CS 2110 — Quiz 6
April 2018

Full name: Answer Key
GT username: _____
This quiz is worth a total of **100 points**.

Important Warnings:

1. Make no assumptions about the sizes of various datatypes. For each question, assume that ONLY <stdlib.h> and <stdio.h> have been included.
2. All code you write MUST compile with the following flags: *-std=c99 -pedantic -Wall -Werror -Wextra*
3. Having anything except the answer inside the boxes or above the fill-in lines reduces autograder performance and might cause incorrect results.
4. Make sure to write your name, username, and answers legibly. You will not receive credit for illegible answers.

Allocating Memory

Graders: Daniel, Josh, Maddie, Yuanhan

1. Initialize space for one *char*, one *short* and one *int* with **a single call** to malloc. Then assign the pointers *c*, *s* and *i* to point at the correct address for the char, the short, and the int, respectively. The space for each of the integers should not overlap, and you should not allocate more space than needed. **Do NOT check malloc for errors for this question. Do NOT call malloc more than once. Do NOT add any lines, write only on lines 1, 3, 5.** [20]

```
1  char *c = malloc(sizeof(char) + sizeof(short) + sizeof(int));
2
3  short *s = (short*) (c + 1)                    ;
4
5  int *i = (int*) (s + 1)                        ;
```

Copying Memory

Graders: Adam, Jim, Kexin, Preston, Yuanhan

2. Implement the function *copy_memory* in the box below. This function, when given a pointer and a number of bytes, should copy that many bytes starting at the pointer into a newly allocated space in memory, and return a pointer to this copy. It should return NULL if anything goes wrong. **You SHOULD check for malloc for errors for this question. You should not use memcpy, you should do the copying yourself.** [30]

```
1  void* copy_memory(void* src, size_t bytes) {
2      // Allocate room for the copy
3
4        char* copy = malloc(bytes);
5        if (copy == NULL) return NULL;
6
7
8
9
10     // Copy over the data
11
12       char* original = src;
13       for (size_t i = 0; i < bytes; i++) {
14          copy[i] = original[i];
15       }
16
17
18
19     // Return pointer to the copy
20
21       return copy;
22 }
```

Allocate 2D array *Graders: Austin, Madison, Sanjay, Vishnu, Yuuna*

3. Write a function that takes in the desired dimensions for a 2d array (the number of rows and the number of columns) and the size of each element in the 2d array (elem_size), allocates the 2d array, and returns a pointer to it. [30]

   The 2d array should be row-major (it should be possible to access the element at row y and column x using the expression *arr[y][x]* where *arr* is the produced array, cast to the correct type.

   Your function does not have to handle malloc errors. The rows of your array do not have to be contiguous in memory: you can allocate them all using a single malloc call or multiple, as you deem fit.

```
1  void **make_2d_array(unsigned int r, unsigned int c, size_t elem_size) {
2
3          void** rows = malloc(r * sizeof(void*));
4
5          for (unsigned int i = 0; i < r; i++) {
6              rows[i] = malloc(c * elem_size);
7          }
8
9          return rows;
10
11
12
13
14
15 }
```

Pointers, Arrays, Structs and Compiler Errors *Graders: Ben, Cem*

4. Look at the code snippet below, and for each of lines 15 to 21, mark if the statements are valid (i.e. will not cause compiler errors.) Mark Y if valid, N if not, on the underscores in the comments. [20]

```
1  char *migos = "Quavo";
2  char gucci[] = "Radric Davis";
3
4  typedef struct _human {int gucci_concerts_attended; int atl_years;} person;
5  typedef struct {int gucci_concerts_attended; int atl_years;} cs2110TA;
6
7  person Austin = {5, 2};
8  cs2110TA Yuanhan;
9  struct _human Sanjay;
10
11 void* voidPtr = malloc(1337);
12 int* intPtr;
13 char* charPtr;
14
15 migos[0] = 'M';      // Is it valid?: __N____
16 gucci[3] = 'D';      // Is it valid?: __Y____
17 Yuanhan = Austin;    // Is it valid?: __N____
18 Sanjay = Austin;     // Is it valid?: __Y____
19 intPtr = voidPtr;    // Is it valid?: __Y____
20 voidPtr = intPtr;    // Is it valid?: __Y____
21 charPtr = intPtr;    // Is it valid?: __N____
```