Man's mind, once stretched by a new idea, never regains its original dimensions. ~*Oliver Wendell Holmes*

CS1010 Programming Methodology

Week 9: Pointers and Functions with Pointer Parameters (Selected Answers)

Run programs **q1a_ans.c**, **q1b_ans.c** and **q2_ans.c** for questions 1 and 2.

3. **Lab1 Ex2: Surface area and Longest Diagonal of a Box**

   In Lab 1 Exercise 2, you wrote two functions **compute_surface_area(int, int, int)** and **compute_diagonal(int, int, int)** to compute the surface area and longest diagonal of a box respectively. The program is available as **box.c** which you may copy from the cs1010 account.

   ```
   cp  ~cs1010/discussion/prog/week9/box.c  .
   ```

   Can you combine the two functions into one, called **compute_surface_area_and diagonal()**, which passes back both the surface area and length of the longest diagonal?

   Answer: See **box_ptr_ans.c**

   Highlight the following to the students:

   Function prototype:

   void compute_surface_area_and_diagonal(int, int, int, **int ***, **double ***);

   Calling the function:

   compute_surface_area_and_diagonal(length, width, height, **&area**, **&diagonal**);

   Function header:

   void compute_surface_area_and_diagonal(int length, int width, int height,
   int *areaPtr, **double *diagonalPtr**);

   Inside the function body:

   ***areaPtr** = 2 * …;
   ***diagonalPtr** = sqrt(…);

   [Note: Using the suffix 'Ptr' in a pointer variable name is my personal style, just to make the code clearer, because otherwise I may get confused. Some books recommend the suffix '_p'. Up to students whether they want to follow this or not.]

III. Design Issues: Programming Methodology and Cohesion

5. After attending CS1010 lecture last week and learning about function with pointer parameters, Brusco is so excited that he replaced this GCD function:

```
// Returns the GCD of a and b
// Precond: a>=0, b>=0 and not both = 0
int brusco_gcd(int a, int b) {
    int remainder;

    while (b != 0) {
         remainder = a % b;
         a = b;
         b = remainder;
    }

    return a;
}
```

with the following function:

```
void brusco_gcd(int a, int b, int *answer) {
    . . . // body of the function same as above

    *answer = a;
}
```

He did not make a wise move? Why?

Given a choice between a function that returns a value and a function that takes in an address parameter, choose the former.

Answer:

He made a bad move. Unless there is good reason to pass an address parameter to a function, one should stick with the simpler function to return the answer, especially when there is only one value to return.

Rewriting the original function with this new function has another disadvantage. The caller would have to declare a variable, and pass the address of that variable into the new function. What if there is really no intention/need to have such a variable in the caller in the first place, as shown below?

```
printf("The GCD is %d\n", brusco_gcd(num1, num2));
```

As always, there are exceptions. Sometimes, we may want the function to pass back the result via an address variable, and at the same time, have it return something (usually an integer) to indicate some status code of the function. For example, the GCD function could be defined as **int gcd(int a, int b, int *answer)** such that the gcd is returned via the pointer parameter answer, and the function returns 0 to indicate a good run, or -1 to indicate some error (for example, when both parameters a and b contain zero.) But in CS1010 the students are not required to write such applications, unless explicitly stated. ← This last paragraph is a little advanced and you may not want to discuss this with your students.

6. After learning in question 5 above that he should stick to his old function instead of using pointer parameter in his GCD function, Brusco, being a very inquisitive and adventurous student (and we all love such students!), tried another new version:

```
void brusco_gcd(int a, int b) {
    . . .

    printf("The GCD is %d\n", a);
}
```

His reason being: since the answer (variable a) is to be returned to the caller and get printed anyway, why can't he just save the returning part (and hence make the function a void function) and print the answer inside the function instead?

Comment on his move.

Do not mix computation task and input/output task in a single function.

Answer:

Again, that is a bad move (but we still love Brusco). Now, a function should perform **one specific task**, and not a mixture of tasks. Such a function is **cohesive**.

We should refrain from mixing a computation task and an input/output task in a function. The GCD is a computation task. Printing the answer is an output task. Hence both should not be present in a function.

Moreover, another good reason to leave out the printing task is **reusability** of code. What if an application needs to compute the GCD of two numbers as part of a bigger algorithm? Having the printf() statement in the GCD function would upset such a plan.

Hence, leave out the printing; just pass the computed result to the caller and let the caller decide what to do with the returned value, whether to print it or use it for the next step in a bigger algorithm.

7. **Lab 1 Ex2: Surface area and Longest Diagonal of a Box – Revisit**
In question 3 you attempted to combine the two functions **compute_surface_area()** and **compute_diagonal()** into a single function **compute_surface_area_and_diagonal()**.

Compare the two approaches. Which one do you think is more desirable in terms of good programming methodology?

Aim for cohesive function, a function that does a single task.

Answer:

In general, writing separate functions is better, because as we've explained in question 5 above, a function should be cohesive, i.e. it does one task.

Another reason is that in this case, the caller would have to declare two variables, one for the surface area and another for the diagonal. As explained in the answer of question 4, such variables might not be needed.

There are exceptions, of course. One situation is where the 2 (or more) values to be returned (via the pointer parameters) are so intimately related to one another that they are considered as one group of information. Then it makes sense to use function with pointer parameters.

Another situation is where efficiency is concerned, and when it is instructed that it takes priority over cohesion. For example, in computing the minimum and maximum values of an array, we should write 2 separate functions, one to compute the minimum, and another to compute the maximum, according to good programming practice. However, writing just one function to compute both minimum and maximum is more efficient (especially for large array), since we scan the array just once, instead of twice. Therefore, sometimes when there is a trade-off we need to make a judgment on which approach to adopt. There is no hard-and-fast rule.