (rounds a number) and log (natural logarithm), can be found in the <cmath> header file:

```
// Include the cmath library
#include <cmath>

cout << sqrt(64);
cout << round(2.6);
cout << log(2);
```