LAB 6
Array of pointers. Command-line arguments (program arguments)

Due: July 8 (Monday) 11:59pm

1. Problem A

Motivation

It is usually a bit challenging to understand arrays of pointers, especially arrays of char pointers -- how to access the pointee strings, what type of pointers can be assigned to the array and how to access the pointee strings via the pointer, and what a pointer array decays to etc. This practice aims at helping you get started.

Specification

1. Download file `lab6A.c`. Look at the first 30 lines of code, which provides a recap of pointer basics. Observe/recall that,
   a. to print an scalar variable such as an integer via its pointer `p`, the argument to `printf` is the "pointee level" `*p`
   b. to print a char array (string), the argument to `printf` is at the "pointer level", i.e., `arr` or `p` where `p = arr = &arr[0]`
   c. to print a char in an array, the argument to `printf` is at the "pointee level", and there are several ways of doing that. The basic rule is that
      `arr[i] == *(arr+i) == *(p+i)` where `p = arr = &arr[0]`

2. Next, complete the "array of pointer to int" section (line 30 - line 55) by following the comments.
   - Hint: note that after initialization, `arrP[0]` contains the pointer (i.e., address) of `i`, and `arrP[1]` contains the pointer to `j` and so on. Now according to observation `1.a` above, to print the value of a scalar variable such as `j`, we pass a "piontee level" argument to `printf`. Hence `*arrP[1]`. Now according to `1.c`, `arrP[1] == *(arrP+1)`, so the argument can also be in the form of `**(arrP+1)`.
   - Hint2: if we want to declare a pointer `pp0` to point to the first element of `arrP`, i.e., `pp0 = &arrP[0]`, what type of `pp0` should it be? Since `arrP[0]` is a pointer to int, `pp0` will contain address `&` of a pointer, so `pp0` should be a <u>pointer to pointer</u>. Also since array name `arrP == &arrP[0]`, we can write `pp0 = arrP` directly. Now according to `1.c` above, `arrP[i] == *(arrP+i) == *(pp0+i)`. Hence to print value of `j`, we can also pass `**(pp0+1)` as argument to `printf`.

3. Next, complete the "array of char pointers" section (line 60 - line 90) by following the comments.
   - Hint: note that for pointer array `planets`, after initialization, `planets[0]` contains a pointer to string `"Mercury"` (more formally, `planets[0]` stores the starting address of "Mercury" which is the address of its first element `'M'`). Likewise `planets[1]` contains the pointer to string "Venus" and so on.

Now according to observation `1.b` above, to print a char array (string) we pass as argument to `printf` the array name or a pointer to the string ("pointer level"). Thus to print "Mercury" we pass as argument to `printf` a pointer to "Mercury", which is `planets[0]` or `*(planets+0)`, and to print "Venus", we pass as argument to `printf` a pointer to "Venus", which is `planets[1]` or `*(planets+1)`, and so on.

- Hint2 above can help determine the type of `pp` and how to print the strings via `pp`.

4. [Optional] Now you may wonder what `*planets[1]` or `**(pp+1)` is and when we should use them? This topic is in the course syllabus, but is not required this semester due to lecture progress. For interested students, do the following optional exercise
   - Uncomment the last three lines of code, and run the program again.
   - Observe how the characters in the pointee strings are accessed using pointer notation. Convince yourself that although they look quite daunting, they make sense. Notice that the parentheses are necessary to enforce the order of evaluation.

Sample Inputs/Outputs The final outputs of the program should be

```
red 329 % a.out
10

hello hello hello
5 5 5
llo llo llo
3 3 3

h h h
e e e
o o o

1 1
3 3
5 5
1
3
5

Mercury  Mercury 7 7
Venus  Venus
Jupiter  Jupiter
Saturn  Saturn
Neptune  Neptune

Mercury
Venus
Jupiter
Saturn
Neptune

M  M  M
i  i  i
u  u  u
red 330 %
```

Submit your program using **`submit 2031 lab6 lab6A.c`**

2. Problem B

Subject

Similarities and differences between 2D char array and array of char pointers, both of which can be used to store rows of input strings.

Specification

Write an ANSI-C program that uses 2D array to read and store user input strings line by line, until a line of `xxx` is entered (similar to lab4). The program then reorders the rows of inputs.

Implementation

Assume that there are at least 6 lines of inputs (excluding the terminator line `xxx`) and there are no more than 30 lines of inputs. Also assume that each line contains no more than 50 characters. Note: each line of input may contain spaces.

- Use a table-like **2D array** to store the user inputs. That is, similar to lab4, define `char inputs[30][50]`.
- Use `fgets(inputs[current_row], 50, stdin)` to read in a line into the table row directly. Note that a trailing `\n` is also read in.
- When all the inputs have been read in (indicated by input line `xxx`), exchange row 0 and row 1 in `main()`, and then send the array to a function `exchange()` to exchange some other rows.
- Define a function `void exchange(char[][50])` which takes as argument an 2D array, and swaps the record in 3rd row of the array with that in 4th row, and swaps the record in the 5th row with that in the 6th row. Assume that the argument 2D array has at least 6 rows.
- Define a function `void printArray(char[][50], int n)` which takes as argument a 2D array, and then prints the first `n` rows of the array on stdout.
  Use this function in `main` to display all the stored rows of the array, both before and after the swapping.

Sample Inputs/Outputs:

```
indigo 329 % a.out
sizeof inputs: 1500

Enter string: this is input 0, giraffes
Enter string: this is input 1, zebras
Enter string: this is input 2, monkeys
Enter string: this is input 3, kangaroos
Enter string: this is input 4, do you like them?
Enter string: this is input 5, yes
Enter string: this is input 6, thank you
Enter string: this is input 7, bye
Enter string: xxx

[0]: this is input 0, giraffes
[1]: this is input 1, zebras
[2]: this is input 2, monkeys
[3]: this is input 3, kangaroos
[4]: this is input 4, do you like them?
[5]: this is input 5, yes
[6]: this is input 6, thank you
```

```
[7]: this is input 7, bye

== after swapping ==
[0]: this is input 1, zebras
[1]: this is input 0, giraffes
[2]: this is input 3, kangaroos
[3]: this is input 2, monkeys
[4]: this is input 5, yes
[5]: this is input 4, do you like them?
[6]: this is input 6, thank you
[7]: this is input 7, bye
```

Name your program `lab6B.c` and submit your program using

`submit 2031 lab6 lab6B.c`

After you submit, as an additional practice, change the formal argument in one of the function definitions (and the corresponding declaration) from `char[][50]` to `char [][]`, for example, `void exchange(char[][])`, and compile. What do you get?

3 Problem C

Subject

Similarities and differences between 2D char array and array of char pointers.
Store strings using Array of (char) Pointers. Pass array of pointers to functions. Swap records of pointer arrays.

Specification

Write an ANSI-C program that reorders the pointees of a pointer array.

Implementation

- Download the program `lab6C.c` and start from there. Observe how an array of char pointers is declared and initialized.
- In `main`, first exchange pointees of the first (element [0]) and the 2nd (element [1]) pointers of the pointer array.
- Then, send the pointer array to function `exchange()` to exchange some other pointees.
- Define a function `void exchange(char * records[])` which takes as argument an array of char pointers, and swaps the pointee of the 3rd element pointer with the fourth element pointer, and swap the pointee of the 5th element pointer with that of the 6th element pointer. Assume the argument array contains at least 6 rows.
  - You should accomplish the swapping without copying/moving the original string data. Specifically, you should not use library functions or loops to do the swapping. This is one of the advantages of using pointer arrays against 2-D arrays.

- Define a function `void printArray(char * records[], int n)` which takes as argument an array of char pointers, and prints the first `n` pointees of `records` on stdout, one line for each pointee of the array.
  Use this function in `main` to display all the pointees pointed by the pointer array, both before and after the swapping. Note that since records is a 'general array' which contains no terminator token, we need to pass `n` as additional argument for the length information.

- Define a function `void printArrayP(char ** records, int n)` which takes as argument an array of char pointers, and prints the first `n` pointees of `records` on stdout, one line for each pointee of the array. **Use pointer notation only, don't use array index notation in this function.**
  Note that the argument is declared as a pointer to pointer `char **`, which is what an array of char pointer `char *` [] is "decayed" to when it is passed to a function (why? Recall that array name contains the address of its first element. Thus when array is passed function, it is decayed to the address of its first element. Since this is a pointer array, the first element is a pointer, so address of a pointer, i.e., a pointer to pointer, is received by the function.)
  Use this function in `main` to display all the pointees pointed by the pointer array, after the swapping.

Sample Inputs/Outputs:

```
red 329 % a.out
sizeof char*: 8, sizeof inputs: 64
```

You might get `4 32` if run on your system.

```
[0] -*-> this is input 0, giraffes
[1] -*-> this is input 1, zebras
[2] -*-> this is input 2, monkeys
[3] -*-> this is input 3, kangaroos
[4] -*-> this is input 4, do you like them?
[5] -*-> this is input 5, yes
[6] -*-> this is input 6, thank you
[7] -*-> this is input 7, bye

== after swapping ==
[0] -*-> this is input 1, zebras
[1] -*-> this is input 0, giraffes
[2] -*-> this is input 3, kangaroos
[3] -*-> this is input 2, monkeys
[4] -*-> this is input 5, yes
[5] -*-> this is input 4, do you like them?
[6] -*-> this is input 6, thank you
[7] -*-> this is input 7, bye

[0] -*-> this is input 1, zebras
[1] -*-> this is input 0, giraffes
[2] -*-> this is input 3, kangaroos
[3] -*-> this is input 2, monkeys
[4] -*-> this is input 5, yes
[5] -*-> this is input 4, do you like them?
[6] -*-> this is input 6, thank you
[7] -*-> this is input 7, bye
red 330 %
```

Submit your program using **`submit 2031 lab6 lab6C.c`**

4. Problem D

Subject:

Command line arguments (program parameters) and pass pointer arrays to functions.

Background:

Command line argument is a parameter supplied to the program when it is invoked. Command-line arguments are given after the name of the executable program (e.g. `a.out`) in command-line shell of Operating Systems. In addition to `scanf, fgets`, command line argument provides another way for the users to interact with the program.

Specification

Write a (short) ANSI-C program that reads command line inputs, which are integer literals, and then outputs the total number of integers, followed by the sum of the input integers.

Implementation

- Define a function `int getSum(char *[], int n)`, which takes as argument an array of char pointers and returns the sum of the first `n` value of the pointees of the array elements.
- Define a function `int getSumP(char **, int n)`, which takes as argument an array of char pointers and returns the sum of the first `n` values of the pointees of the array elements.
  Use pointer notation only, don't use array index notation.
- Display all the command line arguments, and then display the sum twice -- first the result from `getSum()` and then the result from `getSumP()`, as shown in the sample outputs below.

- Assume that each pointee of the pointer elements, which is a char array, is a valid integer literal, such as "42".
- Assume also that there are at least two input integers
- Do not use global variables.
- Name your program `lab6Dargv.c`

Sample Inputs/Outputs:

```
red 377 % gcc lab6Dargv.c
red 378 % a.out 1 2
There are 2 arguments (excluding "a.out")
1 + 2
= 3
= 3

red 379 % a.out 1 2 3 4 23 11 32 345 11 3 4
There are 11 arguments (excluding "a.out")
2 + 3 + 4 + 23 + 11 + 32 + 345 + 11 + 3 + 4
= 439
= 439

red 380 % gcc lab6Dargv.c -o xyz.out
```

```
red 381 % xyz.out 2 5 6 19 40
There are 5 arguments (excluding "xyz.out")
2 + 5 + 6 + 19 + 40
= 72
= 72
red 382 %
```

Name your program `lab6Dargv.c` and submit using

`submit 2031 lab6 lab6Dargv.c`

5. Problem E

Subject

Array of pointers. Dynamic memory allocation. Heap.
We will cover dynamic memory allocation in the next class. But doing this exercise gets you better prepared for that.

Implementations

Download and read `setArr.c`. This simple program declares an array of int pointers, and then tries to initialize the first 3 pointers with pointee values 100, 200 and 300, performed in `main`. Then in function `setArr()`, initializes the fourth and fifth pointers with pointee values 400 and 500.

Now compile and run the program. You will get a *segmentation fault (core dumped)*. Think about why this happens.

Now comment out line 12 and uncomment line 13, and compile and run again.
Observe that,

- Pointers at `arr[0], arr[1], arr[2]` are initialized correctly.
- Pointers at `arr[3]` and `arr[4]`, which are initialized in function `setArr()`, are not set correctly.

Run the program again, and you will observe the similar results, with different (garbage) values for `arr[3]` and `arr[4]`.

Think about why this happens. You are also encouraged to fix the problem so that the two pointers are initialized correctly in `setArr()`. (You should not introduce new global variables and should not move the code of `setArr()` into `main()`.)
Don't spend too much time on it if you can't make it. We will cover this in next class.

No submission for this exercise.

In summary, for this lab you should submit the following files

`lab6A.c lab6B.c lab6C.c lab6Dargv.c`

You may want to issue **`submit -l 2031 lab6`** to view the list of files that you have submitted.

Common Notes

All submitted files should contain the following header:

```
/**************************************
* EECS2031 – Lab 6 *
* Author: Last name, first name *
* Email: Your email address *
* eecs_num: Your eecs login username *
* Yorku #:  Your York student number
**************************************/
```