Comment on Poor Practice in Coding Examples

Conrad Weisert
Information Disciplines, Inc.
Chicago, Illinois

This one-page article was originally published in ACM SIGPLAN (Special Interest Group on Programming Languages) *Notices*, December, 1995, in response to an earlier article by an academic contributor

What's more depressing than a programming instructor who presents "correct" examples based on bad practice? In *Bug Analysis of Pascal Programs* (April, 1994, p. 15) the author posed a simple student problem:

> **Write a program that will take a decimal number and convert it to the corresponding hexadecimal number . . . Your program should read a decimal number on one line and print its hexadecimal equivalent on the next line.**

After discussing the various bugs that appear in typical student solutions, the author offered this "correct" ("more optimal") solution:

```
program dectohex(input,output)
const MAX  = 16;
      BASE = 16;
var  i, j, remainder, number: integer;
     hexno: array[1..MAX] of char;
begin
  readln(number);
  i := 1;
  while (number > 0) do begin
    remainder := number mod BASE;
    number    := number div BASE;
    if (remainder < 10) then
         hexno[i] := chr(ord('0') + remainder)
    else hexno[i] := chr(ord('A') + remainder-10);
    i := i + 1;
    end;
    for j := (i-1) downto 1 do
        write(hexno[j]);
    writeln;
end.
```

If one of my students turned in such an atrocious solution I'd award it a **D** along with a strong suggestion that the student review the basic principles we've been studying. Consider the major flaws:

1. The conversion algorithm is all mixed up with the input-output, violating long established principles of *coupling* and rendering the routine useless in any real-life context. The actual *result* of the conversion is never stored at all!
2. Bidirectional conversions between character codes and integers are a pointless and ugly complication. A simple table[1] (`'0123456789ABCDEF'`) takes less room, runs faster, and is far easier to understand and debug.
3. The lack of commentary combined with cryptic names leaves the reader to decipher the roles of MAX, i, and j.

In addition to those extremely serious shortcomings, I'd mention a few merely moderately serious ones:

4. The scope of the actual conversion routine is unnecessarily constrained, since there's no reason to limit it to *hexadecimal* conversion. Making BASE a symbolic constant is certainly better than hard coding "16", but if it were a *parameter*, conversion to other number bases up to 16 would be a free by-product.
5. It's misleading to refer to the input as *decimal*, since the actual conversion algorithm works on any internal *integer* representation (usually binary). Indeed, we might want to use this algorithm to convert an integer *to* decimal.
6. Printing one character at a time is not only crude and grossly inefficient, but in some contexts won't give the desired output, e.g. at the end of a line.

We sometimes hear an instructor defend a bad example in the early part of an introductory course on the grounds (a) that we're just trying to teach the stored-program concept and (b) that the students don't yet know enough to cope with niceties like function calls. That's a weak excuse. Even with a limited repertoire of programming tools, we can surely avoid many if not all of the shortcomings noted above.

Even in their first week of exposure to programming students should gain an appreciation of the difference between good and bad practice. Examples like this one are all too common in articles, textbooks[2], and classroom handouts. They do a disservice to the students and ultimately to our profession.

---

[1] Since the point of neither the original article nor this comment was how to code integer conversion, I won't include a *good* example here. I'll gladly FAX one to anyone who asks me for it.

[2] Kernighan & Plauger's 1974 classic *The Elements of Programming Style*, took all of its bad coding examples from published textbooks! Twenty years have brought no improvement. Indeed in the recent flood of books on object-oriented and visual programming techniques good practice seems to be the exception.