Unit 3: Run Time Environment

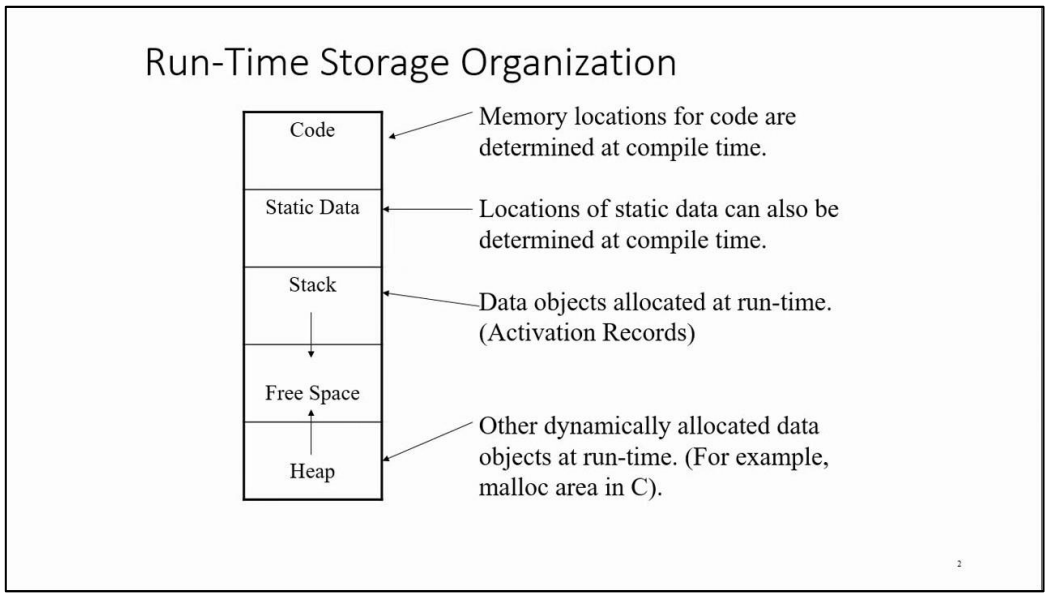
3.1. Source Language Issues

Source languages (like C, Java, Python) influence the structure of the run-time environment. Key aspects include:

- Type Checking: Statically typed (e.g., C, Java) vs. dynamically typed languages (e.g., Python).
- Recursion: Requires stack-based memory management.
- Scoping Rules: Determines how variable bindings are resolved.
    - Static (Lexical) Scope: Scope determined at compile-time.
    - Dynamic Scope: Scope determined at run-time (rare in modern languages).
- Parameter Passing Mechanisms: Affects how arguments are handled at run-time.

3.2 Storage Organization

Memory layout in a run-time environment is typically divided into several segments:



Storage Allocation Strategies

There are three main types of storage allocation:

1. Static Allocation
    - Memory is allocated at compile-time.
    - Used for global and static variables.
    - Size and lifetime are known in advance.
    - Fast access, but not flexible.

2. Stack Allocation
    - Memory allocated/deallocated in LIFO order.
    - Used for local variables and function calls.
    - Managed via stack frames (activation records).
    - Efficient but lacks dynamic flexibility.
3. Heap Allocation
    - Memory allocated/deallocated dynamically at run-time.
    - Used for objects/data whose lifetime isn't known in advance.
    - Requires garbage collection or manual memory management.

Storage Allocation in C

C uses a combination of all three strategies:

- **Static Allocation**: Global/static variables.
- **Stack Allocation**: Function parameters, local variables.
- **Heap Allocation**: malloc(), calloc(), realloc() for dynamic memory; must use free() to release memory.

```
int x = 5;              // Static
void func() {
    int y = 10;         // Stack
    int* z = malloc(10*sizeof(int)); // Heap
    free(z);
}
```

3.3 Parameter Passing Mechanisms

These determine how arguments are transferred from caller to callee:

A. Call by Value

- Copies the actual value.
- Changes made in the function don’t affect the original variable.
- Default in C.

B. Call by Reference

- Pass the address of the variable.
- Changes affect the original variable.
- C uses pointers for this effect.