1-When should we use heap and stack ?

-Heap : When we need to keep the variable around for a long time, if we need big structures, large block of memories, if we need to change dynamically the size of the structure or shrink it.

-Stack : When er need to store variables for a short amount of time (while the function is alive for example), small sized variables, because the stack is easier to use and faster for this kind of things.

2- Memory Layout :

**-Text Segment:**

**Section of a program in a object file which contains executable instruction.**

**Generally , it is placed below the heap or the stack to avoid overflows to override it. Text segment is a read only segment.**

- **Initialized Data Segment:**  
A data segment is a portion of virtual address space of a program, which contains the global variables and static variables that are initialized by the programmer.

Note that, data segment is not read-only, since the values of the variables can be altered at run time.

This segment can be further classified into initialized read-only area and initialized read-write area.

**–BSS:** BSS segment also refers to “Uninitialized data segment”. Data in this segment is initialized by the OS kernel to arithmetic 0 before the program starts executing.  
Typically, this segment starts at the end of the data segment and contains all global and static variables that are initialized to zero or do not have explicit initialization in source code. For instance, a variable declared as static int i; would be allocated to the BSS segment.  
Finally, the BSS segment is Read-Write.

**– Stack:** The stack space is located just under the OS kernel space, generally opposite the heap area and grows downwards to lower addresses. ( it may grow the opposite direction on some other architectures )The stack is LIFO ( last-in-first-out ) data structure. In computer science, a stack is an abstract data type that serves as a collection of elements, with two principal operations:This area is devoted to storing all the data needed by a function call in a program. Calling a function is the same as pushing the called function execution onto the top of the stack, and once that function completes, the results are returned popping the function off the stack. The dataset pushed for function call is named a stack frame, and it contains the following data.

**– Heap:** Heap is the segment where dynamic memory allocation usually takes place, i.e., to allocate memory requested by the programmer for variables whose size can be only known at run-time and cannot be statically determined by the **compiler** before program execution. The heap area begins at the end of the BSS segment and grows upwards to higher memory addresses. It is managed by malloc/new, free/delete, which may use the brk and sbrk **system calls** to adjust its size.  
This area is shared by all shared libraries and dynamically loaded modules in a process.

0x0000000100000f50 <main+0>: push %rbp

0x0000000100000f51 <main+1>: mov %rsp,%rbp

The first two instructions are called the function prologue or preamble. First we push the old base pointer onto the stack to save it for later. Then we copy the value of the stack pointer to the base pointer. After this, %rbp points to the base of main‘s stack frame.

In my own words : we first save a copy of register base pointer to the stack, then the register stack pointer value is saved into rbp.

rbp is the frame pointer on x86\_64. In your generated code, it gets a snapshot of the stack pointer (rsp) so that when adjustments are made to rsp (i.e. reserving space for local variables or pushing values on to the stack), local variables and function parameters are still accessible from a constant offset from rbp.

Objdump : displays informations about one or more object files. There are many flags to diplays particular infos.

Stack and heap are runtime concepts, which have no pre-baked segment in the executable. Since they are just scratch space used dynamically, there's no reason why they should be present in the executable; they are created automatically at process (heap) or thread (stack) creation.

Where are stored each variable :

<https://stackoverflow.com/questions/14588767/where-in-memory-are-my-variables-stored-in-c/14588866>

size : size a.out

This gives you the size of the [text segment](http://en.wikipedia.org/wiki/Code_segment) (also known as code segment), [data segment](http://en.wikipedia.org/wiki/Data_segment), [block started by symbol](http://en.wikipedia.org/wiki/.bss). dec is the size of the text, data and bss size added together. hex is the same number in hexadecimal.