CS31 Written Homework 5: Arrays, Strings and Structs

Spring 2020 Due: Monday, April 6th, 11.59pm EDT

Your SwatID:

This is an individual HW and not paired.

Consider the following declarations and assignments. Use `man` pages to understand the string functions beginning with `str`.

```
struct icecream{
    float quantity;
    char *flavor;
    int available;
 };

int main(){
    int a[1][2],  i=5;
    char *favorite, second_fav[7], f, *c;
    struct icecream *request, icecream_list[3];

    favorite = malloc(sizeof(char)*15); //assume malloc succeeds

    /* function that fills in list of icecream flavors
     * in the following order. flavor is malloced within each struct.
     * chocolate, cookie_dough, strawberry
     * quantity and available are set to 1.        */
    fill_icecream_list(icecream_list);
    request = &icecream_list[2];

    strcpy(favorite, "chocolate");
    strncpy(second_fav, "cookie_dough", 6);
    favorite[5] = '\0';

    for (i=0; i< 3 ; i++){
        f = *(icecream_list[i].(flavor+2));
    }

    if(strcmp(favorite, "chocolate")){
        i = 0;
    }

    c=&f;
    ---> SHOW RESULTS FOR QUESTION 1 and 2 WHEN PROGRAM REACHES HERE
    //free all allocated memory
}
```

Question 1: Based of the C code on the previous page, what are the TYPE, and `sizeof()` of each of the following expressions. If it is an address, describe what it is the address of. You can assume the size of a pointer is 8 bytes, irrespective of the data type it is pointing to.

	TYPE	sizeof()
1. `a`		
2. `a[0]`		
3. `c`		
4. `favorite`		
5. `second_fav`		
6. `request`		
7. `&request`		
8. `*request`		
9. `icecream_list`		
10. `icecream_list[0]`		

Question 2

Trace through the C code on page 1, and draw memory contents (heap and stack) at the execution point indicated in `main`, and show the output produced by a complete run of the program.

```
OUTPUT
------

Ignore Question!
```

```
Stack  and Heap Diagram
-----------------------
```

You can treat the memory allocated for a struct, like memory allocated to an int, char (primitive data type) and draw a single box to represent a single struct.

Question 3: Struct Alignment

For the struct shown below, show the struct alignment in main memory assuming the first memory address available to store the struct is `8880`. Write down an alternate struct declaration to allocate memory more efficiently. Show the struct layout in main memory on the following page for both the current struct layout and your memory efficient layout.

```
struct rect{
   short breadth;
   char color[3];
   int edges;
   short length;
   float area;
};

//Your memory efficient struct declaration
// there may be more than one efficient representation here):
-------------------------
```

Memory Address (in decimal)	struct alignment	memory efficient alignment
8880		
8881		
8882		
8883		
8884		
8885		
8886		
8887		
8888		
8889		
8890		
8891		
8892		
8893		
8894		
8895		
8896		
8897		
8898		
8899		
8900		
8901		
8902		
8903		
8904		
8905		
8906		
8907		
8908		
8909		
8910		
8911		
8912		
8913		
8914		

Question 4: Allocating 2D arrays dynamically

Consider the following 2D-array allocations and fill in the blanks:

```
#define N 2
#define M 3

int main(){
    int *humidity;    //dynamically allocated "2D" array using 1 malloc
    int **rainfall;   //dynamically allocated "2D" array using N+1 mallocs

    humidity = malloc(__________________) //N x M int values.
    if(!humidity){
        perror("malloc");
        exit(1);
    }

   /* dynamically allocate 2D array of N+1 mallocs
    * first: malloc up 1 array of N integer pointers:
    *        i.e., one column of integer pointers.
    * next:  for each integer pointer, allocate an
    *        array of M ints (row of values)
    */

    // malloc up 1 column of N integer pointers (int *).
    rainfall = malloc(sizeof(int *) * ______);
    if(!rainfall){
        perror("malloc");
        exit(1);
    }

    /* for loop to malloc M ints for each of the
     * N integer pointers above.
     */
    for(________ ; ________________ ; ______){

         rainfall[i] = malloc(_____________);
         if(!rainfall[i]){
              perror("malloc");
              exit(1);
         }
     } //end of for loop

     //free space allocated to humidity
     free(_________);
```

```
    /*Write your loop here to free space allocated for each
     * row in rainfall. I.e., rainfall[i]  (row values)*/









     //finally free rainfall



}//end of main
```