# C Introduction

❮ PreviousNext ❯

## What is C?

C is a general-purpose programming language created by Dennis Ritchie at the Bell Laboratories in 1972.

It is a very popular language, despite being old.

C is strongly associated with UNIX, as it was developed to write the UNIX operating system.

## Why Learn C?

- It is one of the most popular programming language in the world
- If you know C, you will have no problem learning other popular programming languages such as Java, Python, C++, C#, etc, as the syntax is similar
- C is very fast, compared to other programming languages, like Java and Python
- C is very versatile; it can be used in both applications and technologies

## Difference between C and C++

- C++ was developed as an extension of C, and both languages have almost the same syntax
- The main difference between C and C++ is that C++ support classes and objects, while C does not

## Get Started

This tutorial will teach you the basics of C.

It is not necessary to have any prior programming experience.

## Get Started With C

To start using C, you need two things:

- A text editor, like Notepad, to write C code
- A compiler, like GCC, to translate the C code into a language that the computer will understand

There are many text editors and compilers to choose from. In this tutorial, we will use an ***IDE*** (see below).

## C Install IDE

An IDE (Integrated Development Environment) is used to edit AND compile the code.

Popular IDE's include Code::Blocks, Eclipse, and Visual Studio. These are all free, and they can be used to both edit and debug C code.

**Note:** Web-based IDE's can work as well, but functionality is limited.

We will use **Code::Blocks** in our tutorial, which we believe is a good place to start.

You can find the latest version of Codeblocks at http://www.codeblocks.org/. Download the `mingw-setup.exe` file, which will install the text editor with a compiler.

## C Quickstart

Let's create our first C file.

Open Codeblocks and go to **File > New > Empty File**.

Write the following C code and save the file as `myfirstprogram.c` (**File > Save File as**):

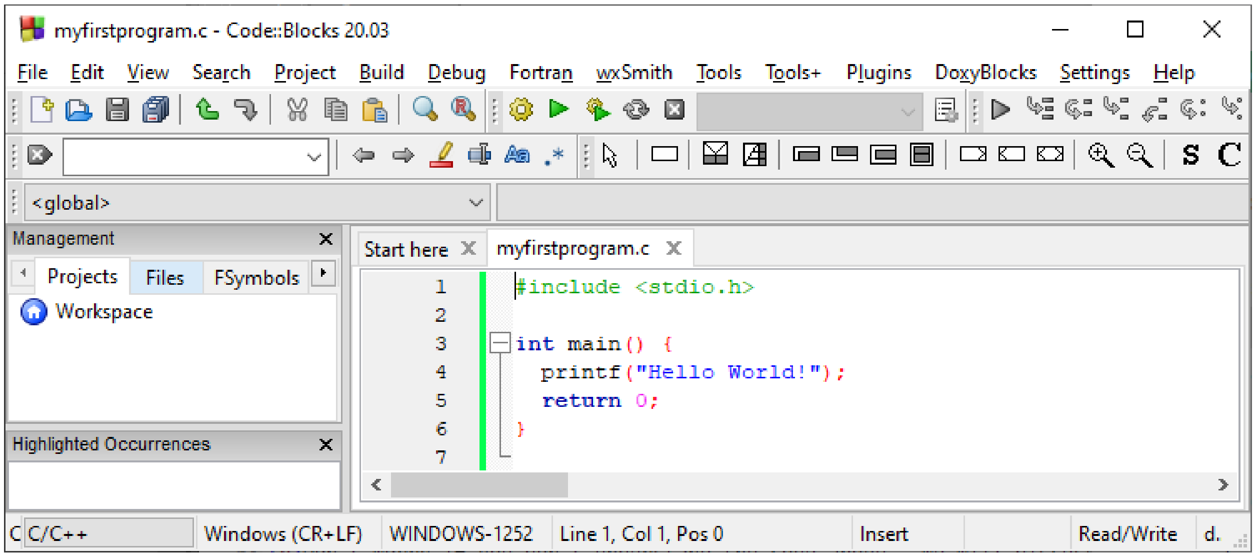
## *myfirstprogram.c*

```
#include <stdio.h>

int main() {
  printf("Hello World!");
  return 0;
}
```

Don't worry if you don't understand the code above - we will discuss it in detail in later chapters. For now, focus on how to run the code.

In Codeblocks, it should look like this:



```
#include <stdio.h>

int main() {
  printf("Hello World!");
  return 0;
}
```

Then, go to **Build > Build and Run** to run (execute) the program. The result will look something to this:

```
Hello World!
Process returned 0 (0x0) execution time : 0.011 s
Press any key to continue.
```

**Congratulations**! You have now written and executed your first C program.

## Learning C At W3Schools

When learning C at W3Schools.com, you can use our "Try it Yourself" tool, which shows both the code and the result. This will make it easier for you to understand every part as we move forward:

### myfirstprogram.c

Code:

```
#include <stdio.h>

int main() {
  printf("Hello World!");
  return 0;
}
```

Result:

```
Hello World!
```

## Syntax

You have already seen the following code a couple of times in the first chapters. Let's break it down to understand it better:

### Example

```
#include <stdio.h>

int main() {
  printf("Hello World!");
  return 0;
}
```

Try it Yourself »

## Example explained

**Line 1:** `#include <stdio.h>` is a **header file library** that lets us work with input and output functions, such as `printf()` (used in line 4). Header files add functionality to C programs.

Don't worry if you don't understand how `#include <stdio.h>` works. Just think of it as something that (almost) always appears in your program.

**Line 2:** A blank line. C ignores white space. But we use it to make the code more readable.

**Line 3:** Another thing that always appear in a C program, is `main()`. This is called a **function**. Any code inside its curly brackets `{}` will be executed.

**Line 4:** `printf()` is a **function** used to output/print text to the screen. In our example it will output "Hello World!".

**Note that:** Every C statement ends with a semicolon `;`

**Note:** The body of `int main()` could also been written as:
`int main(){printf("Hello World!");return 0;}`

**Remember:** The compiler ignores white spaces. However, multiple lines makes the code more readable.

**Line 5:** `return 0` ends the `main()` function.

**Line 6:** Do not forget to add the closing curly bracket `}` to actually end the main function.

## Output (Print Text)

To output values or print text in C, you can use the `printf()` function:

### Example

```
#include <stdio.h>

int main() {
  printf("Hello World!");
  return 0;
}
```

Try it Yourself »

You can use as many `printf()` functions as you want. **However**, note that it does not insert a new line at the end of the output:

### Example

```
#include <stdio.h>

int main() {
  printf("Hello World!");
  printf("I am learning C.");
  return 0;
}
```

Try it Yourself »

# C New Lines

❮ PreviousNext ❯

## New Lines

To insert a new line, you can use the `\n` character:

### Example

```
#include <stdio.h>

int main() {
  printf("Hello World!\n");
  printf("I am learning C.");
  return 0;
}
```

Try it Yourself »

You can also output multiple lines with a single `printf()` function. However, this could make the code harder to read:

## Example

```
#include <stdio.h>

int main() {
  printf("Hello World!\nI am learning C.\nAnd it is awesome!");
  return 0;
}
```

Try it Yourself »

**Tip:** Two `\n` characters after each other will create a blank line:

## Example

```
#include <stdio.h>

int main() {
  printf("Hello World!\n\n");
  printf("I am learning C.");
  return 0;
}
```

Try it Yourself »

### *What is `\n` exactly?*

The newline character (`\n`) is called an **escape sequence**, and it forces the cursor to change its position to the beginning of the next line on the screen. This results in a new line.

Examples of other valid escape sequences are:

| Escape Sequence | Description |
|---|---|
| \t | Creates a horizontal tab |

| | |
|---|---|
| \\ | Inserts a backslash character (\) |
| \" | Inserts a double quote character |

## Comments in C

Comments can be used to explain code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

Comments can be **singled-lined** or **multi-lined**.

## Single-line Comments

Single-line comments start with two forward slashes (`//`).

Any text between `//` and the end of the line is ignored by the compiler (will not be executed).

This example uses a single-line comment before a line of code:

### Example

```
// This is a comment
printf("Hello World!");
```

Try it Yourself »

This example uses a single-line comment at the end of a line of code:

### Example

```
printf("Hello World!"); // This is a comment
```

Try it Yourself »

## C Multi-line Comments

Multi-line comments start with `/*` and ends with `*/`.

Any text between `/*` and `*/` will be ignored by the compiler:

### Example

```
/* The code below will print the words Hello World!
to the screen, and it is amazing */
printf("Hello World!");
```

Try it Yourself »

# C Variables

❮ PreviousNext ❯

Variables are containers for storing data values, like numbers and characters.

In C, there are different **types** of variables (defined with different keywords), for example:

- `int` - stores integers (whole numbers), without decimals, such as `123` or `-123`
- `float` - stores floating point numbers, with decimals, such as `19.99` or `-19.99`
- `char` - stores single characters, such as `'a'` or `'B'`. Char values are surrounded by **single quotes**

## Declaring (Creating) Variables

To create a variable, specify the **type** and assign it a **value**:

## Syntax

```
type variableName = value;
```

Where *type* is one of C types (such as `int`), and *variableName* is the name of the variable (such as **x** or **myName**). The **equal sign** is used to assign a value to the variable.

So, to create a variable that should **store a number**, look at the following example:

## Example

Create a variable called **myNum** of type `int` and assign the value **15** to it:

```
int myNum = 15;
```

You can also declare a variable without assigning the value, and assign the value later:

## Example

```
// Declare a variable
int myNum;

// Assign a value to the variable
myNum = 15;
```

# Output Variables

You learned from the output chapter that you can output values/print text with the `printf()` function:

## Example

```
printf("Hello World!");
```

Try it Yourself »

In many other programming languages (like Python, Java, and C++), you would normally use a **print function** to display the value of a variable. However, this is not possible in C:

### Example

```
int myNum = 15;
printf(myNum);  // Nothing happens
```

Try it Yourself »

To output variables in C, you must get familiar with something called "format specifiers".

## Format Specifiers

Format specifiers are used together with the `printf()` function to tell the compiler what type of data the variable is storing. It is basically a placeholder for the variable value.

A format specifier starts with a percentage sign `%`, followed by a character.

For example, to output the value of an `int` variable, you must use the format specifier `%d` or `%i` surrounded by double quotes, inside the `printf()` function:

### Example

```
int myNum = 15;
printf("%d", myNum);  // Outputs 15
```

Try it Yourself »

To print other types, use `%c` for `char` and `%f` for `float`:

### Example

```
// Create variables
int myNum = 15;            // Integer (whole number)
float myFloatNum = 5.99;   // Floating point number
char myLetter = 'D';       // Character
```

```
// Print variables
printf("%d\n", myNum);
printf("%f\n", myFloatNum);
printf("%c\n", myLetter);
```

Try it Yourself »

To combine both text and a variable, separate them with a comma inside the `printf()` function:

## Example

```
int myNum = 15;
printf("My favorite number is: %d", myNum);
```

Try it Yourself »

To print different types in a single `printf()` function, you can use the following:

## Example

```
int myNum = 15;
char myLetter = 'D';
printf("My number is %d and my letter is %c", myNum, myLetter);
```

Try it Yourself »

You will learn more about Data Types in the next chapter.

## Change Variable Values

**Note:** If you assign a new value to an existing variable, it will overwrite the previous value:

## Example

```
int myNum = 15;  // myNum is 15
myNum = 10;  // Now myNum is 10
```

Try it Yourself »

You can also assign the value of one variable to another:

### Example

```c
int myNum = 15;

int myOtherNum = 23;

// Assign the value of myOtherNum (23) to myNum
myNum = myOtherNum;

// myNum is now 23, instead of 15
printf("%d", myNum);
```

Try it Yourself »

Or copy values to empty variables:

### Example

```c
// Create a variable and assign the value 15 to it
int myNum = 15;

// Declare a variable without assigning it a value
int myOtherNum;

// Assign the value of myNum to myOtherNum
myOtherNum = myNum;

// myOtherNum now has 15 as a value
printf("%d", myOtherNum);
```

Try it Yourself »

## Add Variables Together

To add a variable to another variable, you can use the `+` operator:

### Example

```c
int x = 5;
int y = 6;
int sum = x + y;
printf("%d", sum);
```

Try it Yourself »

## Declare Multiple Variables

To declare more than one variable of the same type, use a **comma-separated** list:

### Example

```c
int x = 5, y = 6, z = 50;
printf("%d", x + y + z);
```

Try it Yourself »

You can also assign the **same value** to multiple variables of the same type:

### Example

```c
int x, y, z;
x = y = z = 50;
printf("%d", x + y + z);
```

Try it Yourself »

## C Variable Names

All C **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

**Note:** It is recommended to use descriptive names in order to create understandable and maintainable code:

### Example

```
// Good
int minutesPerHour = 60;

// OK, but not so easy to understand what m actually is
int m = 60;
```

The **general rules** for naming variables are:

- Names can contain letters, digits and underscores
- Names must begin with a letter or an underscore (_)
- Names are case sensitive (`myVar` and `myvar` are different variables)
- Names cannot contain whitespaces or special characters like !, #, %, etc.
- Reserved words (such as `int`) cannot be used as names

## Real-Life Example

Often in our examples, we simplify variable names to match their data type (myInt or myNum for `int` types, myChar for `char` types etc). This is done to avoid confusion.

However, if you want a real-life example on how variables can be used, take a look at the following, where we have made a program that stores different data of a college student:

### Example

```
// Student data
int studentID = 15;
int studentAge = 23;
float studentFee = 75.25;
char studentGrade = 'B';
```

```
// Print variables
printf("Student id: %d\n", studentID);
printf("Student age: %d\n", studentAge);
printf("Student fee: %f\n", studentFee);
printf("Student grade: %c", studentGrade);
```

Try it Yourself »

## C Exercises

## Exercise:

Create a variable named `myNum` and assign the value `50` to it.

# C Data Types

❮ PreviousNext ❯

## Data Types

As explained in the Variables chapter, a variable in C must be a specified **data type**, and you must use a **format specifier** inside the `printf()` function to display it:

### Example

```
// Create variables
int myNum = 5;             // Integer (whole number)
float myFloatNum = 5.99;   // Floating point number
char myLetter = 'D';       // Character

// Print variables
```

```
printf("%d\n", myNum);
printf("%f\n", myFloatNum);
printf("%c\n", myLetter);
```

Try it Yourself »

## Basic Data Types

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

In this tutorial, we will focus on the most basic ones:

| **Data Type** | **Size** | **Description** |
|---|---|---|
| `int` | 2 or 4 bytes | Stores whole numbers, without decimals |
| `float` | 4 bytes | Stores fractional numbers, containing one or more deci |
| `double` | 8 bytes | Stores fractional numbers, containing one or more deci |
| `char` | 1 byte | Stores a single character/letter/number, or ASCII values |

## Basic Format Specifiers

There are different format specifiers for each data type. Here are some of them:

| Format Specifier | Data Type |
|---|---|
| %d or %i | int |
| %f | float |
| %lf | double |
| %c | char |
| %s | Used for strings **(text)**, which you will learn more about |

## Set Decimal Precision

You have probably already noticed that if you print a floating point number, the output will show many digits after the decimal point:

### Example

```
float myFloatNum = 3.5;
double myDoubleNum = 19.99;

printf("%f\n", myFloatNum); // Outputs 3.500000
printf("%lf", myDoubleNum); // Outputs 19.990000
```

Try it Yourself »

If you want to remove the extra zeros (set decimal precision), you can use a dot (.) followed by a number that specifies how many digits that should be shown after the decimal point:

## Example

```
float myFloatNum = 3.5;

printf("%f\n", myFloatNum); // Default will show 6 digits after the
decimal point
printf("%.1f\n", myFloatNum); // Only show 1 digit
printf("%.2f\n", myFloatNum); // Only show 2 digits
printf("%.4f", myFloatNum);   // Only show 4 digits
```

Try it Yourself »

# C Exercises

## Exercise:

Add the correct data type for the following variables:

```
[      ] myNum = 5;
[      ] myFloatNum = 5.99;
[      ] myLetter = 'D';
```