C language for intermediate:

Example:

You have already seen the following code in the biggening level. Let's break it down to understand it better:

```
#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 functions, such as `printf()` (used in line 4). Header files add functionality to C programs.

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!".

C 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;
}
```

Double Quotes

When you are working with text, it must be wrapped inside double quotations marks `""`.

If you forget the double quotes, an error occurs:

Example:

```
printf("This sentence will work!");


printf(This sentence will produce an error.);
```

Many `printf` Functions:

You can use as many `printf()` functions as you want.

Example:

```
#include <stdio.h>

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

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;
}
```

You can also output multiple lines with a single `printf()` function.

Example:

```c
#include <stdio.h>

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

C Constants:

If you don't want others (or yourself) to change existing variable values, you can use the `const` keyword.

This will declare the variable as "constant", which means **unchangeable** and **read-only**:

Example:

```c
const int myNum = 15;  // myNum will always be 15
myNum = 10;  // error: assignment of read-only variable 'myNum'
```

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;
```

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

Operator	Name	Example	Description
==	Equal to	x == y	Returns 1 if the values are equal
!=	Not equal	x != y	Returns 1 if the values are not equal
>	Greater than	x > y	Returns 1 if the first value is greater than th second value
<	Less than	x < y	Returns 1 if the first value is less than the second value
>=	Greater than or equal to	x >= y	Returns 1 if the first value is greater than, o equal to, the second value
<=	Less than or equal to	x <= y	Returns 1 if the first value is less than, or e to, the second value

Example:

```
int x = 5;
int y = 3;
printf("%d", x > y); // returns 1 (true) because 5 is greater than 3
```

Logical Operators:

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

Logical operators are used to determine the logic between variables or values:

Operator	Name	Example	Description
&&	Logical and	x < 5 && x < 10	Returns 1 if both statements are true
||	Logical or	x < 5 || x < 4	Returns 1 if one of the statements is true
!	Logical not	!(x < 5 && x < 10)	Reverse the result, returns 0 if the result is 1

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 is known as **booleans**.

Booleans represent values that are either `true` or `false`.

Boolean Variables:

In C, the `bool` type is not a built-in data type, like `int` or `char`.

It was introduced in C99, and you must **import** the following header file to use it:

```
#include <stdbool.h>
```

Example:

```
/ Create boolean variables
bool isProgrammingFun = true;
bool isFishTasty = false;

// Return boolean values
printf("%d", isProgrammingFun);   // Returns 1 (true)
printf("%d", isFishTasty);        // Returns 0 (false)
```

comparing Values and Variables:

Comparing values are useful in programming, because it helps us to find answers and make decisions.

Example:

```
int x = 10;
int y = 9;
printf("%d", x > y);
```

in the example below, we use the **equal to** (==) operator to compare different values:

Example

```
printf("%d", 10 == 10); // Returns 1 (true), because 10 is equal to 10
printf("%d", 10 == 15); // Returns 0 (false), because 10 is not equal to
15
printf("%d", 5 == 55);  // Returns 0 (false) because 5 is not equal to 55
```

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

The 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
```

Example:

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

Example:

```
int x = 20;
int y = 18;
if (x > y) {
  printf("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".

C Else:

The else Statement:

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

Syntax

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

Example

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

Try it Yourself »

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".

C Else If:

The else if Statement:

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

Syntax

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

Example

```
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".