Dynamic Allocation in the Foundation Exam

Max Koustikov

July 22, 2023

Contents

1 Reading Files

1.1 Grocery List

Date: September 5, 2020

Link: `https://www.cs.ucf.edu/registration/exm/fall2020/FE-Sept20.pdf`

Solution Link: `https://www.cs.ucf.edu/registration/exm/fall2020/FE-Sept20-Sol.pdf`

Suppose we are planning a party and we would like to create an array to store our list of supplies. Currently our list is stored in a text file with the name of each item to be purchased on a line by itself. Write a function called make-grocery-list that reads these items from a file and stores them in a two-dimensional character array. Your function should take 2 parameters: a pointer to the file and an integer indicating the number of grocery items in the file. It should return a pointer to the array of items. Be sure to allocate memory for the array dynamically and only allocate as much space as is needed. You may assume that all of the strings stored in the file representing grocery items are alphabetic strings of no more than 127 characters (so the buffer declared is adequate to initially read in the string).

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
char ** make_grocery_list (FILE *ifp, int numItems) {
     char buffer[128];
     char **list = NULL;
     int i;
```

1.2 Team/Player

Date: August 27, 2022

Link: `https://www.cs.ucf.edu/registration/exm/fall2022/FE-Aug22.pdf`

Solution Link: `https://www.cs.ucf.edu/registration/exm/fall2022/FE-Aug22-Sol.pdf`

Description: This problem relies on the following Player and the Team struct definitions:

```
typedef struct Player
{
    char pname[50];   //player's name
    char country[50]; //player's country
    int age;
} Player;

typedef struct Team
{
    char tname[50];  // team's name
    Player *players; // all players on the team
    int numPlayers;  // number of players on the team
} Team;
```

We are making a team of players from multiple countries. There is a text file containing the details of a team, where the first line of the file contains the team name, followed by a single space, followed by the number of players on the team, N. The next N lines contain the data for N players. Each player line contains three tokens, each separated by a space: the player name, country, and age (as an integer). Each team name, player name, and country will be a single-word string (no spaces) with a maximum length of 49. Here is a sample file:

```
NewKnights 5
Hannan USA 22
Mabel India 21
Samarina Bangladesh 21
Tamsen USA 21
Susan Mexico 22
```

Write a function that takes a file pointer and then returns a pointer to a dynamically allocated Team struct with all the information loaded into it. You can assume that the file is already opened in read mode and ready to read from the beginning of the file. Do not worry about closing the input file with fclose() when you finish reading it. Assume the function that opened the file and called createTeam() will close the file.

```
Team *createTeam(FILE *fp) {}
```

2 Miscellaneous

2.1 Employees

Date: January 15, 2022

Link: `https://www.cs.ucf.edu/registration/exm/spr2022/FE-Jan22.pdf`

Solution Link: `https://www.cs.ucf.edu/registration/exm/spr2022/FE-Jan22-Sol.pdf`

Consider the following structures and the main function shown below:

```
typedef struct StringType {
    char *string;
    int length;
} StringType;

#include <string.h>
#include <stdio.h>

int main() {
    //array of employees' names
    typedef struct Employee {
        StringType *ename;
        double salary;
    } Employee;

    char nameList[][50] = {"Adam", "Josh", "Kyle", "Ali", "Mohammed"};
    //array of salaries, where 15.80 is the salary of Adam, 13.50 is // the salary of Josh,
    double salaries[5] = {15.80, 13.5, 20.9, 12.99, 10.5};
    Employee *employees = createEmployees(nameList, salaries, 5);
    // Other code here...
    return 0;
}
```

Write a function `createEmployees()` that takes the list of employees' names, list of their salaries, and length of the list (`empCount`) as the parameters, and returns a pointer to a dynamically allocated array of `Employee` storing the relevant information for `empCount` employees. The function dynamically allocates memory for `empCount` number of employees and assigns the name and salaries for each of them from the input parameters. During this process, the names are stored in the dynamically allocated memory of `StringType`, and also make sure you assign the length of the name appropriately. Your code should use the exact amount of memory needed to store the corresponding names. You may assume no name is longer than 49 characters.

```
Employee* createEmployees(char names[][50], double *salaries, int empCount) {
    // implementation here
}
```

2.2 New Smart String

Date: August 8, 2020

Link: `https://www.cs.ucf.edu/registration/exm/fall2016/FE-Dec16.pdf`

Solution Link: `https://www.cs.ucf.edu/registration/exm/fall2016/FE-Dec16-Sol.pdf`

Consider the following struct, which contains a string and its length in one nice, neat package:

```
typedef struct smart_string {
  char *word;
  int length;
} smart_string;
```

Write a function that takes a string as its input, creates a new smart-string struct, and stores a new copy of that string in the word field of the struct and the length of that string in the length member of the struct. The function should then return a pointer to that new smart-string struct. Use dynamic memory management as necessary. The function signature is:

```
smart_string *create_smart_string(char *str) {
```

Now write a function that takes a smart-string pointer (which might be NULL) as its only argument, frees all dynamically allocated memory associated with that struct, and returns NULL when it's finished.

```
smart_string *erase_smart_string(smart_string *s) {
```

2.3 Juice Case

Date: August 8, 2020

Link: https://www.cs.ucf.edu/registration/exm/sum2020/FE-Aug20.pdf

Solution Link: https://www.cs.ucf.edu/registration/exm/sum2020/FE-Aug20-Sol.pdf

Suppose we have a structure to store information about cases of juice. The structure is shown below: the name of the juice in the case is statically allocated. The structure also contains the number of containers of juice in that case. Complete the function below so that will takes 2 parameters: the name of a juice and an integer. Your function should create a new case of juice by allocating space for it, copying in the contents specified by the formal parameters into the struct and returning a pointer to the new case. You may assume that the pointer new-name is pointing to a valid string storing the name of a juice for which memory has already been allocated and is 127 or fewer characters.

```
#include <string.h>
#include <stdlib.h>
struct juice_case {
     char name[128];
     int num_bottles;
};
struct juice_case* create_case(char *new_name, int new_number) {
```

2.4 Pascal's Triangle

Date: May 20, 2023

Link: `https://www.cs.ucf.edu/registration/exm/spr2023/FE-Jan23.pdf`

Solution Link: `https://www.cs.ucf.edu/registration/exm/spr2023/FE-Jan23-Sol.pdf`

Description: Using 0-based indexing, on row i of Pascal's Triangle, there are i+1 positive integer values. One way we can efficiently store the triangle is to allocate the correct amount of memory for each row. Here is a picture of the first five rows of the triangle (rows 0 through 4, inclusive.): If the name of the array is tri, then the rule to fill in the entries in the table are as follows: tri[i][0] = 1, for all positive ints i tri[i][i] = 1, for all positive ints i tri[i][j] = tri[i-1][j-1]+tri[i-1][j], for all ints j, 0 ¡ j ¡ i Write a function that takes in an integer, n, dynamically allocates an array of n arrays, where the ith array (0-based) is allocated to store exactly i+1 ints, fills the contents of the array with the corresponding values of Pascal's Triangle as designated above, and returns a pointer to the array of arrays. You may assume that 1 ¡ n ¡ 31.

```
int** getPascalsTriangle(int n) {}
```

3 Freeing Space

3.1 Holiday List

Date: August 28, 2021

Problem Link: `https://www.cs.ucf.edu/registration/exm/fall2021/FE-Aug21.pdf`

Solution Link: `https://www.cs.ucf.edu/registration/exm/fall2021/FE-Aug21-Sol.pdf`

Suppose we have an array to store all of the holiday presents we have purchased for this year. Now that the holidays are over and all the presents have been given out, we need to delete our list. Our array is a dynamically allocated array of structures that contains the name of each present and the price. The name of the present is a dynamically allocated string to support different lengths of strings. Write a function called `delete_present_list` that will take in the present array and free all the memory space that the array previously took up. Your function should take 2 parameters: the array called `present_list` and an integer, `num`, representing the number of presents in the list and return a null pointer representing the now deleted list. (Note: The array passed to the function may be pointing to NULL, so that case should be handled appropriately.)

```
struct present {
    char *present_name;
    float price;
};
struct present* delete_present_list(struct present* present_list, int num) {}
```

4 Finding Errors

4.1 5 Memory Management Issues

Date: May 21, 2022

Problem Link: https://www.cs.ucf.edu/registration/exm/sum2022/FE-May22.pdf

Solution Link: https://www.cs.ucf.edu/registration/exm/sum2022/FE-May22-Sol.pdf

The following function has 5 memory management issues, each one occurring on a different one of the 17 labeled lines of code. Please clearly list which five lines of code have the errors on the slots provided below. Please list exactly five unique line numbers in between 1 and 17, inclusive. An automatic grade of 0 will be given to anyone who lists MORE than 5 line numbers.

```
int n = 10; // line 1
int *p1, *p2, *p3, **p4; // line 2

char str1[100] = "test string"; // line 3
char *str2; // line 4
strcpy(str2, str1); // line 5

p1 = (int *)malloc(n * sizeof(int)); // line 6
p2 = (int *)malloc(n * sizeof(int)); // line 7
for (int i = 0; i < n; i++) // line 8
{
    p1[i] = rand() % 100; // line 8
}

p2 = p1; // line 9
p3 = (int *)malloc(sizeof(int)); // line 10
*p3 = 50; // line 11

p4 = (int **)malloc(n * sizeof(int *)); // line 12
for (int i = 0; i < n; i++) // line 13
{
    p4[i] = (int *)malloc(sizeof(int)); // line 14
    *(p4[i]) = -5; // line 15
}

free(p1); // line 16
```

```
free(p2); // line 17
free(p4); // line 18
```

In this format, each line of code is displayed vertically, starting with the line number and followed by the corresponding code. Let me know if you need any further adjustments or if there's anything else I can help you with!

```
struct present {
    char *present_name;
    float price;
};
struct present* delete_present_list(struct present* present_list, int num) {}
```

4.2 Input Trimmed Down

Date: January 16, 2021

Problem Link: https://www.cs.ucf.edu/registration/exm/spr2021/FE-Jan21.pdf

Solution Link: https://www.cs.ucf.edu/registration/exm/spr2021/FE-Jan21-Sol.pdf

Suppose we have a function that is designed to take in a large string and trim it down to only the needed size. The function is called `trim_buffer`. It takes in 1 parameter: the buffer, which is a string with a max size of 1024 characters. It returns a string that is only the size needed to represent the valid characters in the buffer. The function is implemented below.

```
#define BUFFERSIZE 1024
// Pre-condition: buffer has a '\0' character at or before index
//                BUFFERSIZE-1.
// Post-condition: returns a pointer to a dynamically allocated
//                 string that is a copy of the contents of buffer,
//                 dynamically resized to the appropriate size.
char * trim_buffer(char * buffer) {
     char *string;
     int length;
     while (length < BUFFERSIZE && buffer[length] != '\0')
               length++;
     string = malloc(sizeof(char) * (length));
     length = 0;
     while ((string[length] = buffer[length]) != '\0')
          length++;
     return string;
}
```

4.3 Linked List Errors

Date: May 20, 2023

Link: https://www.cs.ucf.edu/registration/exm/sum2023/FE-May23.pdf

Solution Link: https://www.cs.ucf.edu/registration/exm/sum2023/FE-May23-Sol.pdf

The struct below is used to define a node in a linked list. Below it is a function, freeList, that is given a pointer to the front/head of a linked list. The function is supposed to free all the dynamically allocated memory associated with the linked list that the given pointer is pointing to. Unfortunately, the function does not work. Explain why the function doesn't work and propose a fix to the function, in words only, conceptually explaining the order in which the key steps have to be executed.

```
typedef struct node {
    int data;
    struct node* next;
} node;
void freeList(node* front) {
    if (front != NULL)
        free(front);
    if (front->next != NULL)
  freeList(front->next);
}
```

Problem(s) with current code:

Proposed fixes (conceptually, in words):