CSE333 Exercise 2

Out: Friday, January 7

Due: Wednesday, January 12 by **11 am PST**

Rating: 2 (note)

Goals

- Write code that uses pointers and data representations
- Examine the relationship between pointers and arrays
- Use format specifiers to handle printing fixed-width integers

Background

As discussed in lecture, arrays and pointers are closely related:

- When used in an expression, an array name *evaluates* to a pointer
- Array subscripting notation is actually pointer manipulation: `ar[i]` ↔ `*(ar+i)`
- An array argument and a pointer argument are functionally equivalent in that the function receives a copy of the pointer

Problem Description

Write a C program (`ex2.c`) that does the following:

- Contains a function called `DumpBytes` with prototype `void DumpBytes(void* pData, int32_t byteLen)`. The `void` pointer can be thought of as a generalized form of an array of bytes and `byteLen` can be thought of as the length of the "array." This function should print out the length, the address passed in, and the bytes of memory as **exactly two digits each in lowercase hexadecimal**, *e.g.*,

  ```
  The 4 bytes starting at 0x7fff1081856c are: ff 01 30 4e
  ```

  - Recall that the address printed may vary from execution to execution due to security measures such as stack randomization.

  - For the case of `byteLen = 0`, it's *ok* to have a space at the end of the output.
  - For the case of `byteLen > 0`, there should *not* be a space at the end of the output.

- Contains a function called `CopyAndSort` that accepts, in order, two arrays of `uint8_t`'s (*i.e.*, two arrays of bytes) and an array length as arguments; you should assume the length of the two arrays are the same. The function should (1) call `DumpBytes` on the first array and its `sizeof` and (2) iterate through the entries of the first array and copy the entries into the second array in non-descending (*i.e.*, ascending with duplicates allowed) sorted order.
  - You should not use any library functions that would perform the sort for you (*e.g.*, `qsort()`), however you may use any type of sort. We would recommend using **insertion sort**.

- Completes the `main` function found below that sorts a local array **{3, 2, 0, 8, 17, 6, 10, 7, 8, 1, 12}** and dumps the bytes of the arrays along with some other variables using the two functions that you wrote. Your main should match the provided code ***exactly***, with the exception of filling in the missing parameters:

```
int main(int argc, char* argv[]) {
  int32_t int_val = 1;
  float   float_val = 1.0f;
  uint8_t arr_unsorted[] = {3, 2, 0, 8, 17, 6, 10, 7, 8, 1, 12};
  uint8_t arr_sorted[]   = {0, 0, 0, 0,  0, 0,  0, 0, 0, 0,  0};

  DumpBytes(&int_val, sizeof(int_val));
  DumpBytes(&float_val, sizeof(float_val));
  DumpBytes(arr_unsorted, _____);
  CopyAndSort(arr_unsorted, arr_sorted, _____);
  DumpBytes(arr_sorted, _____);

  return EXIT_SUCCESS;
}
```

Implementation Notes

DumpBytes

- You will want to match the formatting shown in the example given above *exactly*, including spacing and capitalization.
- You will need to convince the compiler to let you access bytes in memory starting from a `void*`.
- You will need to use format specifiers in `printf` to print out a pointer value as well as a `uint8_t` in lowercase hexadecimal. As a **hint**, take *inspiration* from the following code:

```
uint8_t a_byte = 0xD1;
printf("The byte is: %02" PRIx8 " -- enjoy!\n", a_byte);
```

CopyAndSort

- The array lengths and subscripts can be stored in variables of type `int`.
- Depending on your implementation, you may get a compiler warning stating: `'sizeof' on array function parameter...` . We encourage you to stop and think why gcc believes it is worth warning you about this. It is

fine for your submission to generate this warning when compiled; however, you should fix any other compiler warnings you get in your code.

Style Focus

Don't forget that all of the good practices from previous exercises still apply!

General

For the sake of our autograder, make sure that your function names match the specifications *exactly*, including capitalization. You should write comments explaining the behavior and purpose of functions you define.

Program Layout

As with the previous exercise, make sure that you organize and comment your functions in such a way that follows the best C practices.

Format Specifiers

Utilize the correct format specifiers to avoid implicit casts and to increase the portability of your code.

Constants

Avoid "magic numbers" (*i.e.*, unnamed numerical constants) where possible. Use predefined constants (*e.g.*, `EXIT_SUCCESS`), if available, or use `#define` to define/name any integer constants that have a clear and specific use. Use `sizeof` to increase the portability of your code.

Submission

You will submit: `ex2.c` .

Your code must:

- Compile without errors or warnings on CSE Linux machines (lab workstations, `attu` , or CSE home VM).
- Have no runtime errors, memory leaks, or memory errors (`gcc` and `valgrind`).
- Be contained in the file listed above that compiles with the command:

```
bash$ gcc -Wall -g -std=c17 -o ex2 ex2.c
```

- Have a comment at the top of your `.c` file with your name(s) and CSE or UW email address(es).

- Be pretty: the formatting, modularization, variable and function names, commenting, and so on should be consistent with class style guidelines. Additionally, the linter shouldn't have any complaints about your code (`clint.py`).
- Be robust: your code should deal with hard-to-handle/edge cases and bogus user input (if there are any) gracefully.

Submit your code on Gradescope (https://www.gradescope.com). Don't forget to add your partner if you have one.

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

UW Site Use Agreement (//www.washington.edu/online/terms/)