# Everything You Wanted to Know About the C# Stack and Heap

Understanding the C# Stack and Heap and how it’s use is something every programmer should know. It’s also asked a lot in interviews. This post I will try and answer some of those questions.

## What is the Stack?

In the C# runtime the stack is normally referred to as the “Call Stack”. It is an implementation of the stack data structure. Its purpose is to keep track of which method is currently executing. The item in the top of the stack is the currently executing method. When a C# program runs and executes a method all local variables in that method are grouped together and pushed onto the stack. This is called the “Stack Frame”.

## What is the Heap?

In the C# runtime the heap is normally referred to as the “Managed Heap”. It is an implementation of the heap data structure. Its purpose is to store data which outlives data enclosed in a single method. The heap is used to store reference type variables, which are referred to as objects.

## What are the differences between the stack and the heap?

* + The stack is an efficient place to store small amounts of short-lived data.
  + The stack is generally faster than the heap.
  + The heap is a large block of memory reserved for instances of reference types.
  + The heap is garbage collected.
  + Memory on the bottom of the stack always lives longer than the memory on the top of the stack
  + The stack is strictly ordered.
  + The heap is not ordered.

## How does something end up on one vs the other?

How an item ends up on the stack vs the heap is determined by the type of variable.

* + Local reference type variables have references stored on the stack.
  + Local value type variables have actual values stored on the stack.
  + Objects which are of reference types are stored on the heap.
  + Instance variables that are part of a reference type instance (e.g. a field on a class) are stored on the heap with the object itself.
  + Static variables always live on a heap.

## Which one offers better performance and why?

The stack offers better performance because the choice of the stack data structure guarantees that the memory on the bottom of the stack always lives longer than the memory on the top of the stack and that the stack is strictly ordered. This means that the stack will NEVER have memory holes, thus there is much less overhead when allocating and deallocating items on the stack. The choice of the heap data structure means that objects can live anywhere in the allotted memory chunk, and that over time holes can develop between objects. Managing these holes and arranging and re-arranging the heap memory is more expensive.