What is C?

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

It is a very large popular programming 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.

SYNTAX:

Comment in C

C Variables

C Data Types

C Constants

Operators

C If…Else

C Switch

C While Loop

C For Loop

C Break and Continue

C Arrays

C Functions

C Function Parameters

C Function Declaration and Definition

RECURSION EXAMPLE

C Files

File Handling

Create a File

C Write To Files

C Read Files

**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;

}

Example explained

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

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;

}

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>

Intmain()

{

Printf(“Hello World!”);

Printf(“I am learning C.;”)

Return 0;

}

NEW LINES

To insert a new line, you can use the \ncharacter :

Example

#include <stdio.h>

int main()

{

Printf(“Hello World!\n”);

Printf(“I am learning C.”);

Return 0;

}

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

Example

#include <stdio.h>

Intmain()

{

Printf(“Hello World!\n I am learning C.\n And it is awesome!”);

Return 0:

}

**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;

}

intmyNum=15; // Integer (Whole number)

floatmyFloatNum=5.99; //Floating point number

char myLetter=’D’;

//Print variables

Printf(“%d\n”,myNum);

Printf(“%f\n”,myFloatNum);

Printf(“%c\n”,myLetter);

}

**C DATA TYPES**

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Size** | **Description** |
| 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. |
| char | 1 byte | Stores a single character/letter/number, or ASCII values |

**FORMAT SPECIFIER**

|  |  |  |
| --- | --- | --- |
| **Format Specifier** | **Data Type** | **Description** |
| %d or %i | Int | Used for signed decimal integer |
| %f | Float | Used for floating point numbers |
| %lf | Double | Used to print a double value |
| %c | Char | Used for character |
| %s | Char | Used for String |

**Decimal Precision**

floatmyFloatNum=3.5;

doublemyDoubleNum=19.99;

printf(“%f\n”,myFloatNum); //Outputs 3.500000

printf(“%if\n”,myDoubleNum); //Outputs 19.990000

**C OPERATORS**

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 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 operator
* Assignment operator
* Comparison operator
* Logical operator
* Bitwise operator

**Arithmetic Operator:**

Arithmetic operators perform simple arithmetic operations like addition, subtraction, multiplication, division etc.,

|  |  |  |
| --- | --- | --- |
| Operator | Name | Description |
| + | Addition | Adds together two values |
| - | Subtraction | Subtracts one value from another |
| \* | Multiplication | Multiplies two values |
| / | Division | Divides one value by another |
| % | Modulus | Returns the division remainder |
| ++ | Increment | Increase the value of a variable by 1 |
| -- | Decrement | Decrease the value of a variable by 1 |

Assignment Operators

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

|  |  |  |
| --- | --- | --- |
| 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](https://www.w3schools.com/c/c_booleans.php) and [If..Else](https://www.w3schools.com/c/c_conditions.php) chapter.

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;  
printf("%d", x > y);

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

|  |  |  |  |
| --- | --- | --- | --- |
| **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 statement is true | X<5 || X<4 |
| ! | Logical not | Reverse the result, returns false if the result is true and vice versa | !(X<5 && X<10) |

# **C If ... Else**

Conditions and If Statements

You have already learned 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

int time = 20;  
if (time < 18)

{  
 printf("Good day.");  
}

else

{  
 printf("Good evening.");  
}  
// Outputs "Good evening."

int time = 22;  
if (time < 10)

{  
 printf("Good morning.");  
}

else if (time < 20)

{  
  printf("Good day.");  
}

 else

{  
 printf("Good evening.");  
}  
// Outputs "Good evening”