Errata for

Programming, Problem Solving, and Abstraction with C (revised 2013 edition)

by Alistair Moffat

as at **November 29, 2018**

Preface Preface to the Revised Edition

page IX
This one is a recursive errata, which gave me a chortle of amusement when I spotted it: "An errata page listing known defects in the book appears at `http://www.csse.unimelb.edu.au/~alistair/ppsaa/errata2.pdf`" should (now that my web site has been moved) say "An errata page listing known defects in the book appears at `http://people.eng.unimelb.edu.au/ammoffat/ppsaa/errata2.pdf`"

page X
The same change is required in the URL on this page.

Chapter 1 Computers and Programs

nothing yet

Chapter 2 Numbers In, Numbers Out

page 17
"since the final value `x` printed is not zero" should be "since the final value `z` printed is not zero"

page 22
"is finished when an character that cannot be incorporated" should be "is finished when a character that cannot be incorporated"

page 27
Exercise 2.4: Both `limits.h` and `float.h` are required in order to access the set of six constants that are listed there. And you need to use `%e` format to print `FLT_MIN` and `DBL_MIN`; `%f` will just show them as being zero.

Chapter 3 Selection

nothing yet

Chapter 4 Loops

page 51
The last line, "more important that the particular style", should say "more important than the particular style".

page 60

Exercise 4.3: “Finally. suppose also” should be “Finally, suppose also”

Exercise 4.4: “posible starting point” should be “possible starting point”

Chapter 5 Functions

page 74

“some languages the functional and logic families” should be “some languages in the functional and logic families”

Chapter 6 Functions and Pointers

page 85

“The overall effect would, be the same as if the statement” should be “The overall effect would be the same as if the statement”

page 87

“Which bring us to the” should be “Which brings us to the”

page 90

“modern computers have around a billion of words of memory” should be “modern computers have around a billion words of memory”

page 98

Exercise 6.8: “using integer division..” should be “using integer division.”

Chapter 7 Arrays

page 107

Figure 7.4: the function prototype for `read_int_array` uses a variable called `n`, but the function declaration uses a variable called `maxvals` at that point. While not an error, it *is* confusing. (And note that Figure 7.2 is written assuming that variable is indeed called `maxvals`.)

Chapter 8 Structures

page 129

In Figure 8.1, there should be a semicolon after the declaration of `name`.

page 131

“structures cannot be compared for equality, even of they have the same type” should be “structures cannot be compared for equality, even if they have the same type”

page 136

Figure 8.6: the function prototype for `read_planet_ptr` uses a variable called `one_planet`, but the function declaration uses a variable called `planet` at that point. While not an error, it *is* confusing.

Chapter 9 Problem Solving

page 155

The asymmetry in the way the `if` guard is presented means that the `bisection` function does not converge if the situation is reached in which `f(mid)==0`; that is, if `mid` becomes the exact root by luck while the search is proceeding. To fix the problem, the code needs to deal with three distinct cases rather than two:

```
if (fx1*fmid < 0) {
    /* root is to left of middle */
    x2 = mid;
    fx2 = fmid;
} else if (fx1*fmid > 0) {
    /* root is to right of middle */
    x1 = mid;
    fx1 = fmid;
} else {
    x1 = x2 = mid;
}
```

page 160

Exercise 9.3, "Write a program that deals four random five-card poker hand" should be "Write a program that deals four random five-card poker hands"

Chapter 10 Dynamic Structures

page 181

In Figure 10.11 the use of `char *Ap=(char *A)` is unnecessary, since `void` pointers are also assumed to index bytes. So `Ap` can be declared `void *Ap=A`, or removed entirely (in which case `A` is used to step down the array in the loop).

Chapter 11 Files

page 201

Figure 11.5: There is a bug in this program, because it declares `MAXFILE` elements in the arrays (and `MAXFILE` has the value 10 in the program), but then proceeds to access them via index `i` which ranges from 1 to $\texttt{argc} - 1$ in each of the loops. Hence, if there are ten files specified on the commandline, the program will commence with $\texttt{argc} = 11$, and an erroneous array index of 10 will be used. The best correction would be to use `i-1` to index the arrays. And yes, `argc` should also be error tested against `MAXFILES` right at the start of the program, before anything else happens (or else all of the arrays should be dynamic ones via `realloc()`.

Chapter 12 Algorithms

page 206

In Figure 12.1 the midpoint of the search range is found using `mid =`

`(lo+hi)/2`. If the array `A` has a size that approaches the maximum value that can be stored in an integer (and yes, if your computer has enough memory to store an array of that size!), that arithmetic may overflow. If this risk is a concern, then it is more robust to make use of the equivalent computation `mid = lo+(hi-lo)/2`.

page 207
In Table 12.2, in the row labelled "Sorted array", it says that Search in a sorted array is possible in "$O(1)$" time. That is incorrect, and it should say "$O(\log n)$" in that entry.

page 225
"As a third choice, heap sort is also available ... a little slower than both quick sort and heap sort" should be "As a third choice, heap sort is also available ... a little slower than both quick sort and merge sort".

Chapter 13 Everything Else

page 236
Table 13.5 contains two errors: (1) the two additive operators + and − have *higher* precedence than the two shift operators << and >> that are shown above them (that is, the two rows should swap); and (b) the postfix −− and ++ increment/decrement operators have the *same* precedence as the component selection operators in the top row, but the prefix −− and ++ increment/decrement operators have the same precedence as the unary operators in the third row (that is, the second row does not exist).