0.1 C++ and memory management

In this lesson, you will learn how C++ manages memory, and you will gain understanding of how memory is managed inside your computer. Understanding these concepts is very important for programmers for several reasons:

- Allows you to optimize your program, avoiding to allocate useless memory, or accessing it in the right way.
- Creates awareness that allocating memory is slow and expensive.
- Understand what goes under the hood in dynamic programming languages.
- Can avoid security issues.
- Learn our first data structure!

0.1.1 Memory stack

Every process inside a computer has a certain amount of memory, which is granted to the process by the operative system. Some of that memory will be reserved to keep some information about the process. That memory cannot be accessed, and is reserved to the operative system. Then, we have that is called the **stack**.
The stack is used in order to keep track of the location where we are inside our process. Imagine that you have a certain number of tasks that you need to do. A task may require you to have to complete a different task before you may be able to solve the previous one, and for every task you need to store some information, relative to the task. An excellent way to manage this information is to use a pipe. For every task that you receive we will push it on top our pile, along with the informations that we need in order to complete the task. If the task will require some other task to be completed, then we will begin working on that task, and push it on top of our pipe. Once a task is completed, then we remove it from the top of our pile. When, the pile will be empty again, then our program will be over!
In computers programs, things work in the same way. Instead of tasks we have functions, and our pipe is what we refer to as a stack, an abstract data structure. The information that we need in our process are variables, that we need to keep track of in order to complete our tasks. Lets take a look at an example:

```
1 int task ( int sum )
2 {
```

```
3   int total = 0;
4   for ( int i =0; i < sum; i++ )
5     total +=i;
6   return total;
7 }
8
9 int main ( int argc, const char * argv[] )
10 { //  This is the entry point of our program.
11   int sum = 5;
12   int total = task( sum );
13   total    += task( sum ); // += will add to the existing value
14   return 0;
15 }
```

Now, lets take a look at what happens inside our stack when we execute the program. We start with main inside the stack. Inside the space for main, there will be enough space for the variables *sum*, and *total*. On line *13*, the variable sum will assume the value 5. Then, the function *task* will be called.
New space will be asked for on the stack, so that we can fit the variables that we have inside *task*. The value of *sum* will be copied over, and the function *task* will have its own copy. Likewise, *task* will have its own version of *total*. These are **local variables**, so they only exist in the scope of the function. You cannot access from *task* the local variables of *main*.

0.1.2 Pointers and reference

0.1.3 Allocating memory in C++