Factor	Stack	Heap
Location	RAM	RAM
Who is in charge	Programmer	OS
Size *(to be determined by OS, Compiler, Runtime Environment)*	Fixed	Variable (OS can add more if needed)
Memory deallocation	Automatic	Manual using *free ( )*
Memory preserve	Temporary	Permanent
Access Speed	Fast using stack pointers	Slow we need OS interventions
Possible Problems	Stack Overflow	Memory Leaks and Fragmentation

Stack

- Stored in computer RAM just like the heap.
- Variables created on the stack will go out of scope and are automatically deallocated.
- Much faster to allocate in comparison to variables on the heap.
- Implemented with an actual stack data structure.
- Stores local data, return addresses, used for parameter passing.
- Can have a stack overflow when too much of the stack is used (mostly from infinite or too deep recursion, very large allocations).
- Data created on the stack can be used without pointers.
- *You would use the stack if you know exactly how much data you need to allocate before compile time, and it is not too big.*
- Usually has a maximum size already determined when your program starts.

Heap

- Stored in computer RAM just like the stack.
- In C, variables on the heap must be destroyed manually and never fall out of scope. The data is freed with `free`.
- Slower to allocate in comparison to variables on the stack.
- Used on demand to allocate a block of data for use by the program.
- Can have fragmentation when there are a lot of allocations and deallocations.
- In C, data created on the heap will be pointed to by pointers and allocated with `malloc` respectively.
- Can have allocation failures if too big of a buffer is requested to be allocated.
- *You would use the heap if you don't know exactly how much data you will need at run time or if you need to allocate a lot of data.*
- Responsible for memory leaks.

What is the difference between a stack and a heap?

The differences between the stack and the heap can be confusing for many people. Therefore, we thought we would have a list of questions and answers about stacks and heaps that we thought would be very helpful.

Where is the stack and heap stored?

They are both stored in the computer's RAM (Random Access Memory).

How long does memory on the stack last versus memory on the heap?

Once a function call runs to completion, any data on the stack created specifically for that function call will automatically be deleted. Any data on the heap will remain there until it's manually deleted by the programmer.

Can the stack grow? Can the heap grow?

The stack is set to a fixed size and cannot grow past its fixed size (although some languages have extensions that do allow this). So, if there is **not** enough room on the stack to handle the memory being assigned to it, a ***stack overflow*** occurs. This often happens when a lot of nested functions are being called, or if there is an infinite recursive call.

If the current size of the heap is too small to accommodate new memory, then more memory can be added to the heap by the operating system. This is one of the big differences between the heap and the stack.

How are the stack and heap implemented?

The implementation really depends on the language, compiler, and run-time – the ***small*** details of the implementation for a stack and a heap will always be different depending on what language and compiler are being used. But, in the big picture, the stacks and heaps in one language are used to accomplish the same things as stacks and heaps in another language.

Which is faster – the stack or the heap? And why?

The stack is much faster than the heap. This is because of the way that memory is allocated on the stack. Allocating memory on the stack is as simple as moving the stack pointer up.

How is memory deallocated on the stack and heap?

Data on the stack is ***automatically*** deallocated when variables go out of scope. However, in languages like C and C++, data stored on the heap has to be deleted **manually** by the programmer using one of the built in keywords like **free, delete, or delete [ ].** Other languages like Java and .NET use garbage collection to automatically delete memory from the heap, without the programmer having to do anything.

What can go wrong with the stack and the heap?

If the stack runs out of memory, then this is called a *stack overflow* – and could cause the program to crash. The heap could have the problem of *fragmentation*, which occurs when the available memory on the heap is being stored as noncontiguous (or disconnected) blocks – because *used* blocks of memory are in between the *unused* memory blocks. When excessive fragmentation occurs, allocating new memory may be impossible because even though there is enough memory for the desired allocation, there may not be enough memory in one big block for the desired amount of memory.

Which one should I use – the stack or the heap?

For people new to programming, it's probably a good idea to use the stack since it's easier.
Because the stack is small, you would want to use it when you know exactly how much memory you will need for your data, or if you know the size of your data is very small. It's better to use the heap when you know that you will need a lot of memory for your data, or you just are not sure how much memory you will need (like with a dynamic array).

How do threads interact with the stack and the heap? How do the stack and heap work in multithreading?

In a multi-threaded application, each thread will have its own stack. But all the different threads will share the heap. Because the different threads share the heap in a multi-threaded application, this also means that there must be some coordination between the threads so that they don't try to access and manipulate the same piece(s) of memory in the heap at the same time.

What is the initial size with which a stack/heap is created? And who decides it?

This is compiler- and OS-specific.

- On windows using Visual Studio, default stack size is 1MB
- On Linux the following command can show show your current one: **`ulimit -s or -a`**
- On my Linux mint 64 bit it shows 8192 KB.

Wherein physical memory are they are created? I see a general description as "Heap is created in the top-level-address and stack at the low-level-address".

This is compiler- and OS-specific.

Really. The language standard does not mandate the minimum stack size nor specifies the location of either the stack or the heap in memory. And the reason for that is to make C programs less dependent on these details and therefore more portable to different platforms (read: different OSes, different CPUs, different compilers).