C++ User Input

You have already learned that `cout` is used to output (print) values. Now we will use `cin` to get user input.

`cin` is a predefined variable that reads data from the keyboard with the extraction operator (`>>`).

In the following example, the user can input a number, which is stored in the variable `x`. Then we print the value of `x`:

Example

```
int x;
cout << "Type a number: "; // Type a number and press enter
cin >> x; // Get user input from the keyboard
cout << "Your number is: " << x; // Display the input value
```

Good To Know

`cout` is pronounced "see-out". Used for **output**, and uses the insertion operator (`<<`)

`cin` is pronounced "see-in". Used for **input**, and uses the extraction operator (`>>`)

Creating a Simple Calculator

In this example, the user must input two numbers. Then we print the sum by calculating (adding) the two numbers:

Example

```
int x, y;
int sum;
cout << "Type a number: ";
cin >> x;
cout << "Type another number: ";
cin >> y;
sum = x + y;
cout << "Sum is: " << sum;
```

C++ Data Types

As explained in the Variables chapter, a variable in C++ must be a specified data type:

Example

```
int myNum = 5;               // Integer (whole number)
float myFloatNum = 5.99;     // Floating point number
double myDoubleNum = 9.98;   // Floating point number
char myLetter = 'D';         // Character
bool myBoolean = true;       // Boolean
string myText = "Hello";     // String
```

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

Exercise:

Add the correct data type for the following variables:

```
        myNum = 9;
        myDoubleNum = 8.99;
        myLetter = 'A';
        myBool = false;
        myText = "Hello World";
```

C++ Numeric Data Types

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

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

float

```
float myNum = 5.75;
cout << myNum;
```

double

```
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:

Example

```
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`.

Example

```
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':

Example

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

String Types

The `string` type is used to store a sequence of characters (text). This is not a built-in type, but it behaves like one in its most basic usage. String values must be surrounded by double quotes:

Example

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

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

Example

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

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

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

C++ Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + **operator** to add together two values:

Example

```
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:

Example

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

C++ divides the operators 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**:

Example

```
int x = 10;
```

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

Example

```
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**, and you will learn more about them in the Booleans and If..Else chapter.

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

Example

```
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)

C++ 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*:

Example

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

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

Example

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

C++ <cmath> Header

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

Example

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

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

Other Math Functions

A list of other popular Math functions (from the `<cmath>` library) can be found in the table below:

Function	Description
abs(x)	Returns the absolute value of x
acos(x)	Returns the arccosine of x
asin(x)	Returns the arcsine of x
atan(x)	Returns the arctangent of x
cbrt(x)	Returns the cube root of x
ceil(x)	Returns the value of x rounded up to its nearest integer
cos(x)	Returns the cosine of x
cosh(x)	Returns the hyperbolic cosine of x

exp(x)	Returns the value of E^x
expm1(x)	Returns e^x -1
fabs(x)	Returns the absolute value of a floating x
fdim(x, y)	Returns the positive difference between x and y
floor(x)	Returns the value of x rounded down to its nearest integer
hypot(x, y)	Returns sqrt(x^2 +y^2) without intermediate overflow or underflow
fma(x, y, z)	Returns x*y+z without losing precision
fmax(x, y)	Returns the highest value of a floating x and y
fmin(x, y)	Returns the lowest value of a floating x and y
fmod(x, y)	Returns the floating point remainder of x/y
pow(x, y)	Returns the value of x to the power of y
sin(x)	Returns the sine of x (x is in radians)
sinh(x)	Returns the hyperbolic sine of a double value
tan(x)	Returns the tangent of an angle
tanh(x)	Returns the hyperbolic tangent of a double value

Exercise:

Use the correct function to print the highest value of x and y.

```
int x = 5;
int y = 10;
cout << ____(x, y);
```

C++ Booleans

Very often, in programming, you will need a data type that can only have one of two values, like:

- YES / NO
- ON / OFF
- TRUE / FALSE

For this, C++ has a `bool` data type, which can take the values `true` (1) or `false` (0).

Boolean Values

A boolean variable is declared with the `bool` keyword and can only take the values `true` or `false`:

Example

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

From the example above, you can read that a `true` value returns `1`, and `false` returns `0`.

However, it is more common to return a boolean value by **comparing** values and variables (see next page).

C++ If ... Else

C++ Conditions and If Statements

You already know that C++ supports the usual logical conditions from mathematics:

- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`
- Equal to `a == b`
- Not Equal to: `a != b`

You can use these conditions to perform different actions for different decisions.

C++ has the following conditional statements:

- Use `if` to specify a block of code to be executed, if a specified condition is true
- Use `else` to specify a block of code to be executed, if the same condition is false
- Use `else if` to specify a new condition to test, if the first condition is false
- Use `switch` to specify many alternative blocks of code to be executed

The if Statement

Use the `if` statement to specify a block of C++ code to be executed if a condition is `true`.

Syntax

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

Note that `if` is in lowercase letters. Uppercase letters (If or IF) will generate an error.

In the example below, we test two values to find out if 20 is greater than 18. If the condition is `true`, print some text:

Example

```
if (20 > 18) {
  cout << "20 is greater than 18";
}
```

We can also test variables:

Example

```
int x = 20;
int y = 18;
if (x > y) {
  cout << "x is greater than y";
}
```

Example explained

In the example above we use two variables, **x** and **y**, to test whether x is greater than y (using the `>` operator). As x is 20, and y is 18, and we know that 20 is greater than 18, we print to the screen that "x is greater than y".

Exercise:

Print "Hello World" if `x` is **greater than** `y`.

```
int x = 50;
int y = 10;
      (x      y) {
  cout << "Hello World";
}
```