DeepDive into Functions in C & Debugging

Youtube Live Stream

Functions

Function/subroutine/procedure

- 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.*
- You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.

Function skeleton

```
return_type function_name( parameter list ) {
   body of the function
}
```

- **return_type**– A function may return a value. It is the *data type* of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword `void`.
- **function_name** – This is the actual name of the function.
- **parameter list** – A parameter is like a *placeholder*. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body** – The function body contains a collection of statements that define what the function does.

Predefined Functions

- The C std library provides numerous built-in functions that your program can call. For example, strcat() to concatenate two strings, memcpy() to copy one memory location to another location etc.
- For example,
  - `main()` is a function, which is used to execute code
  - `printf()` is a function; used to output/print text to the screen

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

Creating(Declaring) a function

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

- `myFunction()` is the name of the function
- `void` means that the function does not have a return value.
- Inside the function body (commented area), add code that defines what the function should do

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

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:

```
//Task: Inside main, call myFunction():

#include <stdio.h>

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

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

Function Parameters

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*
- When a parameter is passed to the function during a call, it is called an argument.

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

Function return values

- The `void` keyword e.g `main(void)` 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:

```c
#include <stdio.h>

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

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

Function Declaration and Definition

Declaration & Definition

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)
}
```

Code optimization

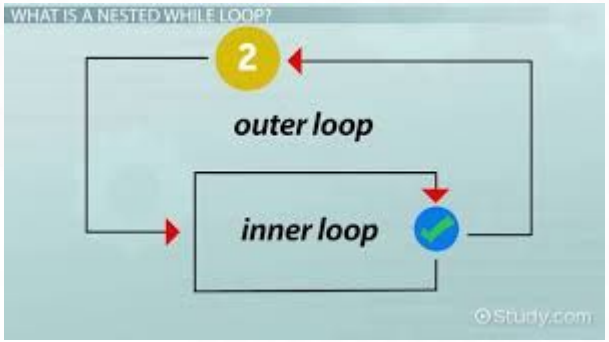
```c
// Function declaration
void myFunction();

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

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

- 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.

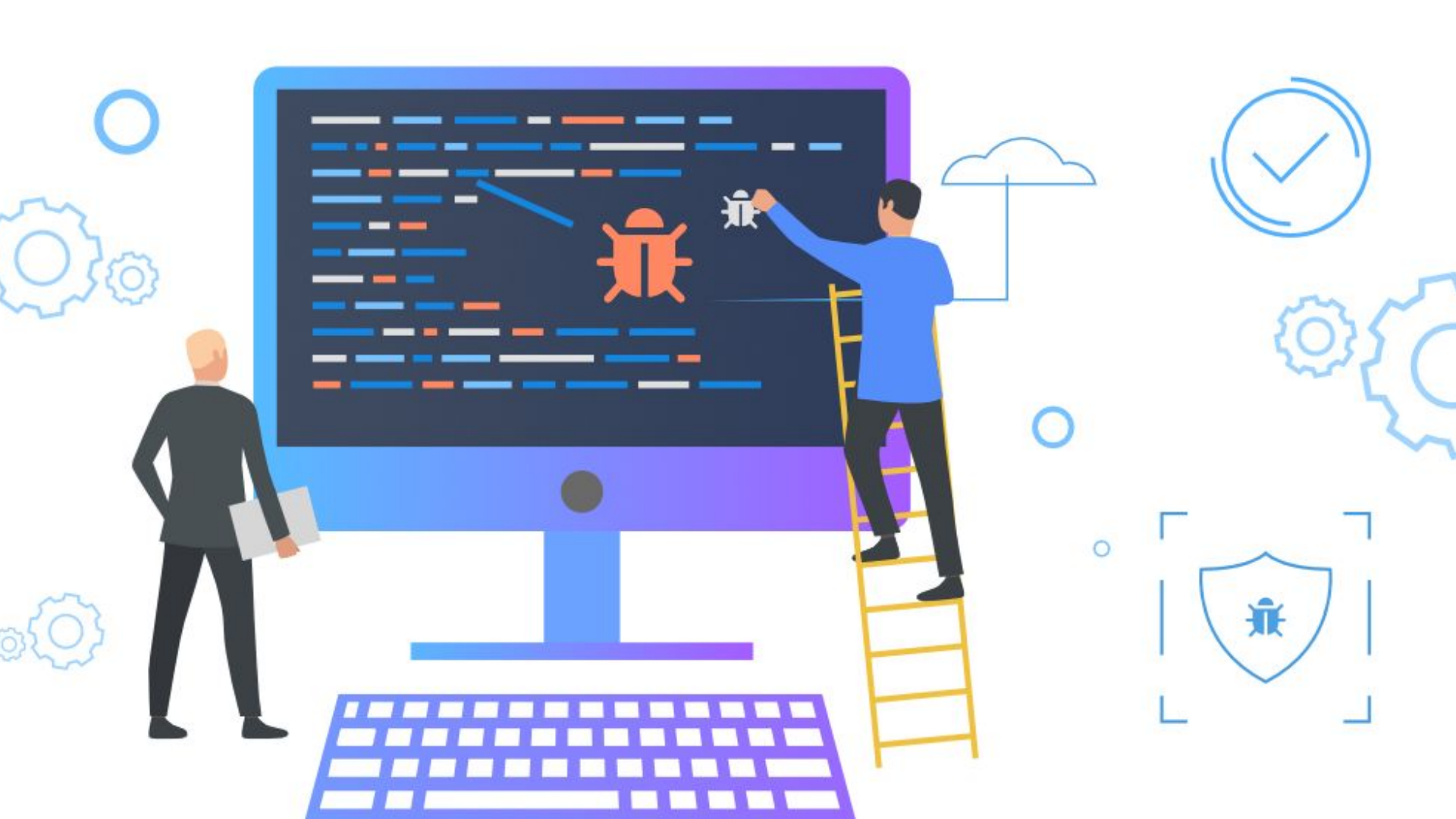
Nested LOOPs



Definition

- A nested loop means a loop statement inside another loop statement. That is why nested loops are also called "loop inside loops".

Debugging



Resources

1. Debugging 4 beginners
2. GDB Debugging Tool
3. Debugging Techniques
4. Nested Loops
5. Feyman’s technique
6. Rubberduck debugging

See you at the next session!