https://youtu.be/PlBDxf5ujyo

Properties of Stack Memory

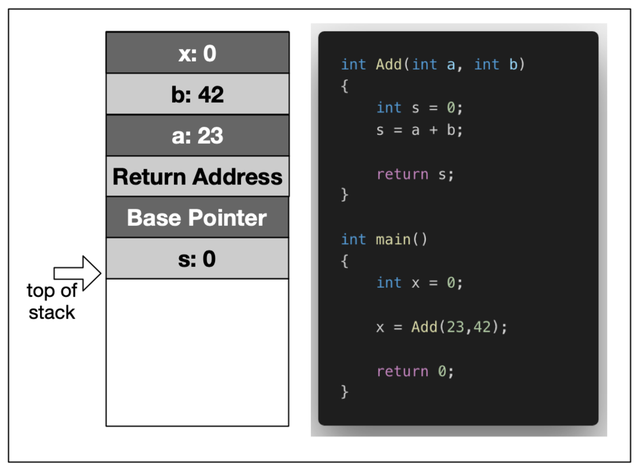
In the available literature on C++, the terms *stack* and *heap* are used regularly, even though this is not formally correct: C++ has the *free space*, *storage classes* and the *storage duration* of objects. However, since stack and heap are widely used in the C++ community, we will also use it throughout this course. Should you come across the above-mentioned terms in a book or tutorial on the subject, you now know that they refer to the same concepts as stack and heap do.

As mentioned in the last section, the stack is the place in virtual memory where the local variables reside, including arguments to functions. Each time a function is called, the stack grows (from top to bottom) and each time a function returns, the stack contracts. When using multiple threads (as in concurrent programming), it is important to know that each thread has its own stack memory - which can be considered thread-safe.

In the following, a short list of key properties of the stack is listed:

1. The stack is a **contiguous block of memory**. It will not become fragmented (as opposed to the heap) and it has a fixed maximum size.
2. When the **maximum size of the stack** memory is exceeded, a program will crash.
3. Allocating and deallocating **memory is fast** on the stack. It only involves moving the stack pointer to a new position.

The following diagram shows the stack memory during a function call:



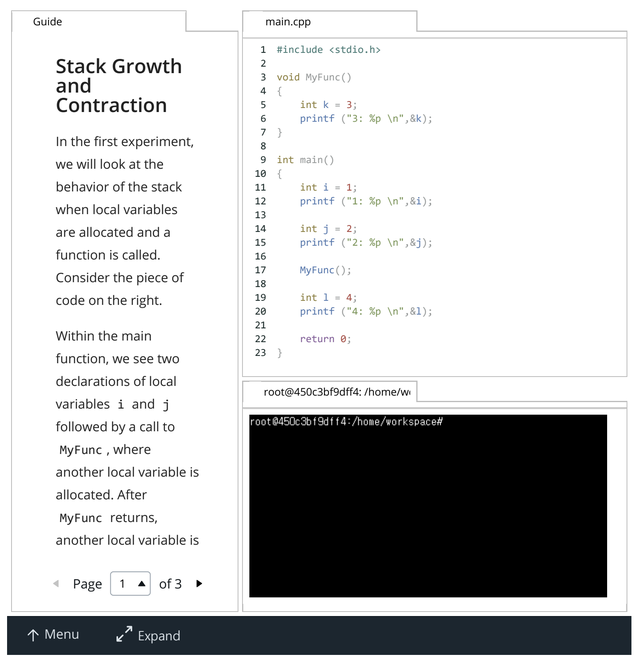
In the example, the variable `x` is created on the stack within the scope of main. Then, a stack frame which represents the function `Add` and its variables is pushed to the stack, moving the stack pointer further downwards. It can be seen that this includes the local variables `a` and `b`, as well as the return address, a base pointer and finally the return value `s`.

In the following, let us dig a little more deeply and conduct some experiments with variables on the stack.

Stack Growth and Contraction

https://youtu.be/W62TL4_NhEs

> Note: Click "Expand" at the bottom left of the workspace below for better readability and overall workspace experience.



Outro

https://youtu.be/gXdpjZiL7m8

Before we take a look at the heap memory in the next lesson, let us briefly revisit the principles of call-by-value and call-by-reference with regard to stack usage.