CS1 – Dynamic Memory Allocation (aka DMA)

Michael McAlpin

UCF

Outline

- Memory Management Overview
- Dynamic Memory Allocation
- Usage
- Other ideas for buffers
- And the problem is…
- There's still a problem…
- Things to ponder…

Memory Management Overview

- Memory management
  - Static
    - Allocated in main memory
    - Along with the code
    - Persists for the life of the execution
  - Automatic
    - Allocated on the stack as required
      - for functions as they come & go
  - Dynamic
    - Allocated from “free store” (aka the *heap*)

Dynamic Memory Allocation

- Dynamic
  - Allocated from "free store" (aka the *heap*)
  - Heap size varies by OS, IDE, compile time options, etc.
- TA DA!
  - malloc() returns a pointer to a chunk of memory that is the requested size data type
    - Allocates (& types) the data from the heap
  - free() returns the chunk of *malloc'd* memory to the heap

Usage[1]

- Malloc
  - `type * varName = malloc(varSize);`
  - Notes:
    - Returns a memory address of the buffer of the requested size
    - On error returns a NULL
    - *DOES NOT INITIAL MEMORY CONTENTS.*
- Free
  - `free(varName);`

Usage^2

- Let's get DEFENSIVE!!!

```
int * array = malloc(10 * sizeof(int));
if (NULL == array) {
  fprintf(stderr, "malloc failed\n");
  return(-1);
}
```

- Why?

Usage[3]

- Arrays of integers and floats are the same length as the number of elements in the array.
  - That is an array of 10 integers will be 10 integers long.
- Arrays of characters are of variable size (just like integers & floats) but are TERMINATED by a NULL character.
  - SO, a buffer created to hold 26 characters will be 26 + 1 characters long.
    - The last character contains a NULL.

Usage[4]

- Creation and destruction is a cycle that must be managed…
  - For example:
    - **`int *p = malloc(500 * sizeof(int));`**
    - must have a corresponding **`free(p);`**
- Why?

Other ideas for buffers

- An array of ten integers?
  - `int array[10];`
    - Static or dynamic? Why?
  - `int * arrayInts = malloc(10 * sizeof(int));`
    - Static or dynamic? Why?
- Accessing the contents of a buffer
  - Logically equivalent? Or not?

```
* arrayInts = 42;
arrayInts[0] = 42;
```

And the problem is…

- Not checking for allocation failures
  - No guarantees of success
  - May return a NULL pointer
    - Can lead to UNDEFINED error(s)
- Memory leaks
  - Failure to FREE leads to memory consumption
    - Lead to allocation failures
- Logical errors

There's still a problem…

- Logical errors
  - Recap
    - Malloc
    - Use buffer
    - Free it
- Error cases
  - Dangling pointer
    - memory usage after FREE
    - calling FREE twice on same buffer
    - calling FREE before MALLOC

Things to ponder…

- Given:

```
char *name = NULL; //buffer
name = malloc(...);//←
```

  - What are the parameter(s) for malloc assuming a maximum name length of 40 characters?

```
// Check your understanding: Why do we
// use name instead of &name in this
// scanf() statement?
scanf("%s", name);
printf("You entered: %s\n\n", name);
```

Recap

- Memory Management Overview
- Dynamic Memory Allocation
- Usage
- Other ideas for buffers
- And the problem is…
- There's still a problem…
- Things to ponder…

Coming soon…

- function arg lists
- Structs
- File I/O
- Tools
- Debugging

- Really…