Memory Management

There are three memory management types in C:

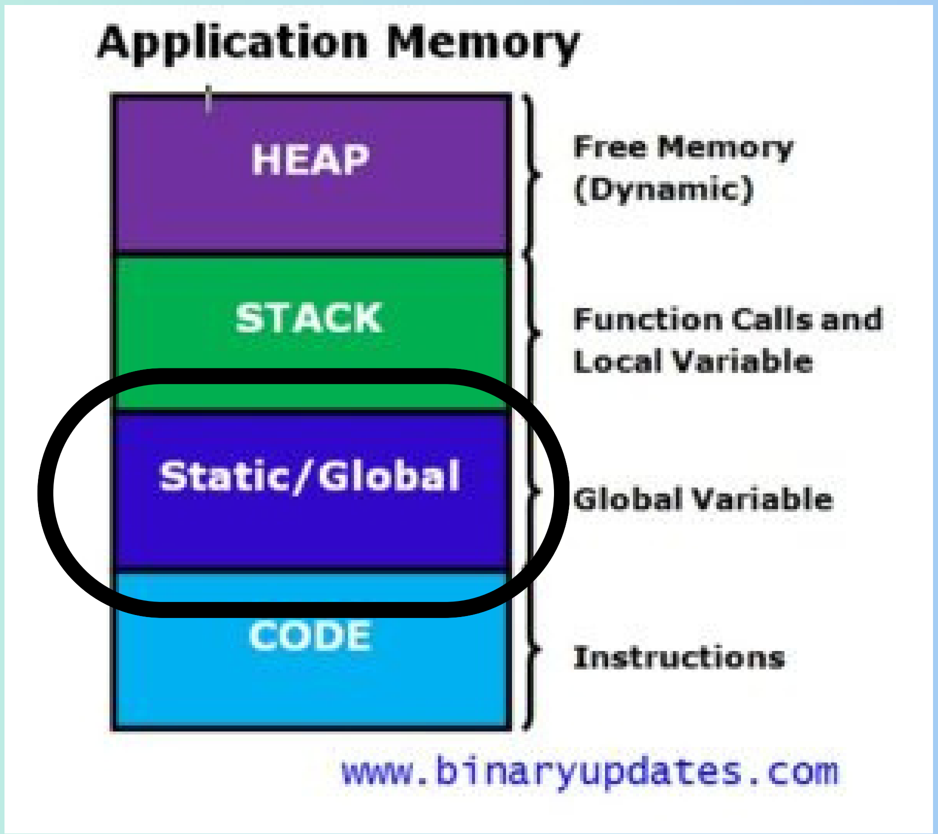
- Static
- Automatic
- Dynamic

Next Slide

1. Static Memory Allocation

In C, static memory allocation occurs at compile time. Variables with static storage (e.g., global or static variables) persist for the lifetime of the program. They are initialized once and reside in the data segment of memory.

- Use Case: Declaring fixed-size arrays or variables that retain their value across function calls.
- Downside: Lack of flexibility; the size is fixed at compile time.

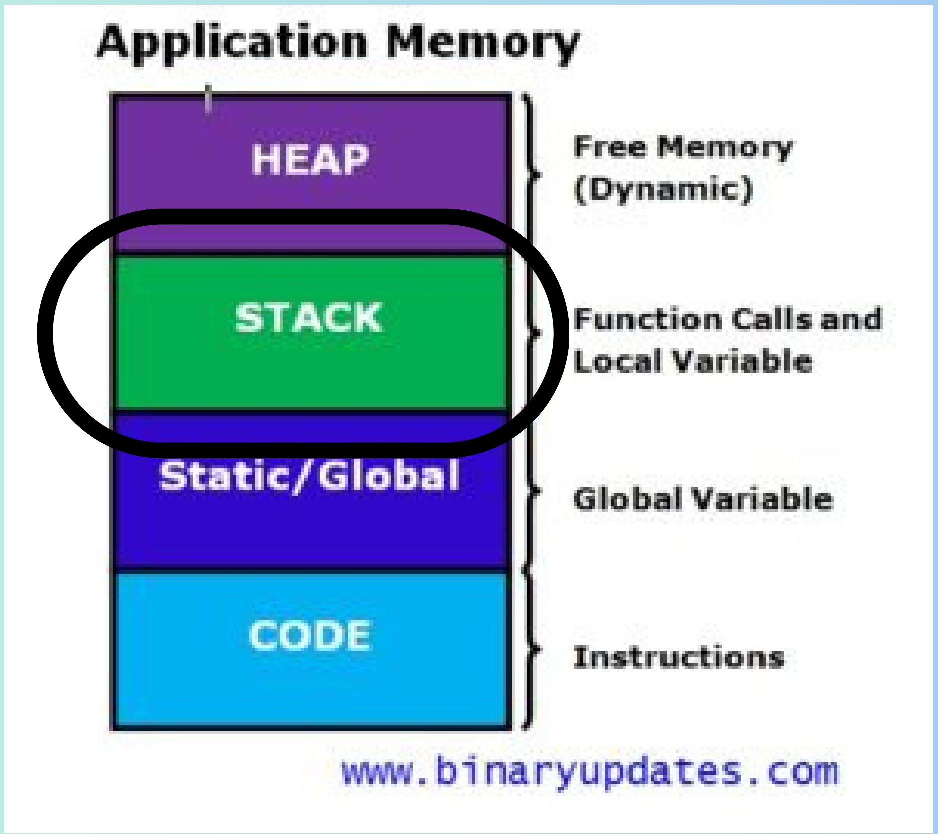


Next Slide

2. Automatic Memory Allocation

Also known as stack allocation, this occurs at function scope—memory for local variables is allocated when a function is called and deallocated when the function exits.

- Use Case: Local variables in functions.
- Downside: Limited by stack size and unsuitable for large, persistent data.

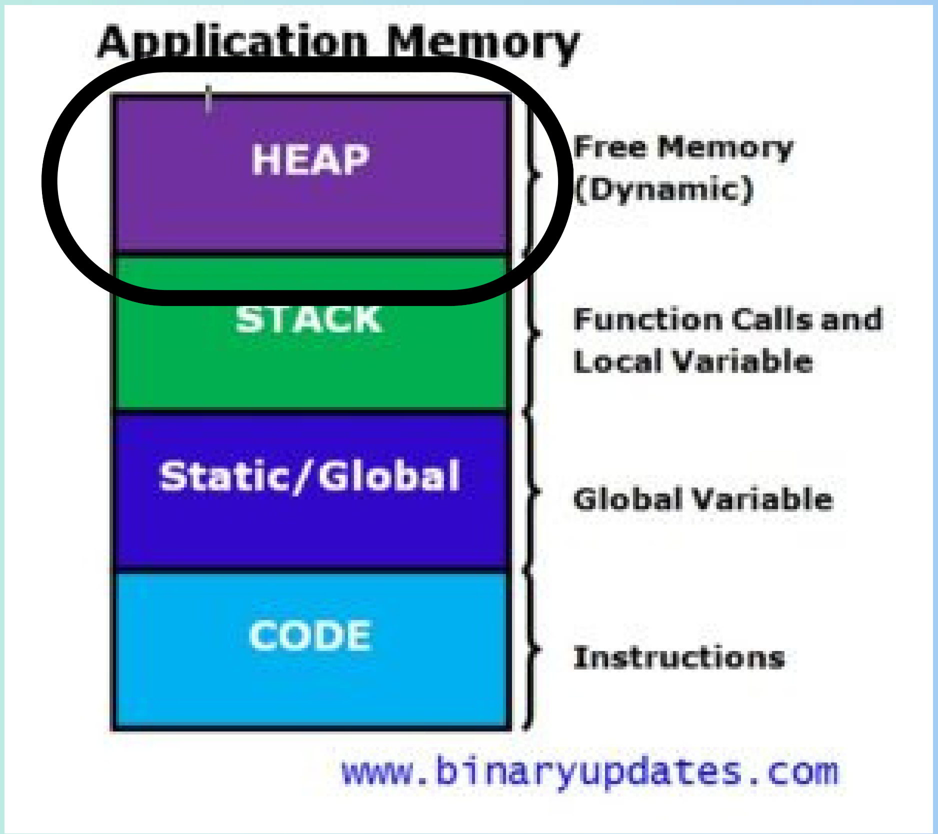


Next Slide

3. Dynamic Memory Allocation

This happens at runtime, where memory is requested from the heap using functions like malloc() and free(). It offers flexibility but requires manual management.

- Use Case: When memory size is unknown at compile time (e.g., dynamic arrays).
- Downside: Risk of memory leaks and fragmentation if not handled properly.



Next Slide

What About Python?

Python abstracts away these types of memory management through its built-in memory manager. It uses C dynamic memory allocation, but the underlying mechanisms differ:

- **Reference Counting**: Python keeps track of object references, and memory is automatically reclaimed when no references are left.
- **Garbage Collection**: Python also uses a garbage collector to handle cyclic references, which reference counting alone cannot manage.

Next Slide

Bonus Tip

Use Immutable Variables for Better Memory Efficiency.

Whenever possible, it’s a good idea to use immutable variables (like tuples) instead of mutable ones (like lists or dictionaries).

Immutable objects are untracked by the garbage collector, meaning Python doesn't have to monitor them, which can lead to more efficient memory usage.

Plus, since they can’t be altered, they are often reused internally, reducing the need for extra allocations.