PROGRAMMING FUNDAMENTALS

WEEK 13: DYNAMIC MEMORY IN C

LECTURE GOALS

- Understand why dynamic memory allocation is needed.
- Learn the 4 key functions: malloc, calloc, realloc, free.
- Practice allocating and freeing memory safely.
- Recognize and avoid memory leaks and dangling pointers.
- See a practical example (e.g., dynamic array).

WHY DYNAMIC MEMORY?

Core Problem: Fixed-size arrays are too small or too wasteful.

Fixed arrays (on stack):

```
int arr[1000];  // Wastes memory if you only need 5
elements
int small[5];   // Crashes if you need 1000 elements
```

WHY DYNAMIC MEMORY?

Dynamic allocation (on heap):

- Ask for exactly the memory you need
- Size can be decided at runtime
- More flexible

Visual:

- Stack: Fixed arrays (fast, limited size)
- Heap: Dynamic arrays (slower, flexible size)
- 💡 Think: Stack is like a **fixed desk**, Heap is like a **library** where you can borrow space as needed.

MALLOC() – ALLOCATE MEMORY

Purpose: Request a block of memory of given size (in bytes) from the heap.

Syntax:

```
void *malloc(size_t size);
```

Returns: Pointer to allocated memory, or NULL if failed.

MALLOC() – ALLOCATE MEMORY

Example 1:

MALLOC() – ALLOCATE MEMORY

Key Points:

- sizeof(int) = 4 bytes (on most systems)
- Cast void* to int* using (int*)
- Always check for NULL (allocation failed)

FREE() – RELEASE MEMORY

Purpose: Return allocated memory back to the heap.

Syntax:

```
void free(void *ptr);
```

Why must we call free()?

- Prevent memory leaks (program keeps asking for memory without giving it back)
- Avoid system slowdown or crash

FREE() – RELEASE MEMORY

- **Example:**
- `int *p = (int*)malloc(10 * sizeof(int));`
- `// ... use p ...`
- `free(p); // Give memory back to system`
- `p = NULL; // Avoid dangling pointer`
- ⚠️ Never use p after free(p) — it's a **dangling pointer**!

CALLOC() VS MALLOC()

- Purpose: Like malloc(), but initializes memory to zero.

Syntax:

```
void *calloc(size_t num, size_t size);
```

Parameters:

- num = number of elements
- size = size of each element

CALLOC() VS MALLOC()

- Example:

```
int *arr = (int*)calloc(5, sizeof(int));  // Allocates 5 ints, all set to 0
// No need to check arr == NULL, but good practice
for (int i = 0; i < 5; i++) {
    printf("%d ", arr[i]);  // Output: 0 0 0 0 0
}
free(arr);
```

CALLOC() VS MALLOC()

- Key Difference:

FUNCTION	INITIALIZES TO ZERO?	SYNTAX
malloc	No	Malloc(size)
calloc	Yes	Calloc(num, size)

💡 Use calloc when you want a clean slate (e.g., counters, flags).

REALLOC() – RESIZE MEMORY

- Purpose: Change the size of an already allocated block.

Syntax:

```
void *realloc(void *ptr, size_t new_size);
```

What it does:

- Can expand or shrink memory block
- May move the block to a new location (returns new address)

REALLOC() – RESIZE MEMORY

- Example 2

REALLOC() – RESIZE MEMORY

Important:

- Always reassign the return value: ptr = realloc(ptr, new_size);
- realloc(NULL, size) == malloc(size)
- realloc(ptr, 0) == free(ptr) (but don’t rely on this)

PRACTICAL EXAMPLE – DYNAMIC ARRAY

Example 3

Goal: Let user decide array size at runtime.

Live Demo: Run with n = 3, then n = 1000 — show flexibility!

MEMORY LEAKS & SAFETY RULES

🔴 Common Mistakes:

- Forgetting free() → Memory leak

```
int *p = malloc(100);  // 100 bytes allocated
// ... use p ...
// forgot free(p) → 100 bytes lost forever!
```

MEMORY LEAKS & SAFETY RULES

🔴 Common Mistakes:

- Using memory after free() → Dangling pointer

```
int *p = malloc(10);
free(p);
*p = 5;  // UNDEFINED BEHAVIOR!
```

- Not checking malloc return → crash if allocation fails

BEST PRACTICES:

1. Always check malloc/calloc/realloc for NULL.
2. Always call free() for every malloc/calloc/realloc.
3. Set pointer to NULL after free() to avoid dangling use.

ASSIGNMENT

1. Write a program that uses malloc to create an array of n floats (user inputs n), fills it with user values, prints them, and frees the memory.
2. Write a program that uses calloc to create an array of 10 integers, prints them (should all be 0), and frees the memory.
3. Create an array of 3 integers with malloc, then use realloc to expand it to 5 integers, assign values to all 5, print them, and free.
4. (Challenge) Write a program that asks the user for n, creates a dynamic array of n integers, fills it with squares of indices (0, 1, 4, 9, ...), prints them, and frees memory.

Instructions:

Submit as .c files