Memory Management

This lesson discusses two memory management areas i.e. stack and heap.

We'll cover the following

In many programming languages, there is no need to bother where the memory is allocated. But in system programming languages like Rust, a program's behavior depends upon the memory being used, i.e., stack or heap.

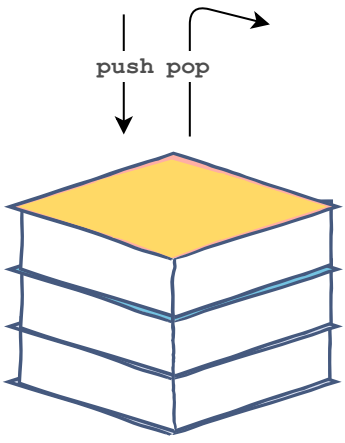
Stack

> When the size of data is known at **compile-time**, a stack is used for storage.

What Is a Stack?

A stack is a Last in First Out(LIFO) data storage memory location meaning all values are stored in a last in first out order.

Let's imagine a *real-life analogy* to understand stack. There is a huge pile of books with a wall all the way around



pile of books

it. You want the one in the middle. You can't just slip one from the middle. So, you'll remove items until the desired location is reached. Inserting values onto the stack is a *push* operation and removing values from the stack is a *pop* operation.

Example

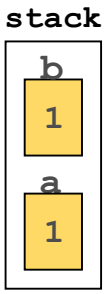
All **primitive data types** that have a fixed size are stored on a stack.

Illustration

The following example has a variable `a` and a variable `b`. Both are initialized to `1` and stored on the stack.

> **Note:** In the example below, variable `b` is stored on the top of variable `a` in a LIFO order.

```
let a = 1;
let b = 1;
```



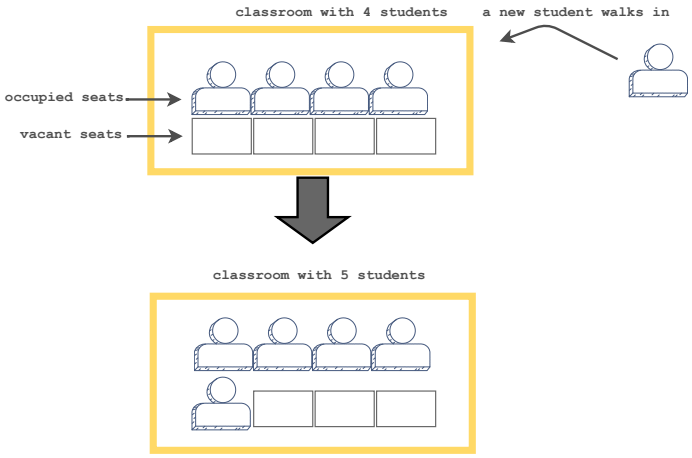
Heap

> When the size of data is not known at compile time rather it is known at the **run time**, it goes in a portion of program memory called a heap.

What Is a Heap?

Heap is a big data storage and stores values whose size is unknown at compile time. The operating system

allocates a space in the heap that is adequate, marks it as in use, and returns a pointer, which is the address of that location.



Let's imagine a *real-life analogy* to understand heap. Suppose there are 4 students in a class and the classroom has the accommodations for 8. One new student gets enrolled and is directed to the classroom. Now the total students in the class becomes 5.

Example

Since **Vectors** and **String Objects** are resizable, they are stored in a heap.

Illustration

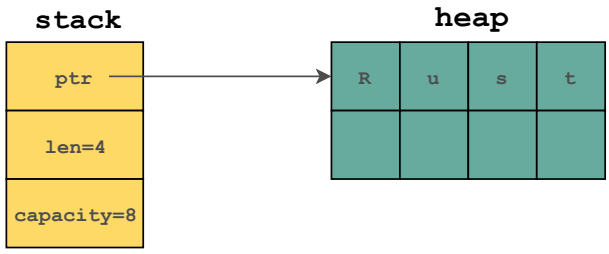
A String is made up of three parts:

- A pointer to the memory that holds the contents of the string
- Length
- Capacity

This group of the information above is stored on the stack.

The memory on the heap holds the value assigned to the string. Below is the example of string object `str` which is initialized to `Rust`.

```
let str=from::String("Rust");
```



Here, `ptr` is a pointer to the base address of the string str. `len` is the total length of the string in bytes and `capacity` is the total amount of memory that the operating

system has provided to the string.

Stack vs. Heap

- **Pushing** values on to the stack is much easier than heap since the operating system does not have to find a big space in memory and mark the pointer for the next allocation.
- **Accessing** data from the stack is much faster than heap since the processor will take less time to access the data that is closer to the other data.

Quiz

Test your understanding of stack and heap.

Quick Quiz on Memory Management!

1 Primitive variables ae stored on:

2 Non-primitive variables are stored on:

3 Which of the following allows unknown data size storage?

4 The known data size is stored on:

Retake Quiz

Now that you have learned about memory management, let’s see how Rust ensures memory safety using ownership in the next lesson.