Lecture 11: Functions and Pointers

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!

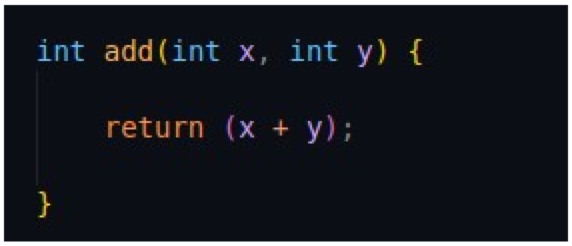
For example, `main()` is a function, which is used to execute code, and `printf()` is a function; used to output/print text to the screen.

Creating your own function:

To declare your own function, you will need to tell the compiler about the essential data it needs to know how it will work with the function:

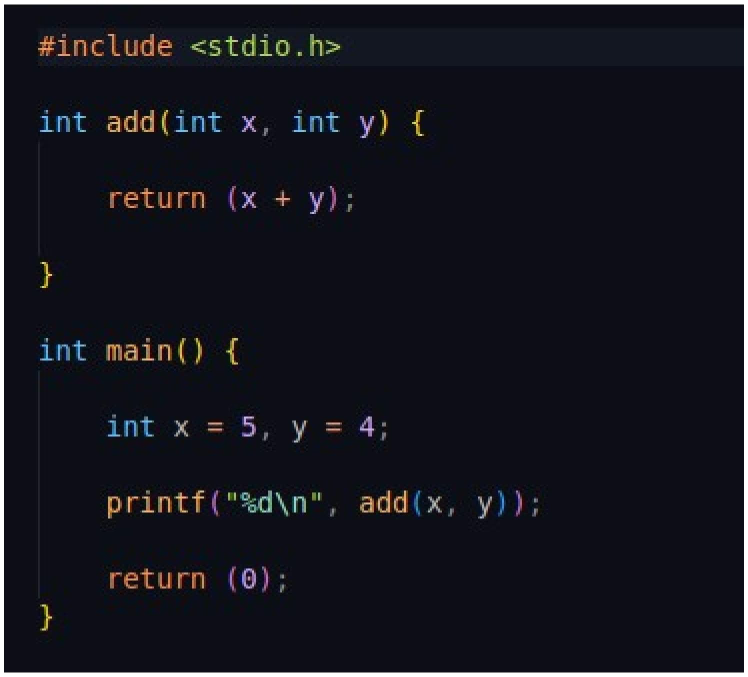
1. First we declare the data type that the function will return to us. In some cases, we would have a function that doesn't return any data, in this case, we have the `void` data type that indicates that there is no data, or the data is 'void', or in other words, none.
2. Second, we will tell the compiler what will be the name of our function, which will be the name that we will be calling the function with to be executed.
3. Then, we will tell the compiler if the function needs any to be passed to it inside the ' ( )'. These values are called 'parameters' when we first declare them, and 'arguments' when we use them.
4. Lastly, we will give the compiler the code that will be executed when calling the function inside the { }.

For example:



```
int add(int x, int y) {

    return (x + y);

}
```

Here we defined a function that returns the data type `int` and has the name “add”. Then we declared the parameters that will be passed to the function x and y, and lastly, the logic that will be executed when calling the function.



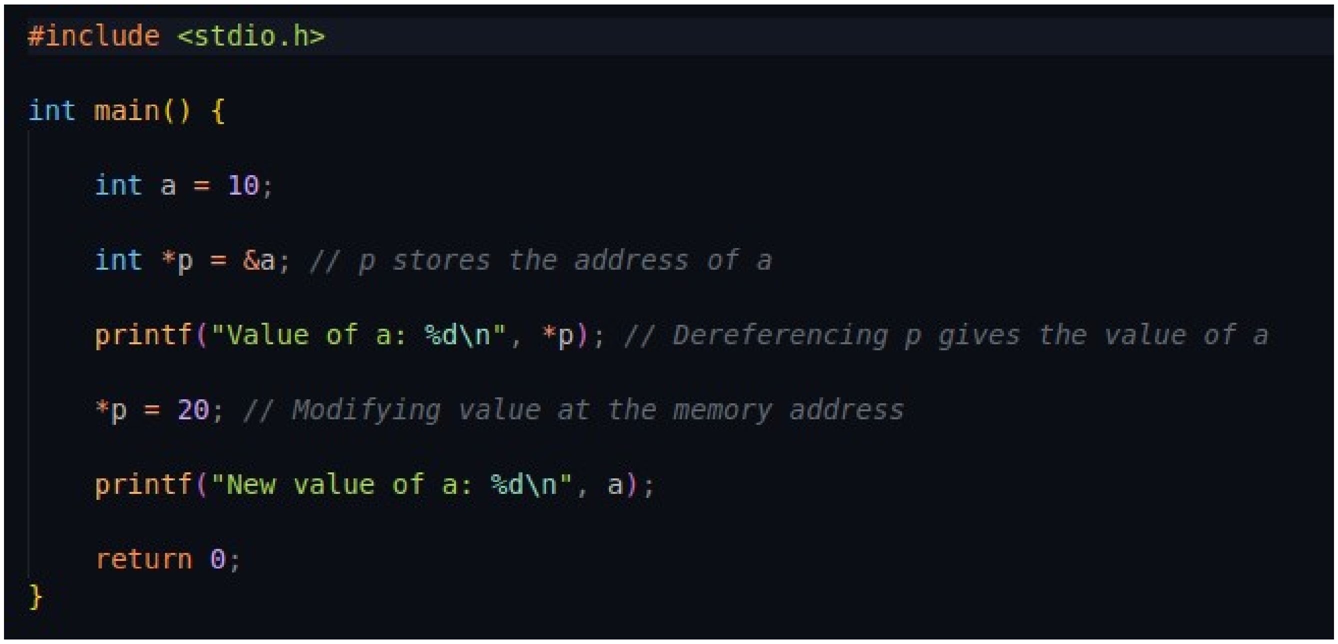
```c
#include <stdio.h>

int add(int x, int y) {

    return (x + y);

}

int main() {

    int x = 5, y = 4;

    printf("%d\n", add(x, y));

    return (0);
}
```

And then, we called the function in our code.

Pointers:

pointers are variables that store the memory address of another variable, allowing for indirect manipulation of data.

The size of a pointer in memory depends on the system architecture, typically 4 bytes on a 32-bit system and 8 bytes on a 64-bit system. Pointers are declared using the `*` symbol, and they can point to any data type, such as `int`, `float`, or `char`. For example, an `int *` points to an integer, while a `char *` points to a character. To use pointers, you first declare the pointer, initialize it with the address of a variable using the ‘`&'` operator, and then access or modify the value at the memory location using the de-reference operator `*`. Pointers are versatile and are commonly used for dynamic memory allocation, function arguments for pass-by-reference, and constructing complex data structures like linked lists and trees. For instance, consider the following example:



```c
#include <stdio.h>

int main() {

    int a = 10;

    int *p = &a; // p stores the address of a

    printf("Value of a: %d\n", *p); // Dereferencing p gives the value of a

    *p = 20; // Modifying value at the memory address

    printf("New value of a: %d\n", a);

    return 0;
}
```

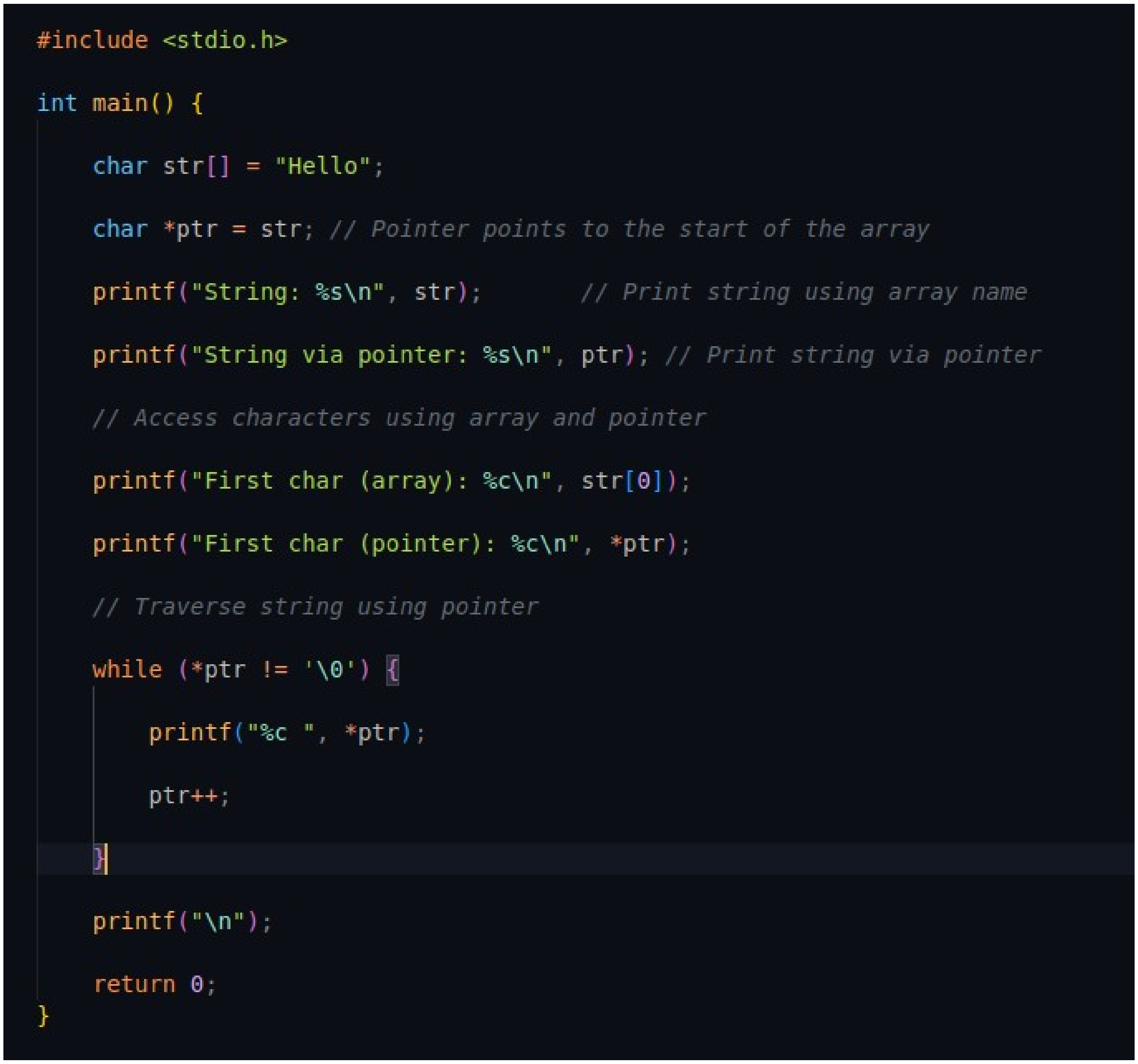
Pointers and Arrays:

In C, pointers and arrays are closely related, especially when dealing with strings, which are essentially arrays of characters terminated by a null character (`\0`).

An array is a collection of elements of the same data type stored in continuous memory locations, allowing you to efficiently manage and access multiple values using a single variable name. Each element in an array is accessed using an index, starting from zero for the first element.

Arrays can be of any data type, such as `int`, `float`, `char`, or even user-defined types like `struct`. They are particularly useful when you need to handle a fixed number of elements, as the size of an array is specified at the time of declaration and cannot be changed. For example, `int arr[5];` declares an integer array of size 5. Arrays are closely related to pointers because the array name acts as a constant pointer to the first element, enabling pointer arithmetic to traverse the array. However, arrays differ from pointers as they are not modifiable value; you cannot reassign an array name to point to another location. Arrays are commonly used for storing and processing collections of data, such as lists of numbers, strings, or matrices, making them an essential feature of the C language.

Here is an example for the above:



```c
#include <stdio.h>

int main() {

    char str[] = "Hello";

    char *ptr = str; // Pointer points to the start of the array

    printf("String: %s\n", str);       // Print string using array name

    printf("String via pointer: %s\n", ptr); // Print string via pointer

    // Access characters using array and pointer

    printf("First char (array): %c\n", str[0]);

    printf("First char (pointer): %c\n", *ptr);

    // Traverse string using pointer

    while (*ptr != '\0') {

        printf("%c ", *ptr);

        ptr++;

    }

    printf("\n");

    return 0;
}
```

Best wishes, Eng. Ali Hassan.