

# **HENKEL CSR BOOTCAMP – 2025**

## **C PROGRAMMING**

## Contents

1	Setup & Introduction .....	6
1.1	<b>What is C?</b> .....	6
1.1.1	Why Learn C? .....	6
1.2	<b>Tool Setup</b> .....	6
1.2.1	Install C .....	6
1.2.2	Install IDE .....	6
1.3	<b>First C Program</b> .....	7
1.4	<b>Structure Explanation</b> .....	8
1.4.1	Many printf Functions .....	8
1.4.2	New Lines .....	9
1.4.3	scanf Functions .....	9
1.4.4	Comments in C .....	10
1.5	<b>Quiz to Ask :</b> .....	10
2	Variables & Types .....	11
2.1	<b>Variables Declaration &amp; Format Specifiers</b> .....	11
2.1.1	Declaring (Creating) Variables .....	12
2.1.2	Format Specifiers .....	12
2.2	<b>Data Types</b> .....	14
2.2.1	Basic Data Types .....	14
2.3	<b>Constants</b> .....	16
2.3.1	Type Conversion.....	17
3	Operators Overview.....	20
3.1	Operators .....	20
3.2	Operator Types.....	20
3.2.1	Arithmetic Operators .....	20
3.2.2	Assignment Operators .....	21
3.2.3	Comparison Operators .....	22
3.2.4	Logical Operators .....	23

4. Conditional Statements .....	23
4.1 Conditions and If Statements.....	23
4.1.1 If Statement .....	23
4.1.2 The else Statement.....	24
4.1.3 The else if Statement .....	25
4.2 Ternary Operator .....	25
4.3 Switch Case Statements.....	26
5. Loop Structures.....	31
5.1 While Loop .....	31
5.2 Do/While Loop .....	31
5.3 For Loop .....	32
5.4 Nested Loop .....	33
5.4 Break and Continue.....	34
6. Arrays .....	35
6.1 Arrays.....	35
6.1.1 Access the Elements of an Array.....	35
6.1.2 Change an Array Element.....	36
6.1.3 Loop Through an Array .....	36
6.1.4 Set Array Size .....	36
6.1.4 Multidimensional Arrays .....	37
7. Functions in C .....	41
7.1 Functions .....	41
7.1.1 Predefined Functions .....	41
7.1.2. Create a Function.....	41
7.1.3. Call a Function .....	42
7.2 Functions Parameters .....	42
7.2.1 Parameters and Arguments .....	42
7.2.2 Multiple Parameters .....	43
7.3 Variable Scope in C .....	43

7.3.1 Local Scope .....	43
7.3.2 Global Scope .....	44
7.3.3 Naming Variables .....	45
8. Recursion in C.....	45
9. Introduce the concept of random number generation using rand() .....	50

# **Day 1: Introduction to C**

# 1 Setup & Introduction

## 1.1 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. The main reason for its popularity is because it is a fundamental language in the field of computer science. C is strongly associated with UNIX, as it was developed to write the UNIX operating system.

### 1.1.1 Why Learn C?

- It is one of the most popular programming languages 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
- If you know C, you will understand how computer memory works
- 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

## 1.2 Tool Setup

### 1.2.1 Install C

- If you want to run C on your own computer, 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 the next steps, we will show you how to use an IDE that includes both.

### 1.2.2 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.
- 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.

## 1.3 First C Program

Let's create our first C file.

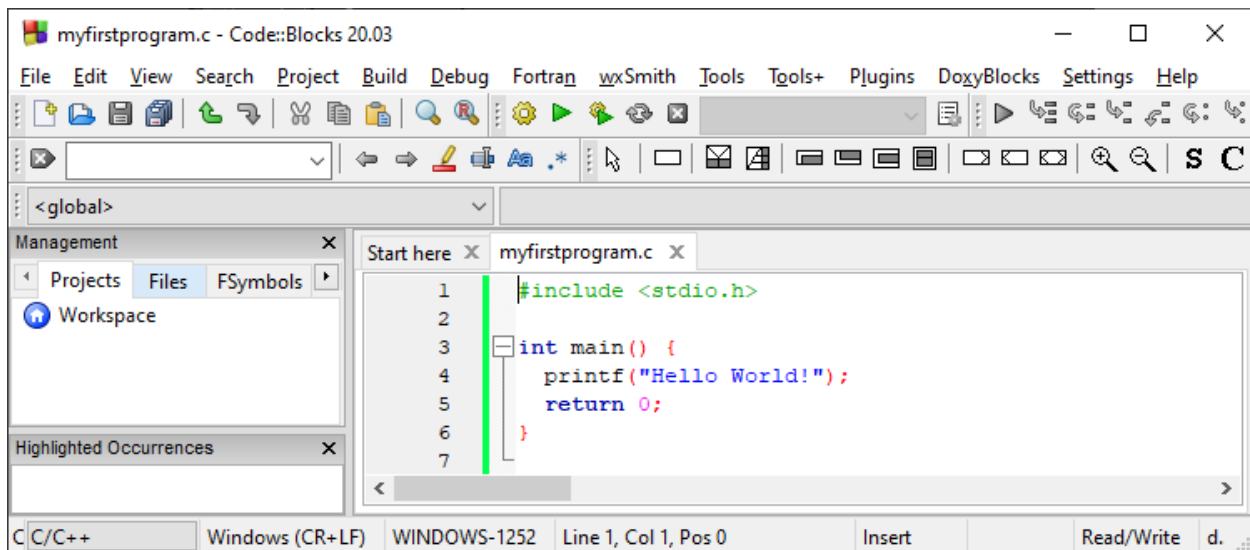
Open Codeblocks and go to **File > New > Empty File**.

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

```
#include <stdio.h>
```

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

In Codeblocks, it should look like this:



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.

## 1.4 Structure Explanation

**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.

### 1.4.1. Many printf Functions

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:

```
#include <stdio.h>
```

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

Output :

Hello World!! am learning C.And it is awesome!



## 1.4.2 New Lines

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

```
#include <stdio.h>

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

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

```
#include <stdio.h>

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

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.

## 1.4.3 scanf Functions

You have already learned that `printf()` is used to output values in C.

To get user input, you can use the `scanf()` function. The `scanf()` function also allow multiple inputs

**Example :**

```
// Create an integer variable that will store the number we get from the user
int myNum;

// Ask the user to type a number
printf("Type a number: \n");

// Get and save the number the user types
```

```
scanf("%d", &myNum);
```

```
// Output the number the user typed  
printf("Your number is: %d", myNum);
```

### 1.4.4 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:

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

Multi-line Comments : Multi-line comments start with /\* and ends with \*/.

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

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

## 1.5 Quiz to Ask :

1. What is the purpose of #include <stdio.h> in a C program?

It defines the main() function

It allows the use of input and output functions, such as printf()

It declares variables

It ends the program

2. **True** or False:

In C, each code statement must end with a semicolon (;).

3. Which function is used to print text in C?

println()

printf()

cout()

echo()

Which function is used to get user input in C?

printf()

scanf()

pgets()

sgets()

5. What is the meaning of comments in C?

To output text

To create text variables

To explain and document code

To print text and numbers with a simple line of code

## 2 Variables & Types

### 2.1 Variables Declaration & Format Specifiers

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'. Characters are surrounded by **single quotes**

### 2.1.1. 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:

Ex : 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:

Ex:

```
// Declare a variable
```

```
int myNum;
```

```
// Assign a value to the variable
```

```
myNum = 15;
```

To output variables in C, you must get familiar with something called "[format specifiers](#)", which you will learn about in the next chapter.

### 2.1.2 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, use the format specifier %d surrounded by double quotes (""), inside the printf() function:

Ex :

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

Exercise : Print Integer , Floating point number , Character

```
// 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);
```

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

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

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

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

### **Print Values Without Variables**

You can also just print a value without storing it in a variable, as long as you use the correct format specifier:

```
printf("My favorite number is: %d", 15);
printf("My favorite letter is: %c", 'D');
```

### **Change Variable Values**

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

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

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

```
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);
```

## 2.2 Data Types

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

### 2.2.1 Basic Data Types

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

Data Type	Size	Description	Example
int	2 or 4 bytes	Stores whole numbers, without decimals	1
float	4 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 6-7 decimal digits	1.99
double	8 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits	1.99
char	1 byte	Stores a single character/letter/number, or ASCII values	'A'

## Character Data Type

allows its variable to store only a single character. The size of the character is **1 byte**. It is the most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.

**Range:** (-128 to 127) or (0 to 255)

**Size:** 1 byte

**Format Specifier:** %c

```
#include <stdio.h>
```

```
int main() {  
    char ch = 'A'; // Character variable declaration  
    printf("ch = %c", ch);  
    return 0;  
}
```

```
ch = A
```

## Float Data Type

used to store single precision floating-point values. These values are decimal and exponential numbers.

**Range:** 1.2E-38 to 3.4E+38

**Size:** 4 bytes

**Format Specifier:** %f

```
#include <stdio.h>
```

```
int main() {  
    float val = 12.45;  
    printf("val = %f", val);  
}
```

```
    return 0;
}
```

```
val = 12.450000
```

## Double Data Type

used to store decimal numbers (numbers with floating point values) with double precision. It can easily accommodate about 16 to 17 digits after or before a decimal point.

**Range:** 1.7E-308 to 1.7E+308

**Size:** 8 bytes

**Format Specifier:** %lf

```
#include <stdio.h>
```

```
int main() {
    double val = 1.4521;
    printf("val = %lf", val);
    return 0;
}
```

```
val = 1.452100
```

## 2.3 Constants

Variables whose value cannot be changed. These variables are called constants and are created simply by prefixing [const](#) keyword in variable declaration.

**const** data\_type name = value;



### 2.3.1 Type Conversion

Process of changing one data type into another. This can happen automatically by the compiler or manually by the programmer. Type conversion is only performed between data types where such a conversion is possible.

```
int x = 5;  
int y = 2;  
int sum = 5 / 2;
```

```
printf("%d", sum); // Outputs 2
```

There are two types of conversion in C:

- **Implicit Conversion** (automatically)
- **Explicit Conversion** (manually)

#### Quiz to Ask :

1. Which data type is used to store integers (whole numbers) in C?

char

float

int

double

2. What format specifier is used to print an integer value in C?

%f

%d

%c

%s

3. What happens when you assign a new value to an existing variable in C?

The previous value is overwritten

The previous value is deleted, and the variable is cleared

The new value is ignored

An error occurs

4. What is the purpose of the const keyword in C?

To declare a variable that can be changed

To declare a variable that cannot be changed

To declare a variable without assigning a value

To declare a variable with a default value

## **Day 2: Operators & Conditions**

## 3 Operators Overview

### 3.1 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 myNum = 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)
```

### 3.2 Operator Types

C divides the operators into the following groups:

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

#### 3.2.1 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

### 3.2.2 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; // Output 15
```

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

<code>&amp;=</code>	<code>x &amp;= 3</code>	<code>x = x &amp; 3</code>
<code> =</code>	<code>x  = 3</code>	<code>x = x   3</code>
<code>^=</code>	<code>x ^= 3</code>	<code>x = x ^ 3</code>
<code>&gt;&gt;=</code>	<code>x &gt;&gt;= 3</code>	<code>x = x &gt;&gt; 3</code>
<code>&lt;&lt;=</code>	<code>x &lt;&lt;= 3</code>	<code>x = x &lt;&lt; 3</code>

### 3.2.3 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.

A list of all comparison operators:

Operator	Name	Example	Description
<code>==</code>	Equal to	<code>x == y</code>	Returns 1 if the values are equal
<code>!=</code>	Not equal	<code>x != y</code>	Returns 1 if the values are not equal
<code>&gt;</code>	Greater than	<code>x &gt; y</code>	Returns 1 if the first value is greater than the second value
<code>&lt;</code>	Less than	<code>x &lt; y</code>	Returns 1 if the first value is less than the second value
<code>&gt;=</code>	Greater than or equal to	<code>x &gt;= y</code>	Returns 1 if the first value is greater than, or equal to, the second value
<code>&lt;=</code>	Less than or equal to	<code>x &lt;= y</code>	Returns 1 if the first value is less than, or equal to, the second value

### 3.2.4 Logical Operators

We can also test for true or false values with logical operators.

Logical operators are used to determine the logic between variables or values, by combining multiple conditions:

Operator	Name	Example	Description
&&	AND	<code>x &lt; 5 &amp;&amp; x &lt; 10</code>	Returns 1 if both statements are true
	OR	<code>x &lt; 5    x &lt; 4</code>	Returns 1 if one of the statements is true
!	NOT	<code>!(x &lt; 5 &amp;&amp; x &lt; 10)</code>	Reverse the result, returns 0 if the result is 1

## 4. Conditional Statements

### 4.1 Conditions and If Statements

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

#### 4.1.1 If Statement

Use the if statement to specify a block of 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:

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

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"

### 4.1.2 The else Statement

Use the else statement to specify a block of code to be executed if the condition is false.

```
int time = 20;  
if (time < 18) {  
    printf("Good day.");  
} else {  
    printf("Good evening.");  
}  
// Outputs "Good evening."
```

#### Example explained

In the example above, time (20) is greater than 18, so the condition is false. Because of this, we move on to the else condition and print to the screen "Good evening". If the time was less than 18, the program would print "Good day".



### 4.1.3 The else if Statement

Use the else if statement to specify a new condition if the first condition is false.

```
int time = 22;
if (time < 10) {
    printf("Good morning.");
} else if (time < 20) {
    printf("Good day.");
} else {
    printf("Good evening.");
}
// Outputs "Good evening."
```

#### Example explained

In the example above, time (22) is greater than 10, so the **first condition** is false. The next condition, in the else if statement, is also false, so we move on to the else condition since **condition1** and **condition2** is both false - and print to the screen "Good evening".

However, if the time was 14, our program would print "Good day."

## 4.2 Ternary Operator

There is also a short-hand if else, which is known as the **ternary operator** because it consists of three operands. It can be used to replace multiple lines of code with a single line. It is often used to replace simple if else statements:

#### Syntax

```
variable = (condition) ? expressionTrue : expressionFalse;
```

#### Example

```
int time = 20;
if (time < 18) {
    printf("Good day.");
} else {
    printf("Good evening.");
} // Output : Good Evening
```

## 4.3 Switch Case Statements

Instead of writing **many** if..else statements, you can use the switch statement.

The switch statement selects one of many code blocks to be executed:

### Syntax

```
switch (expression) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
    default:  
        // code block  
}
```

This is how it works:

- The switch expression is evaluated once
- The value of the expression is compared with the values of each case
- If there is a match, the associated block of code is executed
- The break statement breaks out of the switch block and stops the execution
- The default statement is optional, and specifies some code to run if there is no case match

The example below uses the weekday number to calculate the weekday name:

Example :

```
int day = 4;
```

```
switch (day) {
```

```
case 1:
    printf("Monday");
    break;
case 2:
    printf("Tuesday");
    break;
case 3:
    printf("Wednesday");
    break;
case 4:
    printf("Thursday");
    break;
case 5:
    printf("Friday");
    break;
case 6:
    printf("Saturday");
    break;
case 7:
    printf("Sunday");
    break;
}
```

// Outputs "Thursday" (day 4)

### **The break Keyword**

When C reaches a break keyword, it breaks out of the switch block.

This will stop the execution of more code and case testing inside the block.

When a match is found, and the job is done, it's time for a break. There is no need for more testing.

**A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.**

## The default Keyword

The default keyword specifies some code to run if there is no case match:

Example :

```
int day = 4;
```

```
switch (day) {  
    case 6:  
        printf("Today is Saturday");  
        break;  
    case 7:  
        printf("Today is Sunday");  
        break;  
    default:  
        printf("Looking forward to the Weekend");  
}
```

```
// Outputs "Looking forward to the Weekend"
```

Quiz to Ask :

1.Which of the following operators checks if two values are equal in C?

=

==

!=

<>

2. What is the purpose of the else statement in C?

To execute a block of code if the if condition is true

To execute a block of code if the if condition is false

To create a new condition

To end the program

3. What is the purpose of the ternary operator in C?

To write multiple lines of code within an if...else statement

To replace a simple if...else statement with a single line of code

To handle complex conditions that require nested if statements

To declare a new variable

4. What is the purpose of the switch statement in C?

To select one of many code blocks to be executed

To loop through a set of conditions

To compare two values

To declare multiple variables

## **Day 3 : Loops & Arrays**

## 5. Loop Structures

Loops can execute a block of code as long as a specified condition is reached.

Loops are handy because they save time, reduce errors, and they make code more readable.

### 5.1 While Loop

The while loop loops through a block of code as long as a specified condition is true:

**Syntax:**

```
while (condition) {  
    // code block to be executed  
}
```

**Example:**

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

Output :

```
0  
1  
2  
3  
4
```

### 5.2 Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

**Syntax:**

```
do {  
    // code block to be executed  
}
```

**while (condition);**

**Example:** Even if the condition is false from the start, the code block will still execute one time

```
int i = 10;
```

```
do {  
    printf("i is %d\n", i);  
    i++;  
} while (i < 5);
```

A terminal window with a black background and white text showing the output of the printf statement: "i is 10".

The do/while loop always runs at least once, even if the condition is already false. This is different from a regular while loop, which would skip the loop entirely if the condition is false at the start.

This behavior makes do/while useful when you want to ensure something happens at least once, like showing a message or asking for user input.

## 5.3 For Loop

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:

**Syntax :**

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

**Expression 1** is executed (one time) before the execution of the code block.

**Expression 2** defines the condition for executing the code block.



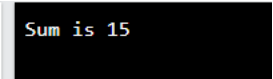
**Expression 3** is executed (every time) after the code block has been executed.

#### Example : Sum of Numbers

```
int sum = 0;
int i;

for (i = 1; i <= 5; i++) {
    sum = sum + i;
}

printf("Sum is %d", sum);
```

A terminal window with a black background and white text. The text reads "Sum is 15".

Sum is 15

## 5.4 Nested Loop

It is also possible to place a loop inside another loop. This is called a **nested loop**.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Example :

```
int i, j;

// Outer loop
for (i = 1; i <= 2; ++i) {
    printf("Outer: %d\n", i); // Executes 2 times

    // Inner loop
    for (j = 1; j <= 3; ++j) {
        printf(" Inner: %d\n", j); // Executes 6 times (2 * 3)
    }
}
```

Output :

```
Outer: 1
  Inner: 1
  Inner: 2
  Inner: 3
Outer: 2
  Inner: 1
  Inner: 2
  Inner: 3
```

## 5.4 Break and Continue

The break statement can also be used to jump out of a **loop**.

This example jumps out of the **for loop** when i is equal to 4:

Example :

```
int i;
```

```
for (i = 0; i < 10; i++) {
    if (i == 4) {
        break;
    }
    printf("%d\n", i);
}
```

```
0
1
2
3
```

### Continue

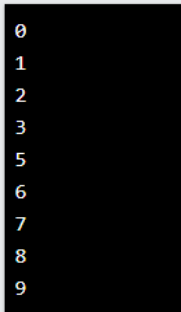
The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 4:

```
int i;
```

```
for (i = 0; i < 10; i++) {
    if (i == 4) {
        continue;
    }
}
```

```
printf("%d\n", i);  
}
```



0  
1  
2  
3  
5  
6  
7  
8  
9

## 6. Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To create an array, define the data type (like int) and specify the name of the array followed by **square brackets []**.

To insert values to it, use a comma-separated list inside curly braces, and make sure all values are of the same data type:

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

### 6.1 Arrays

#### 6.1.1 Access the Elements of an Array

To access an array element, refer to its **index number**.

Array indexes start with **0**: [0] is the first element. [1] is the second element, etc.

This statement accesses the value of the **first element [0]** in myNumbers:

Example :

```
int myNumbers[] = {25, 50, 75, 100};  
printf("%d", myNumbers[0]);
```

```
// Outputs 25
```

## 6.1.2 Change an Array Element

To change the value of a specific element, refer to the index number:

Example :

```
int myNumbers[] = {25, 50, 75, 100};  
myNumbers[0] = 33;
```

```
printf("%d", myNumbers[0]);
```

// Now outputs 33 instead of 25

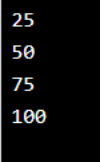
## 6.1.3 Loop Through an Array

You can loop through the array elements with the for loop.

The following example outputs all elements in the myNumbers array:

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

```
for (i = 0; i < 4; i++) {  
    printf("%d\n", myNumbers[i]);  
}
```



```
25  
50  
75  
100
```

## 6.1.4 Set Array Size

Another common way to create arrays, is to specify the size of the array, and add elements later:

```
#include <stdio.h>
```

```
int main() {
```

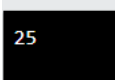
```
    // Declare an array of four integers:
```

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

```
// Add elements to it
myNumbers[0] = 25;
myNumbers[1] = 50;
myNumbers[2] = 75;
myNumbers[3] = 100;

printf("%d\n", myNumbers[0]);

return 0;
}
```



## 6.1.4 Multidimensional Arrays

A multidimensional array is basically an array of arrays.

Arrays can have any number of dimensions. In this chapter, we will introduce the most common; two-dimensional arrays (2D).

### Two-Dimensional Arrays

A 2D array is also known as a matrix (a table of rows and columns).

To create a 2D array of integers, take a look at the following example:

```
int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };
```

The first dimension represents the number of rows [2], while the second dimension represents the number of columns [3]. The values are placed in row-order, and can be visualized like this:

	COLUMN 0	COLUMN 1	COLUMN 2
ROW 0	1	4	2
ROW 1	3	6	8

### Access the Elements of a 2D Array

To access an element of a two-dimensional array, you must specify the index number of both the row and column.

This statement accesses the value of the element in the first row (0) and third column (2) of the matrix array.

Example :

```
int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };
```

```
printf("%d", matrix[0][2]); // Outputs 2
```

### Change Elements in a 2D Array

To change the value of an element, refer to the index number of the element in each of the dimensions:

The following example will change the value of the element in the first row (0) and first column (0):

```
int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };
```

```
matrix[0][0] = 9;
```

```
printf("%d", matrix[0][0]); // Now outputs 9 instead of 1
```

Quiz to Ask :

1.What is the purpose of the while loop in C?

To execute a block of code a fixed number of times

To execute a block of code as long as a specified condition is true

To execute code only if a condition is false

To declare multiple variables in a loop

2. What is the main difference between a do/while loop and a while loop?

The do/while loop checks the condition first

**The do/while loop executes the code block at least once before checking the condition**

The do/while loop only runs if the condition is initially false

The do/while loop is only used for infinite loops

3. What is the main use of a for loop in C?

To execute a block of code indefinitely

**To loop through a block of code a specific number of times**

To check conditions without executing any code

To run a loop based on user input only

4. Of the following, what is the correct syntax to declare an array of integers called myNumbers with the values 25, 50, 75, and 100?

`int myNumbers[25, 50, 75, 100];`

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

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

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

5. What does the following declaration represent?

`int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };`

A 1-dimensional array of 5 elements

**A 2-dimensional array with 2 rows and 3 columns**

A 3-dimensional array with 2 rows and 3 columns

A 1-dimensional array with 6 elements

## **Day 4: Functions in C**



## 7. Functions in C

### 7.1 Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.

#### 7.1.1 Predefined Functions

`main()` is a function, which is used to execute code, and `printf()` is a function; used to output/print text to the screen

**Example :**

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

#### 7.1.2. Create a Function

To create (often referred to as *declare*) your own function, specify the name of the function, followed by parentheses () and curly brackets {}:

**Syntax :**

```
void myFunction() {  
    // code to be executed  
}
```

**Example Explained**

`myFunction()` is the name of the function

`void` means that the function does not have a return value. You will learn more about return values later in the next chapter

Inside the function (the body), add code that defines what the function should do

### 7.1.3. Call a Function

Declared functions are not executed immediately. They are "saved for later use", and will be executed when they are called.

To call a function, write the function's name followed by two parentheses () and a semicolon ;

In the following example, myFunction() is used to print a text (the action), when it is called:

Example :

Inside main, call myFunction():

```
// Create a function
void myFunction() {
    printf("I just got executed!");
}

int main() {
    myFunction(); // call the function
    return 0;
}

// Outputs "I just got executed!"
```

## 7.2 Functions Parameters

### 7.2.1 Parameters and Arguments

Information can be passed to functions as a parameter. Parameters act as variables inside the function.

Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma:

**Syntax :**

```
returnType functionName(parameter1, parameter2, parameter3) {
    // code to be executed
}
```

When a **parameter** is passed to the function, it is called an **argument**. So, from the example above: name is a **parameter**, while Liam, Jenny and Anja are **arguments**.

## 7.2.2 Multiple Parameters

Inside the function, you can add as many parameters as you want:

Example :

```
void myFunction(char name[], int age) {  
    printf("Hello %s. You are %d years old.\n", name, age);  
}
```

```
int main() {  
    myFunction("Liam", 3);  
    myFunction("Jenny", 14);  
    myFunction("Anja", 30);  
    return 0;  
}
```

```
// Hello Liam. You are 3 years old.  
// Hello Jenny. You are 14 years old.  
// Hello Anja. You are 30 years old.
```

## 7.3 Variable Scope in C

### Scope

Now that you understand how functions work, it is important to learn how variables act inside and outside of functions.

In C, variables are only accessible inside the region they are created. This is called scope.

### 7.3.1 Local Scope

A variable created inside a function belongs to the *local scope* of that function, and can only be used inside that function. A **local variable** cannot be used outside the function it belongs to.

### Example :

```
void myFunction() {  
    // Local variable that belongs to myFunction  
    int x = 5;  
  
    // Print the variable x  
    printf("%d", x);  
}  
  
int main() {  
    myFunction();  
    return 0;  
}
```

### 7.3.2 Global Scope

A variable created outside of a function, is called a **global variable** and belongs to the *global scope*.

Global variables are available from within any scope, global and local

### Example :

A variable created outside of a function is global and can therefore be used by anyone:

```
// Global variable x  
int x = 5;  
  
void myFunction() {  
    // We can use x here  
    printf("%d", x);  
}  
  
int main() {  
    myFunction();  
  
    // We can also use x here
```

```
printf("%d", x);  
return 0;  
}
```

### 7.3.3 Naming Variables

If you operate with the same variable name inside and outside of a function, C will treat them as two separate variables; One available in the global scope (outside the function) and one available in the local scope (inside the function):

**Example :**

```
// Global variable x  
int x = 5;  
  
void myFunction() {  
    // Local variable with the same name as the global variable (x)  
    int x = 22;  
    printf("%d\n", x); // Refers to the local variable x  
}  
  
int main() {  
    myFunction();  
  
    printf("%d\n", x); // Refers to the global variable x  
    return 0;  
}
```

## 8. Recursion in C

Recursion is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve.

Recursion may be a bit difficult to understand. The best way to figure out how it works is to experiment with it.

Example :

Adding two numbers together is easy to do, but adding a range of numbers is more complicated. In the following example, recursion is used to add a range of numbers together by breaking it down into the simple task of adding two numbers:

```
int sum(int k);
```

```
int main() {  
    int result = sum(10);  
    printf("%d", result);  
    return 0;  
}
```

```
int sum(int k) {  
    if (k > 0) {  
        return k + sum(k - 1);  
    } else {  
        return 0;  
    }  
}
```

### Example Explained

When the sum() function is called, it adds parameter k to the sum of all numbers smaller than k and returns the result. When k becomes 0, the function just returns 0. When running, the program follows these steps:

```
10 + sum(9)  
10 + ( 9 + sum(8) )  
10 + ( 9 + ( 8 + sum(7) ) )  
...  
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + sum(0)  
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0
```

Since the function does not call itself when k is 0, the program stops there and returns the result.

Quiz to Ask :

1.What does the keyword void indicate when used with a function in C?

The function returns an integer

The function can only be called once

The function does not return a value

The function can accept any type of parameter

2. What will the following code output?

```
void myFunction(char name[]) {  
    printf("Hello %s\n", name);  
}
```

```
int main() {  
    myFunction("Alice");  
    return 0;  
}
```

Hello World

Hello Alice

Alice

Hello

3. Where can a variable with local scope be accessed?

Inside the function it was created

Anywhere in the program

Only inside the main function

Only in global scope

4. What is recursion in C programming?

A function calling another function

A function calling itself

Repeating a loop indefinitely

Using a function inside another function



## **Day 5: Number Guessing Game**

## 9. Introduce the concept of random number generation using rand()

Computers can pretend to choose a random number using a special function. In C, we use rand() to ask the computer to give us a random number

"Imagine a box with numbers from 1 to 100. You close your eyes and pick one — that's what the computer is doing with rand()!"

### **Game Rules :**

- The computer picks a number between 1 and 100.
- You (the player) try to guess it.
- The computer tells you if your guess is too high or too low.
- Keep guessing until you find the right number.