

---

# C Programming Training Notes: C Home to C Functions (W3Schools)

## 1. C Home & Getting Started

### C Home (Introduction)

- **What is C?** C is a **general-purpose programming language** created by Dennis Ritchie in 1972. It is highly influential, powerful, and forms the basis for many modern languages (C++, C#, Java, JavaScript, Python, etc.).
- **Why C?** It is used for developing **Operating Systems** (like Windows, Linux), databases, compilers, and hardware devices. It is fast and efficient.

### C Get Started (Setup)

- To start writing C, you need a **Text Editor** and a **Compiler** (like GCC). A common development environment is an **Integrated Development Environment (IDE)** such as Code::Blocks or Visual Studio Code.
- **How to Run a C Program:**
  1. Write the code (e.g., `myfirstprog.c`).
  2. Compile the code using a command like `gcc myfirstprog.c`.
  3. Execute the output file (`./a.out` on Linux/macOS, or `a.exe` on Windows).

## 2. Basic Syntax and Structure

### C Program Structure (The Hello World Example)

The structure of a basic C program includes:

Component	Example	Description
<b>Header File</b>	<code>#include &lt;stdio.h&gt;</code>	Includes the standard I/O library functions.
<b>Main Function</b>	<code>int main() { ... }</code>	The entry point for all C programs. Code inside the curly braces {} is executed.
<b>Output</b>	<code>printf("Hello World!");</code>	A function to output text to the screen.

Component	Example	Description
<b>Return</b>	<code>return 0;</code>	Indicates that the program executed successfully.

## C

```
#include <stdio.h>

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

## C Syntax Rules

- **Statements:** Every statement must end with a **semicolon** ( ; ).
- **Case Sensitivity:** C is **case-sensitive**. myVar is different from myvar.
- **Code Blocks:** Code blocks are defined by **curly braces** ( {} ).

## C Output (`printf()`)

- The primary function for outputting text is `printf()`.
- To add a new line, use the **newline character**: \n.
- Example: `printf("Line 1.\nLine 2.");`

## C Comments

- **Single-line comments:** Start with two forward slashes (//).
- **Multi-line comments:** Start with /\* and end with \*/.

## 3. Variables, Data Types, and Operators

### C Variables

A variable is a container for storing data values.

Type	Syntax	Example
<b>Integer</b>	<code>int variableName = value;</code>	<code>int myNum = 15;</code>
<b>Floating Point</b>	<code>float variableName = value;</code>	<code>float myFloat = 5.99;</code>
<b>Character</b>	<code>char variableName = 'A';</code>	<code>char myLetter = 'D';</code>

## Format Specifiers

Used inside `printf()` to display the value of a variable:

Specifier	Data Type	Example
<code>%d</code> or <code>%i</code>	<code>int</code> (Integer)	<code>printf("%d", myNum);</code>
<code>%f</code>	<code>float</code> (Floating Point)	<code>printf("%f", myFloat);</code>
<code>%c</code>	<code>char</code> (Character)	<code>printf("%c", myLetter);</code>

## C Data Types

Type	Size	Description
<code>int</code>	2 or 4 bytes	Stores whole numbers (integers).
<code>float</code>	4 bytes	Stores fractional numbers (decimals), 6-7 decimal digits.
<code>double</code>	8 bytes	Stores fractional numbers, 15 decimal digits.
<code>char</code>	1 byte	Stores a single character/letter.

## C Operators

Operators are used to perform operations on variables and values.

Type	Operator	Description	Example
<b>Arithmetic</b>	<code>+, -, *, /, %</code>	Addition, Subtraction, Multiplication, Division, Modulus.	<code>x = 5 + 3;</code>

Type	Operator	Description	Example
Assignment	=, +=, -=	Assigns a value; combines operation and assignment.	x += 3; (Same as x = x + 3;)
Comparison	==, !=, >, <, >=, <=	Compares two values; returns 1 (True) or 0 (False).	x == 5
Logical	&&, ^		, !

## 4. Control Flow (Conditions and Loops)

### C If...Else

Used to execute different code blocks based on a **Boolean** condition.

```
C
// Basic If
if (condition) {
    // block of code to be executed if the condition is True
}

// If...Else
if (condition) {
    // True block
} else {
    // False block
}

// If...Else If...Else
if (condition1) {
    // Executes if condition1 is true
} else if (condition2) {
    // Executes if condition1 is false AND condition2 is true
} else {
    // Executes if all conditions are false
}
```

### C Switch

Used to select one of many code blocks to be executed. It evaluates an expression once.

```
C
switch (expression) {
    case 1:
        // code block 1
        break; // Stops the execution inside the switch block
    case 2:
```

```

    // code block 2
    break;
default:
    // code block that runs if no case matches
}

```

## C Loops (While and For)

Loops execute a block of code repeatedly as long as a specified condition is met.

### While Loop

C

```

int i = 0;
while (i < 5) {
    printf("%d\n", i);
    i++; // Increment the counter
}

```

### For Loop

The **For Loop** is typically used when you know exactly how many times you want to loop.

C

```

for (statement 1; statement 2; statement 3) {
    // code block to be executed
}

// Statement 1: executed once before the loop starts (initialization).
// Statement 2: defines the condition for running the loop.
// Statement 3: executed every time after the loop code has been executed
// (increment/decrement).

for (int i = 0; i < 5; i++) {
    printf("%d\n", i);
}

```

### Break and Continue

- **break:** Jumps out of a loop or a `switch` statement entirely.
- **continue:** Skips the current iteration of the loop and moves to the next one.

## 5. Arrays, Strings, and Pointers

### C Arrays

Arrays are used to store multiple values of the **same data type** in a single variable.

- Declaration: Specify the type, name, and size.

```
int myNumbers[4];
```

- Initialization:

```
int myNumbers[] = {25, 50, 75, 100};
```

- Access: Array indices start at 0.

```
printf("%d", myNumbers[0]); // Output: 25
```

## C Strings

A string is a sequence of characters stored as an array of characters.

- Declaration:

```
char greetings[] = "Hello";
```

- The string is terminated by a special character: the **null character** (\0).
- **String Functions:** Must include <string.h>.
  - `strlen(string)`: Returns the length of the string.
  - `strcpy(destination, source)`: Copies a string.
  - `strcat(string1, string2)`: Concatenates (joins) two strings.
  - `strcmp(string1, string2)`: Compares two strings (returns 0 if equal).

## C Pointers

A **pointer** is a variable that stores the **memory address** of another variable.

Operator	Name	Description
&	<b>Address-of Operator</b>	Returns the memory address of a variable.
*	<b>Dereference Operator</b>	Returns the value stored at the memory address.

### C

```
int myAge = 43;      // Variable
int* ptr = &myAge;  // Pointer variable stores the address of myAge

// Output the value (43)
printf("%d\n", myAge);

// Output the address (e.g., 0x7ffe5367e044)
printf("%p\n", &myAge); // Using %p format specifier

// Output the value using the pointer (Dereference: 43)
printf("%d\n", *ptr);
```

## 6. C Functions

### What are C Functions?

A function is a block of code that only runs when it is called. Functions are used to:

1. **Reuse Code:** Define the code once and use it multiple times.
2. **Modularize Code:** Break down a large program into smaller, manageable parts.

## Function Declaration and Definition

Part	Description
<b>Declaration</b> (or Prototype)	Tells the compiler the function's name, return type, and parameters. (Optional if defined before <code>main</code> ).
<b>Definition</b>	Contains the actual body of the function.

### Example:

```
C
// 1. Function Declaration (outside main)
void myFunction();

// 2. Main function (caller)
int main() {
    myFunction(); // Call the function
    return 0;
}

// 3. Function Definition
void myFunction() {
    printf("I just got executed!");
}
```

## Function Parameters

Information can be passed to functions as **parameters** (or arguments).

```
C
// Function Declaration with parameters
void printName(char name[]);

// Function Definition
void printName(char name[]) {
    printf("Hello %s\n", name);
}

// Calling the function
int main() {
    printName("Liam"); // Output: Hello Liam
    return 0;
}
```

## Return Values

To return a value, use a data type other than `void` and the `return` keyword.

```
C
// Function returns an integer (int)
int mySum(int x, int y) {
    return x + y;
}

int main() {
    int result = mySum(5, 3);
    printf("%d", result); // Output: 8
    return 0;
}
```

# C Programming Training Notes: C Home to C Functions (W3Schools)

## 1. C Fundamentals and Basic Syntax

### C Home (Introduction)

- **What is C?** C is a **general-purpose programming language** created by Dennis Ritchie in 1972. It is highly influential, powerful, and forms the basis for many modern languages (C++, C#, Java, JavaScript, Python, etc.).
- **Why C?** It is used for developing **Operating Systems** (like Windows, Linux), databases, compilers, and hardware devices. It is fast and efficient.

### C Get Started (Setup)

- To start writing C, you need a **Text Editor** and a **Compiler** (like GCC). A common development environment is an **Integrated Development Environment (IDE)** such as Code::Blocks or Visual Studio Code.
- **How to Run a C Program:**
  1. Write the code (e.g., `myfirstprog.c`).
  2. Compile the code using a command like `gcc myfirstprog.c`.
  3. Execute the output file (`./a.out` on Linux/macOS, or `a.exe` on Windows).

---

## 2. Basic Syntax and Structure

### C Program Structure (The `Hello World` Example)

The structure of a basic C program includes:

Component	Example	Description
<b>Header File</b>	#include <stdio.h>	Includes the standard I/O library functions.
<b>Main Function</b>	int main() { ... }	The entry point for all C programs. Code inside the curly braces {} is executed.
<b>Output</b>	printf("Hello World!");	A function to output text to the screen.
<b>Return</b>	return 0;	Indicates that the program executed successfully.

C

```
#include <stdio.h>

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

## C Syntax Rules

- **Statements:** Every statement must end with a **semicolon** (;).
- **Case Sensitivity:** C is **case-sensitive**. myVar is different from myvar.
- **Code Blocks:** Code blocks are defined by **curly braces** ({}).

## C Output (`printf()`)

- The primary function for outputting text is `printf()`.
- To add a new line, use the **newline character**: \n.
- Example: `printf("Line 1.\nLine 2.");`

## C Comments

- **Single-line comments:** Start with two forward slashes (//).
- **Multi-line comments:** Start with /\* and end with \*/.

## 3. Variables, Data Types, and Operators

### C Variables: The Lifecycle

Understanding the difference between Declaration, Initialization, and Assignment is essential.

Action	Description	Examples
Declaration	Reserving memory by giving a <b>type</b> and a <b>name</b> . The variable has no known value yet.	<pre>int a; char k; char name[30];</pre>
Initialization	<b>Declaration + giving the variable its first value</b> at the same time.	<pre>int a = 10; char k = 'M'; char name[30] = "Mahalakshmy";</pre>
Assignment	Changing the value of an <b>already declared</b> variable.	<pre>a = 50; k = 'L';</pre>

*Note: While C does not have a native string type, it is common to use `char[]` to represent strings, as shown above.*

## C Data Types

Type	Size	Description
<code>int</code>	2 or 4 bytes	Stores whole numbers (integers).
<code>float</code>	4 bytes	Stores fractional numbers (decimals), 6-7 decimal digits.
<code>double</code>	8 bytes	Stores fractional numbers, 15 decimal digits.

Type	Size	Description
char	1 byte	Stores a single character/letter.

## C Operators

Operators are used to perform operations on variables and values.

Type	Operator	Description	Example
Arithmetic	+, -, *, /, %	Addition, Subtraction, Multiplication, Division, Modulus.	x = 5 + 3;
Assignment	=, +=, -=	Assigns a value; combines operation and assignment.	x += 3; (Same as x = x + 3;)
Comparison	==, !=, >, <, >=, <=	Compares two values; returns 1 (True) or 0 (False).	x == 5
Logical	&&, ^		, !

---

## 4. Control Flow (Conditions and Loops)

### C If...Else

Used to execute different code blocks based on a **Boolean** condition.

```
C
// If...Else If...Else Structure
if (condition1) {
    // Executes if condition1 is true
} else if (condition2) {
    // Executes if condition1 is false AND condition2 is true
} else {
    // Executes if all conditions are false
}
```

### C Switch

Used to select one of many code blocks to be executed based on a single expression.

```
C
switch (expression) {
    case 1:
        // code block 1
        break; // Important: Stops the execution inside the switch block
    case 2:
        // code block 2
        break;
    default:
        // code block that runs if no case matches
}
```

## C Loops (While and For)

### While Loop

```
C
int i = 0;
while (i < 5) {
    printf("%d\n", i);
    i++; // Increment the counter
}
```

### For Loop

The **For Loop** is typically used when you know exactly how many times you want to loop.

```
C
for (int i = 0; i < 5; i++) {
    printf("%d\n", i);
}
// Statement 1: Initialization (int i = 0)
// Statement 2: Condition (i < 5)
// Statement 3: Increment/Decrement (i++)
```

### Break and Continue

- **break**: Jumps out of a loop or a `switch` statement entirely.
- **continue**: Skips the current iteration of the loop and moves to the next one.

---

## 5. Arrays, Strings, and Pointers

### C Arrays

Arrays are used to store multiple values of the **same data type** in a single variable.

- Initialization:

```
int myNumbers[] = {25, 50, 75, 100};
```

- Access: Array indices start at 0.

```
printf("%d", myNumbers[0]); // Output: 25
```

## C Strings

A string is a sequence of characters stored as an array of characters, terminated by the **null character** (\0).

- Declaration:

```
char greetings[] = "Hello";
```

- **String Functions:** Must include <string.h>. Common functions include `strlen()`, `strcpy()`, `strcat()`, and `strcmp()`.

## C Pointers

A **pointer** is a variable that stores the **memory address** of another variable.

Operator	Name	Description
&	<b>Address-of Operator</b>	Returns the memory address of a variable.
*	<b>Dereference Operator</b>	Returns the value stored at the memory address.

C

```
int myAge = 43;      // Variable
int* ptr = &myAge;   // Pointer variable stores the address of myAge

// Output the value using the pointer (Dereference: 43)
printf("%d\n", *ptr);
```

---

## 6. C Functions

### What are C Functions?

A function is a block of code that only runs when it is called. Functions are used for **code reuse** and **modularity**.

### Function Declaration and Definition

Part	Description
<b>Declaration</b> (or Prototype)	Tells the compiler the function's name, return type, and parameters.
<b>Definition</b>	Contains the actual body of the function.

### Example:

```
C
// 1. Declaration
void myFunction();

// 2. Definition
void myFunction() {
    printf("I just got executed!");
}
```

## C Function Parameters or Arguments

Parameters are variables defined in the function signature that accept values when the function is called.

**We pass only the variable name and its value as function(s) or method(s) parameter(s) or argument(s).**

When you call a function, the values you pass are used to **initialize** the local parameter variables within the function's scope.

### Example of Passing Arguments:

```
C
// Function Declaration/Definition with parameters
void printName(char name_arg[], int age_arg) {
    // name_arg and age_arg are the parameter variables
    printf("Hello %s, you are %d\n", name_arg, age_arg);
}

int main() {
    // Variables being passed (arguments)
    char student_name[] = "Mahalakshmy";
    int student_age = 25;

    // Call the function, passing the values
    printName(student_name, student_age);

    // Output: Hello Mahalakshmy, you are 25
    return 0;
}
```

## Return Values

To return a value, specify a data type (e.g., `int`) instead of `void` and use the `return` keyword.

```
C
// Function returns an integer (int)
int mySum(int x, int y) {
    return x + y;
}

int main() {
    int result = mySum(5, 3);
    printf("%d", result); // Output: 8
    return 0;
}
```