Scope of Variables...
Lifetime (duration) of Variables
Duration (lifetime) of Variables

- Period during which an identifier exists in memory.
- Some identifiers exist briefly.
- Some are repeatedly created and destroyed.
- Others exist for the entire execution of a program.
Lifetime of Variables in C

- The **lifetime (duration)** of an identifier is the **time** in which memory is reserved while the program is executing.

- **Automatic**
- **Static**
- **Dynamic**
1) Automatic Lifetime

- When you call the function or enter a block, memory will be **allocated** for all local variables defined inside the function (**entering the scope**).

- Finally memory will be **deallocated** when the function exits (**exiting the scope**).

- The variable is “alive” while the block/function is executing [**automatic lifetime**]
Automatic Lifetime

```c
int functionA(int a) {
    int b;
    ...
    ...
    return 0;
}

int main() {
    functionA();
    return 0;
}
```

The variable `b` is **created** when the function is called.

The variable `b` is **destroyed** when its scope is left during the execution of a program.
2) Static Lifetime

Local variables declared with the `static` keyword:

```plaintext
variable_type static variable_name;
```

```plaintext
int static age;    double static income;
```

- "Permanent" storage duration
  - Here "permanent" means as long as the program still runs in computer's memory
- Occupies same memory location throughout
- Does not lose its value
Example: Static Local Variables

```c
#include <stdio.h>

int nextNumber () {
    int static current = 0;
    current++;
    return current;
}

int main()
{
    printf("The current number is: %d\n", nextNumber());
    printf("The current number is: %d\n", nextNumber());
    printf("The current number is: %d\n", nextNumber());
}
```
Example: Static Local Variables

Every static variable is initialized only once!!!
Duration versus Scope of Variables

A variable with **static** lifetime and **local** scope!
Duration versus Scope of Variables

A variable with **static lifetime and local scope**!
3) Dynamic Lifetime

- We dynamically request the memory or release it for a variable (on demand)!

- The lifetime of a dynamic object begins when memory is allocated for the object (e.g., by a call to malloc()) and ends when memory is deallocated (e.g., by a call to free()). Dynamic objects are stored in "the heap".

- Dynamic memory will be covered when we learn about pointers.