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.

Predefined Functions:-

So it turns out you already know what a function is. You have been using it the whole time while studying this tutorial!

For example, 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;

}
```

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

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!"
```

A function can be called multiple times:

Example

```
void myFunction() {
    printf("I just got executed!");
}
```

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

// I just got executed!
// I just got executed!
// I just got executed!
```

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
}
```

The following function that takes a string of characters with name as parameter. When the function is called, we pass along a name, which is used inside the function to print "Hello" and the name of each person.

Example

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

int main() {
    myFunction("Liam");
    myFunction("Jenny");
    myFunction("Anja");
    return 0;
}

// Hello Liam
// Hello Jenny
// Hello Anja
```

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.

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.
```

Note that when you are working with multiple parameters, the function call must have the same number of arguments as there are parameters, and the arguments must be passed in the same order.

Pass Arrays as Function Parameters

You can also pass arrays to a function:

Example

```
void myFunction(int myNumbers[5]) {
  for (int i = 0; i < 5; i++) {
    printf("%d\n", myNumbers[i]);
  }
}

int main() {
  int myNumbers[5] = {10, 20, 30, 40, 50};
  myFunction(myNumbers);
  return 0;
}
```

Example Explained

The function (myFunction) takes an array as its parameter (int myNumbers[5]), and loops through the array elements with the for loop.

When the function is called inside main(), we pass along the myNumbers array, which outputs the array elements.

Note that when you call the function, you only need to use the name of the array when passing it as an argument myFunction(myNumbers). However, the full declaration of the array is needed in the function parameter (int myNumbers[5]).

Return Values

The void keyword, used in the previous examples, indicates that the function should not return a value. If you want the function to return a value, you can use a data type (such as int or float, etc.) instead of void, and use the return keyword inside the function:

Example

```
int myFunction(int x) {
  return 5 + x;
}
```

```
int main() {
    printf("Result is: %d", myFunction(3));
    return 0;
}

// Outputs 8 (5 + 3)
```

This example returns the sum of a function with two parameters:

Example

```
int myFunction(int x, int y) {
    return x + y;
}

int main() {
    printf("Result is: %d", myFunction(5, 3));
    return 0;
}

// Outputs 8 (5 + 3)
```

You can also store the result in a variable:

Example

```
int myFunction(int x, int y) {
    return x + y;
}

int main() {
    int result = myFunction(5, 3);
    printf("Result is = %d", result);
    return 0;
}
// Outputs 8 (5 + 3)
```

Real-Life Example

To demonstrate a practical example of using functions, let's create a program that converts a value from fahrenheit to celsius:

Example

```
// Function to convert Fahrenheit to Celsius
float toCelsius(float fahrenheit) {
    return (5.0 / 9.0) * (fahrenheit - 32.0);
}


int main() {
    // Set a fahrenheit value
    float f_value = 98.8;


    // Call the function with the fahrenheit value
    float result = toCelsius(f_value);


    // Print the fahrenheit value
    printf("Fahrenheit: %.2f\n", f_value);


    // Print the result
    printf("Convert Fahrenheit to Celsius: %.2f\n", result);


    return 0;
}
```

Function Declaration and Definition

You just learned from the previous chapters that you can create and call a function in the following way:

Example

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

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

A function consist of two parts:

Declaration: the function's name, return type, and parameters (if any)

Definition: the body of the function (code to be executed)

```
void myFunction() { // declaration
    // the body of the function (definition)
}
```

For code optimization, it is recommended to separate the declaration and the definition of the function.

You will often see C programs that have function declaration above main(), and function definition below main(). This will make the code better organized and easier to read:

Example

```
// Function declaration

void myFunction();


// The main method

int main() {

   myFunction();    // call the function

   return 0;

}


// Function definition

void myFunction() {

   printf("I just got executed!");

}
```

Another Example

If we use the example from the previous chapter regarding function parameters and return values:

Example

```
int myFunction(int x, int y) {
    return x + y;
}


int main() {
    int result = myFunction(5, 3);
    printf("Result is = %d", result);
    return 0;
}
// Outputs 8 (5 + 3)
```

It is considered good practice to write it like this instead:

Example

```
// Function declaration
int myFunction(int, int);


// The main method
int main() {
    int result = myFunction(5, 3); // call the function
    printf("Result is = %d", result);
    return 0;
}


// Function definition
int myFunction(int x, int y) {
    return x + y;
}
```

Recursion

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.

Recursion 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:

Example

```
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;
    }
}
```