16.216: ECE Application Programming

Summer 2013

Lecture 10: Key Questions

August 9, 2013

1. Describe what a structure is in C, and how structures can be useful.

2. Explain how we can essentially declare our own types using structures.

3. Show how variables of a given structure type can be declared and initialized.

4. Show how elements within a structure can be accessed.

5. **Example:** What does the following program print?
```c
#include <stdio.h>

typedef struct {
	double real;
	double imag;
} Complex;

int main() {
	Complex a = {1, 2};
	Complex b = {3.4, 5.6};
	Complex c, d, e;

	printf("A = %.2lf + %.2lfi\n", a.real, a.imag);
	printf("B = %.2lf + %.2lfi\n", b.real, b.imag);

	c = a;
	d.real = a.real + b.real;
	d.imag = a.imag + b.imag;
	e.real = a.real - b.real;
	e.imag = a.imag - b.imag;

	printf("C = %.2lf + %.2lfi\n", c.real, c.imag);
	printf("D = %.2lf + %.2lfi\n", d.real, d.imag);
	printf("E = %.2lf + %.2lfi\n", e.real, e.imag);

	return 0;
}
```

6. Explain how pointers are used to access structure variables.

7. Explain how structures are passed to and returned from functions.

8. **Example:** Write the following functions that use the `StudentInfo` structure

- Given a pointer to a single `StudentInfo` variable, print all of the student info to the screen using the following format:
  - `Michael J. Geiger`
  - `ID #12345678`
  - `GPA: 1.23`

- Given an array of `StudentInfo` variables, compute and return the average GPA of all students in the list

- Prompt the user to enter 3 lines of input (using the format below), read the appropriate values into `StudentInfo` elements, and return a value of type `StudentInfo`
  - Format (user input underlined)
  - `Enter name: Michael J. Geiger`
  - `Enter ID #: 12345678`
  - `Enter GPA: 1.23`

9. Explain how dynamic memory allocation is useful.

10. Explain the `malloc()` function.

11. Explain the use of type casting, and why it is necessary with the allocation functions.

12. Explain the `calloc()` function.

13. Explain the `realloc()` function.

14. Explain the use of the `free()` function and why it is necessary.

15. Explain how dynamic memory allocation can be used with arrays.

16. What does the following program print? What is the state of the array after each allocation function call?

```
void main() {
     int *arr;
     int n, i;

     n = 7;
     arr = (int *)calloc(n, sizeof(int));
     for (i = 0; i < n; i++)
          printf("%d ", arr[i]);
     printf("\n");

     n = 3;
     arr = (int *)realloc(arr, n * sizeof(int));
     for (i = 0; i < n; i++) {
          arr[i] = i * i;
          printf("%d ", arr[i]);
     }

     n = 6;
     arr = (int *)realloc(arr, n * sizeof(int));
     for (i = 0; i < n; i++) {
          arr[i] = 10 - i;
          printf("%d ", arr[i]);
     }
}
```

17. Explain the following common pitfalls of dynamic memory allocation, as well as their solutions:

a. Memory leaks

b. Dangling pointers