Functions - Defining a function

- When you create a function, you specify the function header as the first line of the function definition.
    - ➔ Followed by a starting curly brace {
    - ➔ The executable code in between the starting and ending braces.
    - ➔ The ending curly brace }
    - ➔ The block of code between braces following the function header is called function body.
- The function header defines the name of the function,
    - ➔ Parameters (which specify the number and types of values that are passed to the function when it's called)
    - ➔ The type for the values that the function returns.

```
Return _type   Function_name ( parameters - separated by commas)
{
        //Statements...
}
```

- The first line of a function definition tells the compiler (in order from left to right) three things about three things about the function:
    1. The type of value it returns
    2. Its name
    3. The argument it takes
- Choosing meaningful functions names is just as important as choosing meaningful variable names greatly affects the programs readability.

```
void printfMessage (void)
{
        printf("Programming is fun.\n");
}
```

- The first line of the printMessage() function definition tells the compiler that the function returns no value.
    - ➔ Keyword void
- Next is its name: printMessage
- After that is that it takes no arguments (the second use of the keyword void)

- The statements in the function body can be absent, but the braces must be present.
- If there are no statements in the body of a function, the return type must be void, and the function will not do anything.
    - ➔ Defining a function with an empty body is often useful during the testing phase of a complicated program.

- ➔ Allows you to run the program with only selected functions actually doing something.
- ➔ You can then add the detail for the function bodies step by step, testing at each stage, until the whole thing is implemented and fully tested.

Naming functions.

- The name of a function can be any legal name.
  - ➔ Not a reserved word (such as int , double , sizeof , and so on)
  - ➔ Is not the same name as another function in your program.
  - ➔ Is not the same name as any of the standard library functions.
    - ★ Would prevent you from using the library function.
- A legal name had the same form as that of a variable .
  - ➔ A sequence of letters and digits
  - ➔ First character must be a letter
  - ➔ Underline character counts as a letter
- The name that you choose should be meaningful and relevant to what the function does.
- You will often define function names (a variable names, too) that consist of more than one word.
- There are three common approaches you can adopt.
  - ➔ Separate each of the words in a function name with an underline character
  - ➔ Capitalize the first letter of each word.
  - ➔ Capitalize words after the first (camelCase)
- Can pick any one you want, but use the same approach throughout your program.

Function Prototypes

- A function prototype is a statement that defines a function.
  - ➔ Defines its name, its return value type of each of its parameters.
  - ➔ Provides all the external specifications for the function.
- You can write a prototype for a function exactly the same as the function header.
  - ➔ Only difference is that you ass a semicolon at the end.

```
void printMessage (void);
```

- A function prototype enables the compiler to generate the appropriate instructions at each point where you call the function.
  - ➔ It also checks that you are using the function correctly in each invocation.
- When you include a standard header file in a program, the header file adds the function prototypes for that library to your program.
  - ➔ The header file stdio.h contains function prototypes for printf(), among others
- Generally appear at the beginning of a source file prior to the implementations of any functions or in a header file .
- Allows any of the functions in the file to call any function regardless of where you have placed the implementation of the functions.

- Parameter names do not have to be the same as those used in the function definition,
    - ➔ Not required to include the names of parameters in a function prototype.
- Its good practice to always include declarations for all of the functions in a program source file, regardless of where they are called.
    - ➔ Will help keep our programs more consistent in design.
    - ➔ Prevent any errors from occurring if, at any stage , you choose to call a function from another part of your program.

Example

```
// #include & #define directives...

// Function prototypes
double Average(double data_values[], size_t count);
double Sum(double *x, size_t n);
size_t GetData(double*, size_t);

int main(void)
{
  // Code in main() ...
}

// Definitions/implementations for Average(), Sum() and GetData()...
```