Name: _____

Problem	Points	Score
1	50	
2	50	
Total	100	

Notes:

(1) The first step in this exam is to create a workspace in the following directory:

/data/courses/ece_1111/current/exams/ex_02/lastname_firstname

Following the usual rules. Store the code for problem no. 1 in a subdirectory p01, and the code for p02 in a subdirectory p02.

(2) Your code must use Makefiles, header files, driver program, implementation files, etc. as we have done all semester.

(3) You can, of course, reuse your code or any code I wrote.

(4) For this exam you are allowed to open a terminal window on your computer, you are allowed to web surf with Google, but you cannot use online chat or other interactive services, such as ChatGPT. Trust me – if you use that tool, I'll probably be able to figure it out ;)

(5) You must complete problem no. 1 before attempting problem no. 2. If your solution to problem no. 1 does not work, you will not get credit for problem no. 2.

(50 pts) Problem No. 1:

The goal of this problem is to write a program that stores command line arguments in an array of structures. The structure should contain the value of each command line argument and an integer that holds the arguments position on the list. Your main program will read the command line, convert the arguments to values in a structure, and store each structure in an array. You will then print the contents of the array.

This is what the interface to your program should look like:

```
ece-000_[1]: p01.exe apple orange banana
struct no. 0:
 value = p01.exe
index = 0
struct no. 1:
 value = apple
 index = 1
...
```

Your program should work for any number of command line arguments, and for any length argument. Memory allocation should be dynamic. You will be judged in part based on the memory efficiency of your code.

The driver program should:

(1) Declare a struct named **`Fruit`** in your header file.

(2) Declare an array of **`Fruit`** structs to hold all the command line arguments.

(3) In a function called **`myallocate`**, create a struct for each array element, copy the value of the command line argument into that struct, set the index, and return.

(4) In a function called **`myprint`**, loop over the array and print each element using the format above. Use a file pointer as an argument, as we did in the last quiz.

(5) In a function called **`mydeallocate`**, clean up memory before the program exits.

Since you have access to the number of command line arguments in argc, there is no real problem with (1). There should be no memory leaks in your code. Make sure you copy the contents of each command line argument rather than simply copy the pointer.

(50 pts) Problem No. 2:

Copy your *p01* code to your p02 subdirectory. Create a new function called **`myreallocate`** that accepts the array created in *p01* as an argument and returns a new struct that is $\boldsymbol{argc + BSIZE}$ elements long. **`BSIZE`** should be defined in your header file as a long integer with a value of 10.

The prototype for **`myreallocate`** should be this:

```
bool myreallocate(output_array, output_size, input_array, input_size, increment);
```

I will leave it up to you what the types of these arguments must be to get this to work properly. The argument **`increment`** should be passed as the value **`BSIZE`** from your header file.

Insert the call to **`myreallocate`** immediately before you call **`myprint`** in the driver program. The function **`myprint`** should iterate over the expanded array and print all the values, including the empty structs. The function **`myreallocate`** should copy the contents of the input array (values, not memory locations). The copy in the output array should be an independent copy and should not share any memory space with the input (in other words, allocate new space for the values in the input array). For example, if I delete the input array, the output array should not be impacted. You should be able to use your existing **`mydeallocate`** function with no modification.