Memory Segments in C Programs (Deep Dive)

In C, executable programs are divided into **memory segments** (sections) by the compiler and OS. These segments help organize data and code efficiently. Here's a breakdown:

1. Text Segment (`.text`)

- **Stores**: Executable machine code (compiled program instructions).
- **Permissions**: Read-only + executable (for CPU to fetch instructions).
- **Example**:

```
int main() { return 0; }  // Code goes here.
```

2. Read-Only Data Segment (`.rodata`)

- **Stores**: Constants like **string literals** (`"hello"`), `const` variables, and global `static` constants.
- **Permissions**: Read-only (writing crashes the program).
- **Example**:

```
const int x = 10;        // Stored in `.rodata`.
```

-
```
char *str = "hello";     // "hello" is in `.rodata`.
```

3. Data Segment (`.data`)

- **Stores**: **Initialized** global/static variables (with explicit values).
- **Permissions**: Read + write.
- **Example**:

```
int global = 42;         // Stored in `.data`.
```

-
```
static int y = 100;      // Also in `.data`.
```

4. BSS Segment (`.bss`)

- **Stores**: **Uninitialized** global/static variables (set to zero by default).
- **Permissions**: Read + write.
- **Example**:

```
int uninit_global;      // Stored in `.bss` (auto-initialized to 0).
```

- `static int z;` `// Also in .bss.`

5. Stack Segment

- **Stores**: Local variables, function arguments, return addresses.
- **Permissions**: Read + write.
- **Grows downward** (from high to low memory addresses).
- **Example**:

```
void foo() {
```

- `    int local = 5;` `// Stored on the stack.`
- `}`

6. Heap Segment

- **Stores**: Dynamically allocated memory (`malloc`, `calloc`, `free`).
- **Permissions**: Read + write.
- **Grows upward** (from low to high memory addresses).
- **Example**:

```
int *arr = malloc(10 * sizeof(int));  // Heap memory.
```

How String Literals Fit In

```
char *s1 = "hello";   // "hello" → `.rodata` (READ-ONLY).
char s2[] = "world";  // "world" copied → **stack** (modifiable).
```

- `s1` points to `.rodata` → **crash if modified**.

- `s2` is a stack array → **safe to modify**.

Summary Table

Segment	Stores	Permissions	Example
`.text`	Executable code	Read + Exec	`main()`
`.rodata`	String literals, `const` vars	Read-only	`"hello"`
`.data`	Initialized globals/ statics	Read + Write	`int x = 10;`
`.bss`	Zero-initialized globals/statics	Read + Write	`int y;`
Stack	Local variables, function calls	Read + Write	`int local;`
Heap	Dynamic memory (`malloc`)	Read + Write	`int *arr = malloc(10);`

Why Does the Original Code Crash?

```
char *str1 = "good morning\n";  // `str1` points to `.rodata` (READ-ONLY).
strcpy(str1, "hello");          // Tries to WRITE → **Segmentation Fault**.
```

Fix: Use a **modifiable** buffer (stack or heap):

```
char str1[] = "good morning\n";  // Stack (modifiable).
strcpy(str1, "hello");           // Works!
```