How Functions Work in C++?

Let us first understand in brief what a stack data structure is:

A stack is a way of storing data in a specific order. Imagine a stack of plates. You can only add a new plate to the top of the stack, or take the top plate off the stack. This means that the most recently added plate is the first one to be removed (also known as last in, first out).

In C++, a stack is implemented as a template class, which means that it can store data of any type (for example, integers, strings, or custom classes). You can create a stack and add elements to it using the "push" function, and remove elements from it using the "pop" function. The "top" function can be used to see the element at the top of the stack without removing it.

```
Please note that, this is just to explain you the concept o
f stack data structure.
We would be dealing with stack data structure in details la
ter on in the course.
```

The function call stack is a data structure that is used to store information about the active functions in a program. When a function is called, its information is pushed onto the top of the stack, and when the function returns, its information is popped off from the top of the stack.

The function call stack is important because it allows the program to keep track of which function is currently being executed and which functions have been called. It also enables the program to return to the correct location when a function finishes its execution.

For example, consider the following C++ program:

```
#include <iostream>
using namespace std;

void printMessage() {
  cout << "Hello, world!" << endl;
}

int add(int x, int y) {
  return x + y;
}

int main() {
  int a = 3;
  int b = 4;
  int c = add(a, b); // c will be assigned the value 7
  printMessage();
  return 0;
}
```

Output

```
Hello, world!
```

When this program is executed, the following sequence of events occurs:

1. The **main()** function is called and its information is pushed onto the top of the function call stack.
2. The **add()** function is called and its information is pushed onto the top of the stack.
3. The **add()** function executes and returns a result, which is then assigned to the variable **c**.
4. The **add()** function's information is popped off the top of the stack.
5. The **printMessage()** function is called and its information is pushed onto the top of the stack.
6. The **printMessage()** function executes and prints a message to the screen.
7. The **printMessage()** function's information is popped off the top of the stack.
8. The **main()** function returns and its information is popped off the top of the stack.

In this example, the function call stack is used to store information about the **main()**, **add()**, and **printMessage()** functions as they are executed. When a function is called, its information is pushed onto the top of the stack, and when the function returns, its information is popped of fromf the top of the stack. This allows the program to keep track of which function is currently being executed and to return to the correct location when a function finishes its execution.

The function call stack is managed by the compiler and is not visible to the programmer. However, it is an important part of how functions works in C++ and plays a crucial role in the execution of a program.