IC221 Lab 3: C Programming with Formatted I/O and Arrays

See solutions at https://github.com/umbertofontana/systems-programming

Learning Objectives

- Compile and execute basic C programs
- Write programs with formatted input and output with printf() and scanf()
- Write programs using basic pointers and structures
- Write programs using arrays

Submission: Submit `manipulator.c` to the submit web site

Test Script `./test.sh`

Compilation: `gcc -Wall -g manipulator.c -o manipulator`

You will write a single program called manipulator which will take in user input to construct an array, and then perform basic manipulations over that array as desired by the user. Here is a basic run of the program for reference:

```
$ ./manipulator
Enter the length:
4
Enter 4 numbers
(space separated):
1 2 3 4
Choose an operation:
(0) : exit
(1) : print array
(2) : reverse array
(3) : randomize array
(4) : sort array
1
{ 1 2 3 4 }
```

```
Choose an operation:
(0) : exit
(1) : print array
(2) : reverse array
(3) : randomize array
(4) : sort array
2
{ 4 3 2 1 }
```

```
Choose an operation:
(0) : exit
(1) : print array
(2) : reverse array
(3) : randomize array
(4) : sort array
3
{ 4 1 2 3 }
```

```
Choose an operation:
(0) : exit
(1) : print array
(2) : reverse array
(3) : randomize array
(4) : sort array
4
{ 1 2 3 4 }
```

Part 1: Read Input from the User and Check for Errors (20 points)

The first part of the lab is to get user input, error check that input, and populate the array. See the TODO PART 1 comments to direct you to the appropriate places to work. Below is some additional information to help you complete this task.

Error checking `scanf()` Recall that in C we write output and read input using format strings. In this part of the lab you will write a simple program that takes input from the user and manipulates that input in some way depending on the user choice. You must also error check user input. From the `scanf()` man page:

```
RETURN VALUES

These functions return the number of input items assigned.  This can be fewer than provided
for, or even zero, in the event of a matching failure.  Zero indicates that, although there
was input available, no conversions were assigned; typically this is due to an invalid input
character, such as an alphabetic character for a `%d' conversion.  The value EOF is returned
if an input failure occurs before any conversion such as an end-of-file occurs.  If an error
or end-of-file occurs after conversion has begun, the number of conversions which were
successfully completed is returned.
```

If a user enters something that is not a number, then `scanf()` will return 0, indicating that it could not properly convert the input into a number given the `%d` format directive. We can easily check for such a condition in our code:

```
int a, res;
printf("Enter a number:\n");
res = scanf("%d", &a);
if( res == 0){
    printf("ERROR: Invalid input");
    //exit, return, or take some other action
}
```

Instructions

Complete the TODO labels that request user input for the user:

- Read in the desired length of the array to be manipulated. If the user input is not a number, return 1 and exit with the following print statement: `printf("ERROR: Invalid input\n");`
- Read in the appropriate number of values into the array and check to make sure that they are valid integers. If the user input is not a number, return 1 and exit with the following print statement: `printf("ERROR: Invalid input\n");`
- Set up the logic for dealing with user choice of operations. For example, if the user chooses 0, then the loop should break, and if an invalid number is entered the loop should continue with the following error printed: `printf("ERROR: Invalid number. Choose again.\n\n");`

Here is some sample output with the error conditions:

```
$ ./manipulator
Enter the length:
a
ERROR: Invalid input
```

```
$ ./manipulator
Enter the length:
5
Enter 5 numbers (space
separated):
1 2 a d a m
ERROR: Invalid input
```

```
$ ./manipulator
Enter the length:
5
Enter 5 numbers (space
separated):
1 2 3 4 5
Choose an operation:
(0) : exit
(1) : print array
(2) : reverse array
(3) : randomize array
(4) : sort array
a
ERROR: Invalid input. Choose
again.
```

Part 2: Print the Array (20 points)

For this part of the lab, complete the `print_array()` function and properly pass the arguments to the array.

Passing Arrays as Function Arguments

When you pass an array as an argument to a function, since you are only passing a reference to the data, you are not providing any information about the *length* of the array. To manage array size, it is customary to pass both the reference to the array and the *size* of the array to the function to avoid out of bounds references:

```
void foo(int a[], int a_length){
  //...
}
```

Requirements

- Complete the `print_array()` function and properly call it in the `main()` function when the user provides option `1`.
- The output format of the array must match the following example: `{ 0 1 2 3 4 }`
- The entire array must be printed.
- Numbers must be separated by spaces.
- Use a leading and trailing brace -- `{` and `}` respectively (also space-separated).
- Pass both the array itself and the array length as arguments to `print_array()`

Sample Output:

```
$ ./manipulator
Enter the length:
5
Enter 5 numbers (space separated):
0 1 2 3 4 5
Choose an operation:
(0) : exit
(1) : print array
(2) : reverse array
(3) : randomize array
(4) : sort array
ERROR: Invalid number. Choose again.
```

```
Choose an operation:
(0) : exit
(1) : print array
(2) : reverse array
(3) : randomize array
(4) : sort array
1
{ 0 1 2 3 4 }
```

Part 3: Reverse the Array (25 points)

Directions

- Complete the `reverse_array()` function and call it from `main()` when the user enters `2`.
- After reversing, print the array.

Sample output:

```
$ ./manipulator
Enter the length:
5
Enter 5 numbers (space separated):
0 1 2 3 4

Choose an operation:
(0) : exit
(1) : print array
(2) : reverse array
(3) : randomize array
(4) : sort array
1
{ 0 1 2 3 4 }

Choose an operation:
(0) : exit
(1) : print array
(2) : reverse array
(3) : randomize array
(4) : sort array
2
{ 4 3 2 1 0 }
```

Part 4: Randomize the Array (25 points)

Random Integers A random number generator is built into the C standard library: `random()` returns a random integer between 0 and 2^{64}-1. For the purposes of this lab, you'll only need random numbers in the range of the length of the array. To properly bound the random numbers being generated, use the modulo operator, `%`:

```
int r = random() % bound; //produce random number between 0 and bound-1
```

Seeding Random Numbers As you test your program, you may want your random numbers to be deterministic -- that is, be random but predictably random so that, for each run, you get the same random numbers. To do this, we *seed* the random number generator like in the example. This is only done once, in `main()`.

```
srandom(1992); //seed random number generator with seed 1992
```

Pseudocode for array randomization Randomizing an array of values can be accomplished by performing random swaps of items in the array. Here is some pseudocode to use:

```
foreach index i in array:
  choose a random index j
  swap array[i] with array[j]
```

Directions

- Complete the `randomize_array()` function and properly call it when the user selects option 3.
- Your randomization routine must execute in the manner of the pseudocode above.

Sample output

```
$ ./manipulator
Enter the length:
5
Enter 5 numbers (space separated):
0 1 2 3 4

Choose an operation:
(0) : exit
(1) : print array
(2) : reverse array
(3) : randomize array
(4) : sort array
3
{ 2 3 0 4 1 }
```

Part 5: Sort the Array (10 points)

You may or may not have had experience writing sorting functions before, so you are encouraged to look at something like the insertion sort article on Wikipedia, which is perhaps the easiest to implement. Your sorting implementation does *not* need to be optimally efficient.

Requirements Complete the `sort_array()` function and properly call it when the user selects option 4.

Sample output:

```
$ ./manipulator
Enter the length:
5
Enter 5 numbers (space
separated):
0 1 2 3 4
Choose an operation:
(0) : exit
(1) : print array
(2) : reverse array
(3) : randomize array
(4) : sort array
3
{ 3 1 2 4 0 }
```

```
Choose an operation:
(0) : exit
(1) : print array
(2) : reverse array
(3) : randomize array
(4) : sort array
4
{ 0 1 2 3 4 }
```

```
Choose an operation:
(0) : exit
(1) : print array
(2) : reverse array
(3) : randomize array
(4) : sort array
0
```