

CL1002
INTRODUCTION TO
COMPUTING

LAB 09
Functions, Recursion and Strings

Learning Objectives

This lab will cover the following topics:

- Functions
- Recursion
- Strings

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:

```
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 `{}`:

```
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 1:

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

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

// Outputs "I just got executed!"
```

Example 2:

```
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  
}
```

Example 1:

```
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
```

Example 2:

```
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.
```

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 1:

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

Example 2:

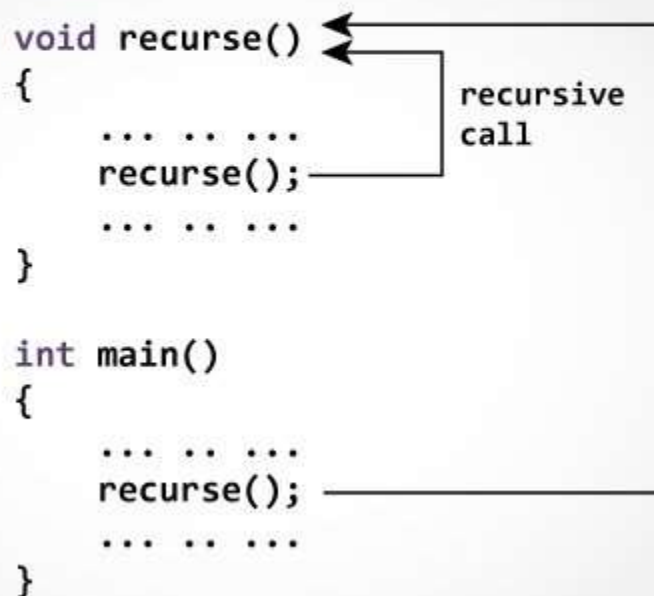
```
int myFunction(int x, int y) {  
    return x + y;  
}  
  
int main() {  
    int result = myFunction(5, 3);  
    printf("Result is = %d", result);  
  
    return 0;  
}  
  
// Outputs 8 (5 + 3)
```

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.

How does recursion work?



```
void recurse()
{
    ... ..
    recurse();
    ... ..
}

int main()
{
    ... ..
    recurse();
    ... ..
}
```

The diagram shows two function definitions. The first is `void recurse()` with a body containing three lines: `... ..`, `recurse();`, and `... ..`. The second is `int main()` with a body containing three lines: `... ..`, `recurse();`, and `... ..`. A horizontal arrow points from the `recurse();` line in `main()` to the `recurse()` line in the `recurse()` function. A second arrow starts from the `recurse();` line inside the `recurse()` function, goes up and then left, pointing back to the `recurse()` line in the `recurse()` function. The label "recursive call" is placed between these two arrows.

The recursion continues until some condition is met to prevent it.

To prevent infinite recursion, if...else statement (or similar approach) can be used where one branch makes the recursive call, and other doesn't.

Example: Sum of Natural Numbers Using Recursion

```
#include <stdio.h>
int sum(int n);

int main() {
    int number, result;

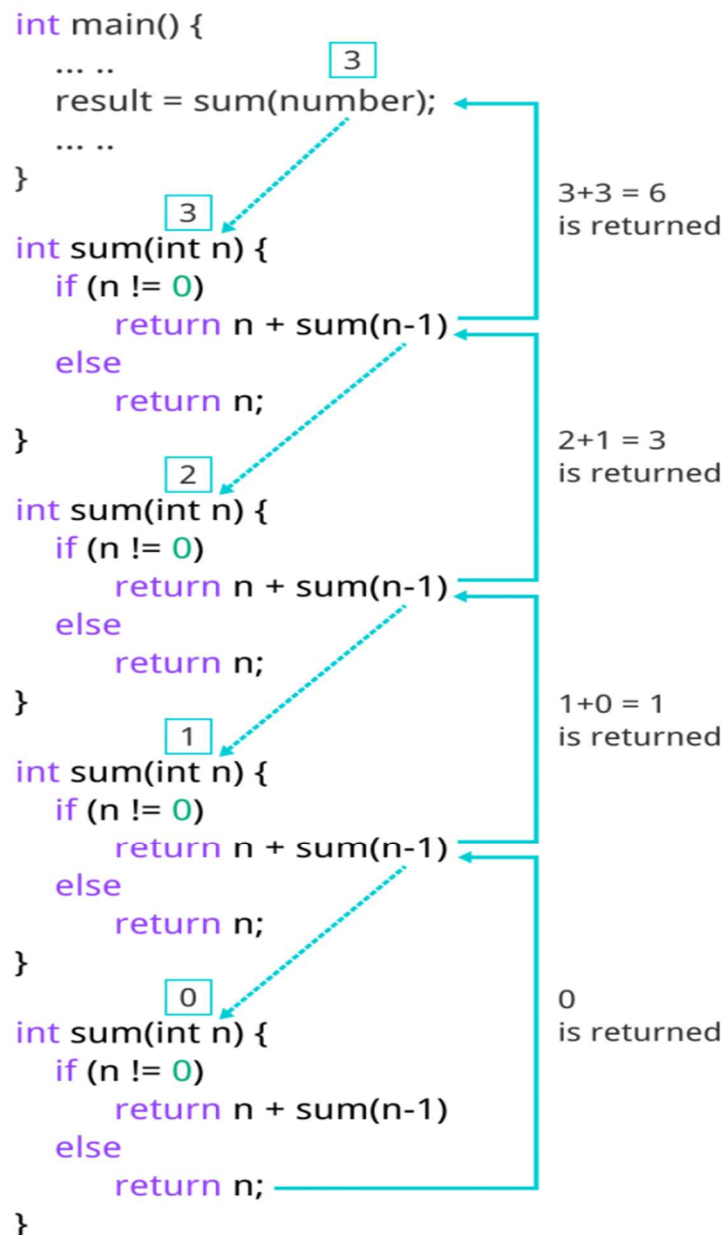
    printf("Enter a positive integer: ");
    scanf("%d", &number);

    result = sum(number);

    printf("sum = %d", result);
    return 0;
}

int sum(int n) {
    if (n != 0)
        // sum() function calls itself
        return n + sum(n-1);
    else
        return n;
}
```

Looks a bit complicated let's break it down



Initially, the `sum()` is called from the `main()` function with `number` passed as an argument.

Suppose, the value of `n` inside `sum()` is 3 initially. During the next function call, 2 is passed to the `sum()` function. This process continues until `n` is equal to 0.

When `n` is equal to 0, the `if` condition fails and the `else` part is executed returning the sum of integers ultimately to the `main()` function.

Strings

Strings are used for storing text/characters.

For example, "Hello World" is a string of characters.

Unlike many other programming languages, C does not have a **String type** to easily create string variables. Instead, you must use the `char` type and create an [array](#) of characters to make a string in C:

```
char greetings[] = "Hello World!";
```

Note that you have to use double quotes ("").

To output the string, you can use the `printf()` function together with the format specifier `%s` to tell C that we are now working with strings:

Example

```
char greetings[] = "Hello World!";  
printf("%s", greetings);
```

Access Strings

Since strings are actually [arrays](#) in C, you can access a string by referring to its index number inside square brackets `[]`.

This example prints the **first character (0)** in **greetings**:

Example

```
char greetings[] = "Hello World!";  
printf("%c", greetings[0]);
```

Modify Strings

To change the value of a specific character in a string, refer to the index number, and use **single quotes**:

Example

```
char greetings[] = "Hello World!";
greetings[0] = 'J';
printf("%s", greetings);
// Outputs Jello World! instead of Hello World!
```

Another Way Of Creating Strings

In the examples above, we used a "string literal" to create a string variable. This is the easiest way to create a string in C.

You should also note that you can create a string with a set of characters. This example will produce the same result as the example in the beginning of this page:

Example

```
char greetings[] = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '!', '\0'};
printf("%s", greetings);
```

Why do we include the `\0` character at the end? This is known as the "null terminating character", and must be included when creating strings using this method. It tells C that this is the end of the string.

The difference between the two ways of creating strings, is that the first method is easier to write, and you do not have to include the `\0` character, as C will do it for you.

You should note that the size of both arrays is the same: They both have **13 characters** (space also counts as a character by the way), including the `\0` character:

Example

```
char greetings[] = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '!', '\0'};
char greetings2[] = "Hello World!";

printf("%lu\n", sizeof(greetings)); // Outputs 13
printf("%lu\n", sizeof(greetings2)); // Outputs 13
```

Strings - Special Characters

Because strings must be written within quotes, C will misunderstand this string, and generate an error:

```
char txt[] = "We are the so-called "Vikings" from the north.";
```

The solution to avoid this problem, is to use the **backslash escape character**.

The backslash (\) escape character turns special characters into string characters:

Escape character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

Example

```
char txt[] = "We are the so-called \"Vikings\" from the north.";
```

One more case is when we insert spaces in the string

```
#include<stdio.h>
#include<string.h>
int main(){
    char string[50];
    scanf("%s",&string);
    printf("\n%s",string);
}
```

If we insert the text Test 123 we get the following output

Test 123

Test

This is because it reads and ignores the new space and interprets it as another input.

To get around this we can use a little hack.

```
#include<stdio.h>
#include<string.h>
int main(){
    char string[50];
    scanf("%[^\n]s",&string);
    printf("\n%s",string);
}
```

Test 123

Test 123

Now the output prints correctly the syntax seems confusing but let's break it down.

Here, [] is the scanset character. ^\n tells to take input until newline doesn't get encountered. Here we used ^ (XOR -Operator) which gives true until both characters are different. Once the character is equal to New-line ('\n'), ^ (XOR Operator) gives false to read the string. So we use "%[^\n]s" instead of "%s". So to get a line of input with space we can go with scanf("%[^\n]s",str);

Exercises

1. **Write C Program to Find the Factorial of a Number Using Recursion**
Input: Enter Number for its factorial = 5
Output: Factorial of Number 5 is = 120
2. **Write C Program to Replace all Occurrences of a character in a String. Note the character must be taken from the user**
Input: Enter String = HELLO
Enter character to be replaced = H
Enter replacement character = J
Output: JELLO
Note: IF there are multiple occurrences replace all of them.
3. **Create a Function that takes two variables and swaps the elements between the variables**
4. **Write C Program to Calculate the Number of Upper-Case Letters, Lower Case Letters and blank spaces in a String**
5. **Given a number and its reverse. Find that number raised to the power of its own reverse.**
Note: As answers can be very large, print the result modulo $10^9 + 7$.

Input:

N = 12

Output: 864354781

Explanation: The reverse of 12 is 21
and 12^{21} , when divided by 1000000007
gives remainder as 864354781.

6. **Write a C Program that can calculate the LCM and GCD of a number (Note use two separate functions for this arrangement in a recursive manner no loops)**

7. Given a 2-D integer array A of order NXN, consisting of only zeroes and ones, find the row or column whose all elements form a palindrome. If more than one palindrome is found, the palindromes found in rows will have priority over the ones found in columns and within rows and columns, the one with lesser index gets the highest priority. Print the palindrome with the highest priority along with its index and indication of it being a row or column.

Input:

N = 3

A[][] = [[1, 0, 0], [0, 1, 0], [1, 1, 0]]

Output: 1 R

Explanation:

In the first test case, 0-1-0 is a palindrome occurring in a row having index 1.

8. The tower of Hanoi is a famous puzzle where we have three rods and N disks. The objective of the puzzle is to move the entire stack to another rod. You are given the number of discs N. Initially, these discs are in the rod 1. You need to print all the steps of discs movement so that all the discs reach the 3rd rod. Also, you need to find the total moves.

Note: The discs are arranged such that the top disc is numbered 1 and the bottom-most disc is numbered N. Also, all the discs have different sizes and a bigger disc cannot be put on the top of a smaller disc.

Input:

N = 2

Output:

move disk 1 from rod 1 to rod 2

move disk 2 from rod 1 to rod 3

move disk 1 from rod 2 to rod 3

3

Explanation: For N=2 , steps will be as follows in the example and total 3 steps will be taken.

9. Given an array of strings sorted in lexicographical order, print all of its permutations in strict lexicographical order.

Input: Enter String Amount = 3

Enter String 1 = ab

Enter String 2 = bc

Enter String 3 = cd

Output: ab bc cd

bc ab cd

cd bc ab

bc cd ab

cd ab bc

ab cd bc