16.216: ECE Application Programming

Fall 2015

Lecture 31: Key Questions

November 20, 2015

1. **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:
  - o `Michael J. Geiger`
  - o `ID #12345678`
  - o `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`

2. Explain the `malloc()` function.

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

4. Explain the `calloc()` function.

5. (Explain the `realloc()` function.

6. **Example:** What does the following program print?

```
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]);
   }

   free(arr);
}
```

7. What are the common pitfalls of dynamic memory allocation?

8. Explain how to use dynamic memory allocation with strings.

9. Explain how to use dynamic memory allocation with two-dimensional arrays.

10. **Example:** Write each of the following functions:

a. **`char *readLine():`** Read a line of data from the standard input, store that data in a dynamically allocated string, and return the string (as a **`char *`**)
Hint: Read the data one character at a time and repeatedly reallocate space in string

b. **`int **make2DArray(int total, int nR):`** Given the total number of values and number of rows to be stored in a two-dimensional array, determine the appropriate number of columns, allocate the array, and return its starting address
Note: if **nR** does not divide evenly into **total**, round up. In other words, an array with 30 values and 4 rows should have 8 columns, even though 30 / 4 = 7.5