week 3_1:

functions

rajin teaches programming

UNDERSTANDING FUNCTIONS IN C PROGRAMMING:

– INTRODUCTION TO FUNCTIONS:

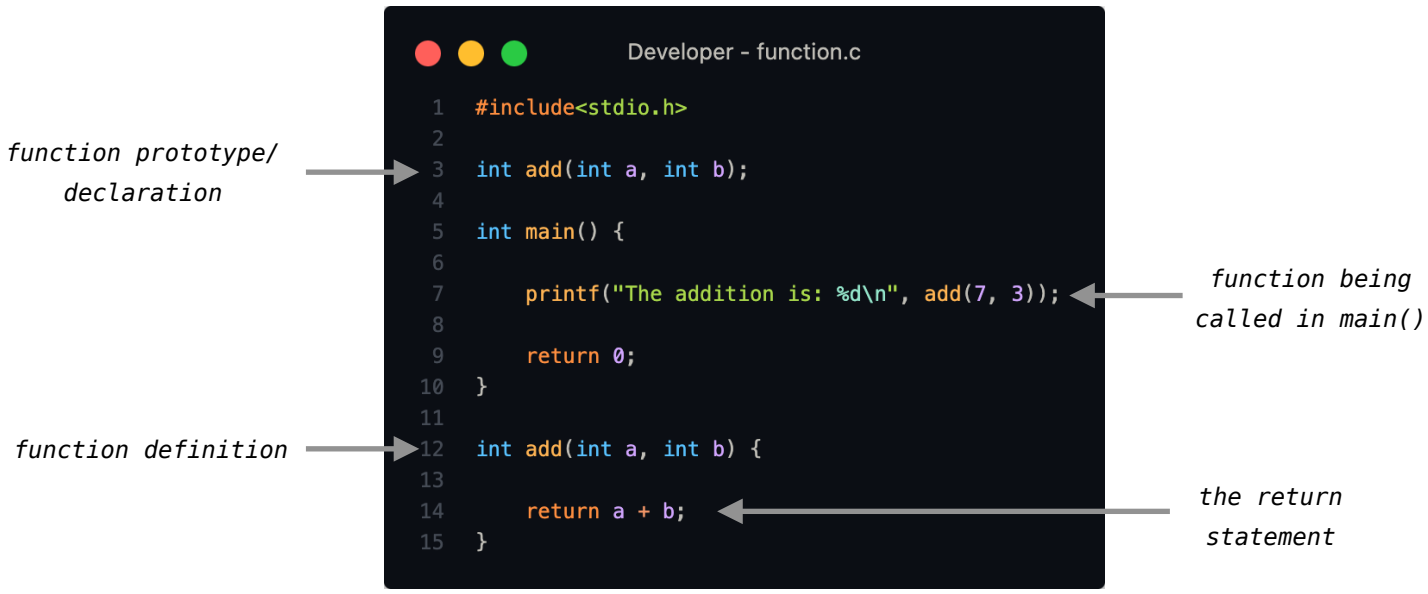
Functions are **fundamental** building blocks in C programming. They allow us to **break down** our program into **smaller**, manageable **pieces** of code, making it easier to **understand**, **maintain**, and **debug**. Functions also promote **code reusability**, as they can be **called multiple times** from **different parts** of the program.

– SYNTAX OF A FUNCTION:

FUNCTION

```
return_type function_name(parameters) {
    // Function body
    // Code to be executed
    return value; (optional)
}
```

term	explanation
return_type	Data type of the value that the function returns.
function_name	Name of the function.
parameters	Input values passed to the function (optional).
return value	Value returned by the function (optional).



an example of a typical function for adding two numbers.

– FUNCTION PROTOTYPES:

A function **prototype** is a **declaration** of a function **before** its **definition.** It tells the **compiler** about the function's **name, return type,** and **parameters.** It is the **first** thing labelled in the above diagram.

– FUNCTION DEFINITION:

The function is then **defined after** the **main** code. To define a function, you need to **specify** its **return type** (**void function** if **no return type**), **name,** and **parameters** (**if** any).

– THE RETURN STATEMENT:

The **return statement** is used to **exit** a **function** and **return** a **value** (**if** any). It can be used to **pass back** a **result** to the **calling function.** This means that a return **statement** is **mandatory,** while a return **value** is optional according to our **needs.**

– CALLING A FUNCTION:

To **call** a function, use its **name** followed by **parentheses containing** the **arguments** (**if** any). It can be called in **any block** of code **once defined.** (Even in **itself.** That is called **recursion,** and we shall discuss it **later.**)

– SCOPE OF VARIABLES:

Variables declared **inside** a function are **local** to **that** function and **cannot** be accessed **outside** it (local variables). Similarly, **variables** declared **outside any** function (global variables) can be accessed by **any function** in the program.

Let's analyze the function above. The variable **num** is passed **into** the **void** function **varchange().** In this function, the value of **any number** that is **passed** is **incremented.** So why is it that when the **num** variable with the value of **10** is **passed,** and then **printed after** the function is **called,** it is **not** 11? This is because the **value** that is **passed into** the function is **assigned** to a **new variable** in the **void function.** We call that a **local variable. Changes** to the **local variable** are **only visible in** its **own code block.** Thus, the num variable **does not** change its value, as **no changes** were made to **num** in its **local code block.** (We can **fix** this problem with **pointers,** but that is a topic we will discuss around the **end** of this course.)

PRACTICE QUESTIONS:

- Write a function to calculate the factorial of a number, and call and use it in main(). (*Function Name: factorial, Input: int n, Output: factorial of n)*

- Write a function to calculate the to calculate the power of a number, and call and use it in main. (*Function Name: power, Input: int base, int exponent, Output: base raised to the power of exponent)*

- Write a function to implement a calculator that takes two numbers as an input, then asks the user which operation to perform, and displays the result. You must use functions to implement your solution. (*Function to be included: add, subtract, multiply, divide)*

next class 3_2:

arrays

rajin teaches programming

reach out for classes at: rajin.khan2001@gmail.com