ESP_LEC_5

Compile time refers to the time duration in which the programming code is converted to the machine code (i.e binary code) and usually occurs before runtime. The operations performed at compile time usually include syntax analysis, type checks and code generation. Programming language definitions usually specify compile time requirements that source code must meet to be successfully compiled.

Run time is the final phase of a computer program's life cycle, in which the code is being executed on the computer's central processing unit (CPU) as machine code.

Static memory allocation

Static Allocation means, that the memory for your variables is allocated when the program starts. The size is fixed when the program is created. It applies to global variables, file scope variables, and variables qualified with static defined inside functions.

Is allocated at compile time, and the lifetime of a variable in static memory is the lifetime of the program. The allocation of static memory is handled by the compiler rather than allocated at compile time.

Automatic memory allocation

This is what is commonly known as 'stack' memory, and is allocated when you enter a new scope (usually when a new function is pushed on the call stack). Once you move out of the scope, the values of automatic memory addresses are undefined, and it is an error to access them.

int a = 43;

Note that scope does not necessarily mean function. Scopes can nest within a function, and the variable will be in-scope only within the block in which it was declared.

Dynamic memory allocation

Is memory allocated at runtime using calloc(), malloc(.

You now control the exact size and the lifetime of these memory locations. If you don't free it, you'll run into memory leaks, which may cause your application to crash, since at some point of time, system cannot allocate more memory.

Use dynamic in the following situations:

1. When you need a lot of memory.
2. When the memory must live after the function returns. Stack memory gets destroyed when function ends, dynamic memory is freed when you want.
3. When you're building a structure (like array, or graph) of size that is unknown (i.e. may get big), dynamically changes or is too hard to precalculate. Dynamic allocation allows your code to naturally request memory piece by piece at any moment and only when you need it.
4. This also ensures you allocate exactly as much memory as you really need