Week 3 Recitation Exercise

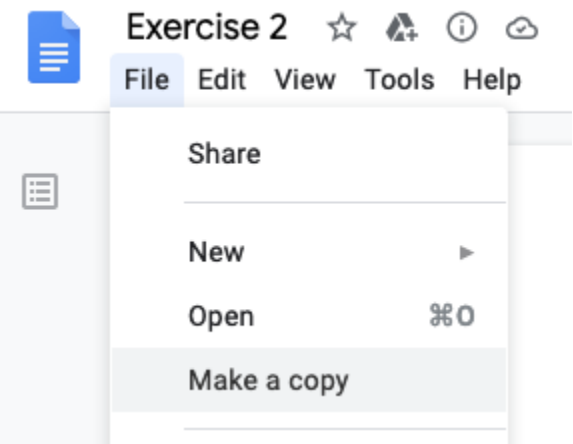
CS 261 – 10 points

In this exercise, you will explore how pointers and memory allocation work in C. This is a pencil-and-paper exercise, not a programming exercise. In it, you will be presented with two different C programs. Working in a small group, you will identify all of the memory that is allocated within each program, and you'll describe certain characteristics of each piece of allocated memory (e.g. whether that piece of memory is allocated on the call stack or on the heap).

Follow the instructions below, and when you're finished, make sure to submit your completed exercise on Canvas. Please submit one copy per group (as long as all group members' names are listed below, it's fine if just one person submits). Remember that this exercise will be graded based on effort, not correctness, so don't worry if you don't get all the right answers. Do make sure you *try* to get the right answers though.

Step #1: Organize your group and create your submission document

Your TA will assign you to a small group of students. Among your group, select one group member to record your group's answers for this exercise. This group member should make a personal copy of this document so they are able to edit it:



Step #2: Add your names

Next, your group's recorder should add the names and OSU email addresses of all of your group's members in the table below:

Group member name	OSU email address
HyunTaek, Oh	ohhyun@oregonstate.edu

Step #3: Examine two C programs

There are two different C programs contained in the following GitHub Gist (`prog1.c` and `prog2.c`):

https://gist.github.com/robwhess/5b102e5f7dbaad3f51b48cd7ef9ac7a5

Carefully examine each of these programs, and imagine in your mind how the program will behave as it's executing. In particular, pay special attention to the memory allocated by the program's `main()` function, both on the call stack and in the heap, and make note of how the value held in each piece of memory changes as the program runs.

Step #4: Describe all the memory allocated by the program above

In the tables below, describe each piece of memory allocated by the `main()` function of each program in the Gist linked above. List one piece of memory per row of the table (you may not need to use all of the table's rows), and for each piece of memory, list the following information:

- **Allocation type** – whether the piece of memory is statically allocated (i.e. allocated on the call stack) or dynamically allocated (i.e. allocated on the heap).
- **Size** – the number of bytes allocated to this piece of memory. Express each size using `sizeof()`, e.g. `sizeof(struct product)` or `16 * sizeof(int)`.

- **Data type** – the type of data stored in this piece of memory, e.g. `float` or `char*`. For arrays, indicate the type of data stored in the individual elements of the array.
- **Name** – *for statically allocated variables only*, indicate the variable name associated with this piece of memory.
- **Freed?** – *for dynamically allocated variables only*, indicate whether or not this piece of memory was correctly freed.
- **Value(s)** – the value stored in this piece of memory. If this value changes as the program executes, list all of the values stored in this piece of memory. Use the conventions below to describe some specific types of values:
    - Use the symbol "?" to indicate when a value is uninitialized.
    - If this piece of memory holds an array, list all of the array values within square brackets, e.g. `[ 1, 2, 3, 4, 5 ]`.
    - If this piece of memory holds a `struct` value, indicate the values for all fields of the `struct`.
    - If the value stored in this piece of memory is a pointer (i.e. a memory address), use the table's row labels to indicate *which* piece of memory is pointed to. For example, if this piece of memory contains a pointer to the memory you listed in the table row labeled "4", then indicate the value stored in this piece of memory as "&4".

prog1.c

	Allocation type (static / dynamic)	Size	Data Type	Name	Freed?	Value(s)
1	Static	4	Int	n		10
2	Static	4	Int	x		16
3	Static	4	Int	y		32
4	Static	4	Int	z		64
5	Static	4	Int	p1		16
6	Dynamic	(10 * sizeof(int))	Int*	a	X	
7	Static	4	Int	p2		
8	Static	4	Int	x		32
9	Static	4	Int	z		32
10	Static	4	Int	p1		32
11	Static	4	Int	y		64
12	Dynamic	(10 * sizeof(int))	Int*	a	O	
13						
14						
15						
16						
17						
18						
19						
20						

prog2.c

	Allocation type (static / dynamic)	Size	Data Type	Name	Freed?	Value(s)
1	Static	4	Int	I		
2	Static	4	Int	n		5
3	Dynamic	5 * sizeof(struct product*)	Struct product*	Products	X	
4	Dynamic	5 * Sizeof(struct product)	Struct product*	Product[0~4]	X	
5	Dynamic	(1+strlen(name))*sizeof(char))	Struct product*	Product[0]->name	X	“A product name”
6	Static	Sizeof(struct product)	Struct product*	Product[0]->price		32.99
7	Dynamic	(1+strlen(name))*sizeof(char))	Struct product*	Product[1]->name	X	“A product name”
8	Static	Sizeof(struct product)	Struct product*	Product[1]->price		32.99
9	Dynamic	(1+strlen(name))*sizeof(char))	Struct product*	Product[2]->name	X	“A product name”
10	Static	Sizeof(struct product)	Struct product*	Product[2]->price		32.99
11	Dynamic	(1+strlen(name))*sizeof(char	Struct product*	Product[3]->name	X	“A product name”

		))				
12	Static	Sizeof(struct product)	Struct product*	Product[3]->price		32.99
13	Dynamic	(1+strlen(name))*sizeof(char))	Struct product*	Product[4]->name	X	“A product name”
14	Static	Sizeof(struct product)	Struct product*	Product[4]->price		32.99
15	Dynamic	5 * Sizeof(struct product)	Struct product*	Product[0~5]	O	
16	Dynamic	5 * sizeof(struct product*)	Struct product*	Products	O	
17						
18						
19						
20						

Step #5 (extra credit): Identify memory leaks

Based on your analysis above, you should be able to identify leaked memory in one of the programs. For 1 point of extra credit, indicate below which program contains a memory leak, and describe both how the memory leak occurs and how to fix it.

Note that this extra credit portion of the assignment will be graded based on correctness, not just effort.

Extra credit solution:

[Explanation about Identifying memory leaks.

1. Which program contains a memory leak?
    - Prog2.c

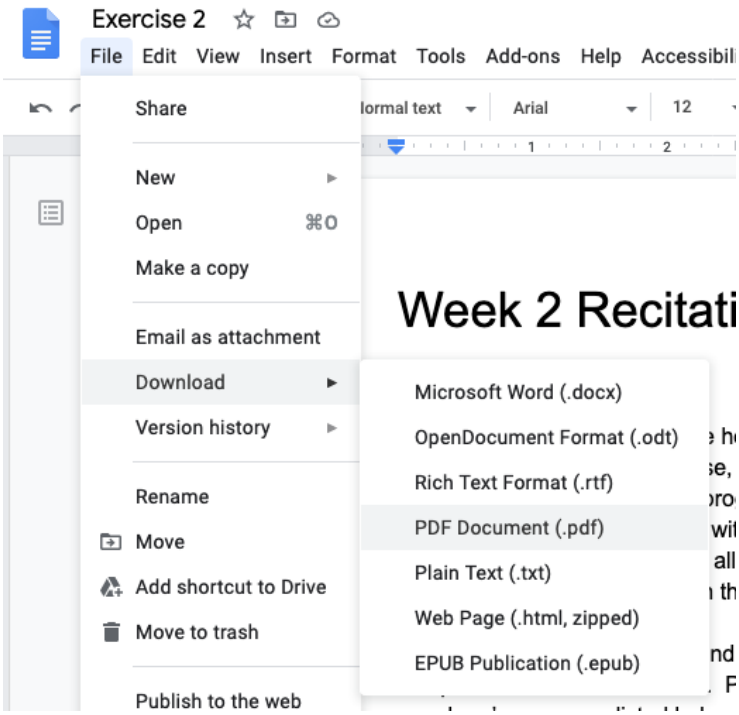
2. Describe both how the memory leak occurs and how to fix it?
    1) In the create_product function, there are two instructions using malloc() function such as p and p->name. p is not the cause of memory leak because when product[i] is free, p is also free (product[i]=create_product(~)). However, p->name is the cause of memory leaking and should be free before p is free. At first, I think when product[i] is free, p->name would also be free but it's not.

    2) As I mentioned above, In the create_product function, an instruction freeing the heap memory of "p->name" should be added [free(p->name)].

```
Struct product* create_product(char* name, float price){
    Struct product* p=malloc(sizeof(struct product));
    p->name = malloc((1+strlen(name)) * sizeof(char));
    strcpy(p->name, name);
    free(p->name);
    p->price = price;
    return p;
}
```

Submission and wrap-up

Once you've completed this exercise, download your completed worksheet as a PDF and submit that PDF on Canvas (refer to the image below to help find how to download a PDF from Google Docs). Again, please submit one copy of your completed worksheet for your entire group. It's OK if just one member submits on Canvas. All group members will receive the same grade as long as their names and email addresses are included above.


Exercise 2
File Edit View Insert Format Tools Add-ons Help Accessibil
Share
New
Open ⌘O
Make a copy
Email as attachment
Download
Version history
Rename
Move
Add shortcut to Drive
Move to trash
Publish to the web
Microsoft Word (.docx)
OpenDocument Format (.odt)
Rich Text Format (.rtf)
PDF Document (.pdf)
Plain Text (.txt)
Web Page (.html, zipped)
EPUB Publication (.epub)
Arial
12
Week 2 Recitati