CS1 – Dynamic Memory Allocation (& pointers) Part II

Michael McAlpin

UCF

Outline

- Usage Review
- Array review
- Pointer techniques for arrays
- Coming soon…

Usage$^{\text{Review}}$

- Let’s get DEFENSIVE!!!

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

Array review

- Accessing the elements of an array

```
int i, *p = NULL, array[5];
// print the contents of array
for (i = 0; i < 5; i++)
     printf("array[%d]: %d\n", i, array[i]);
```

Pointer techniques for arrays[1]

- Remember:

```
//Using array as a pointer, the result
//should be the same as &(array[0]):
   printf("array: %p\n", array);
   printf("&(array[0]): %p\n\n", &(array[0]));
```

- IF that is true then…

Pointer techniques for arrays^2

- Remember printing the array's contents:

```
for (i = 0; i < 5; i++)
      printf("array[%d]: %d\n", i, array[i]);
```

- AND remember the first element of the array can be accessed as follows:

```
printf("array: %p\n", array);
printf("&(array[0]): %p\n\n", &(array[0]));
```

Pointer techniques for arrays[3]

- And the second element of the array can be accessed as follows:

```
printf("array: %p\n", array + 1);
printf("&(array[1]): %p\n\n", &(array[1]));
```

- So, it seems that C has the parsing power to recognize that when dealing with a pointer to ***int***s in an array that the +1 indicates to move FORWARD one ***int***. (Typically 32 bits or 4 bytes.)*
- This is ***very*** powerful.

Recap

- Usage Review
- Array review
- Pointer techniques for arrays
- Coming soon…

Coming soon…

- Homework 1
- function arg lists
- Structs
- File I/O
- Tools
- Debugging

- Really…