Variable Addresses in the Stack

Let's take a look at how variables in C++ are stored in memory.

WE'LL COVER THE FOLLOWING

Structure of the Stack

Suppose we declare a simple integer called `var`:

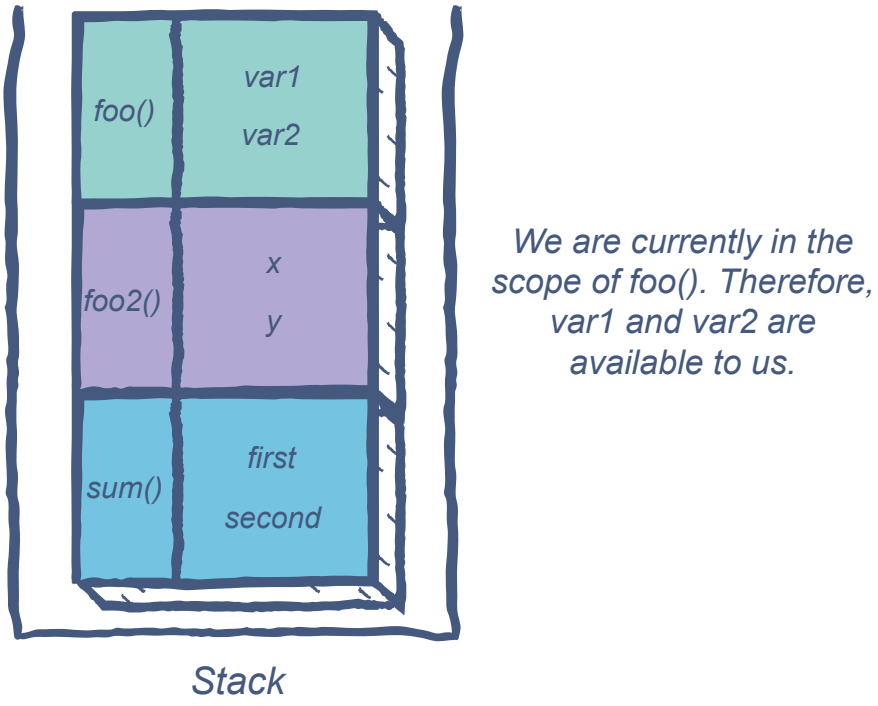
```
int main(){
  int var = 10;
}
```

This reserves memory for `var` in the **stack**. In C++, the stack represents the static section of the RAM. It stores different functions in a stack form (the most recently called function at the top) along with all the variables being used in them.

Whenever a function is freed, it's space in the memory is released and those variables are lost forever. Because of this, variables like `var` are "local" in nature.

Stack variables are alotted memory at compile time and there is a restriction to the amount of space that can be reserved for a variable.

The members of a function can only be accessed if the said function is at the top of the stack (the function being executed).



In the illustration above, `foo()` is at the top of the stack, implying that it is the function being executed right now. As soon as it finishes, it is popped out of the stack and `foo2()` becomes available to us. This is how the Stack handles function scopes.

Accessing the Address of a Variable

Each variable in the stack is stored at a specific address. The address of a variable can be accessed using the `&` operator:

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

int main() {
  int var = 10;
  cout << &var;
}
```

As we can observe, the name of the address doesn't make much sense, but it is useful in the concept of pointers.

useful in the concept of pointers.

We'll discuss this further in the next lesson.