1. **Global Variables**:
    - **Linkage**: Global variables have external linkage by default, meaning they can be accessed from other translation units (source files). This allows multiple files to share and modify the same variable.
2. **Static Variables**:
    - **Linkage**: Static variables have internal linkage when declared at the file level (i.e., outside of any function). This means they are only accessible within the file they are defined in and cannot be accessed from other translation units, even if they have the same name.
    - **Lifetime**: Both global and static variables maintain their value throughout the program's execution.

In terms of storage, both global and static variables are typically placed in the data segment, but the static variables are isolated to their defining file, preventing name conflicts across different files. This is particularly useful for encapsulating data within a single module while still retaining persistent state.

Yes, that's correct! In C, local (automatic) variables are typically allocated on the stack. However, when the compiler optimizes the code, it may decide to allocate certain variables in registers instead of on the stack. Here's a breakdown:

1. **Stack Allocation**:
    - Local variables are generally allocated on the stack, which allows for quick allocation and deallocation as functions are called and return.
    - When a function is called, space for local variables is created on the stack, and this space is freed when the function exits.
2. **Register Allocation**:
    - The compiler can optimize performance by keeping frequently accessed variables in CPU registers instead of on the stack.
    - If a variable is small and used often, the compiler might allocate it to a register. This is faster than accessing the stack, but registers are limited in number.
3. **Spilling**:
    - If there are not enough registers available (due to many variables or other constraints), the compiler may "spill" a variable to the stack. This means that it will store the variable's value in stack memory instead of keeping it in a register.

So, while local variables are typically on the stack, optimizations can lead to them being held in registers for performance gains, provided they don't "spill" due to register limitations.