Modern $\neq$ Better

Roland Backhouse

November 28, 2003

1 Testing Programming Texts

Over the last twenty years there has been a massive increase in the number of so-called "computer science" degrees, the number of academic staff in "Computer Science" departments, and the number of "computer science" graduates. The result has not been an improvement in standards. Quite the opposite; the scientific principles on which computing is based are being increasingly ignored. Nowhere is this more evident than in the teaching of elementary programming.

There is a small number of basic programming principles with which all programmers should be conversant. These are sequential decomposition, case analysis, and the use of invariants and bound functions in the design of loops.

Binary search is an elementary programming problem that illustrates all three principles. A first test of any text on elementary programming is whether or not its authors are able to implement binary search correctly. Sadly, this is not always the case. A correct implementation of binary search involves the design of just five components, each a simple assignment or test. Some authors succeed in getting the correct implementation, either because they have copied it from elsewhere, or by chance. A decreasingly small number demonstrate how to construct the components. And some authors actually manage to get such an elementary program wrong!

This note records the errors made in two recent textbooks on programming. The message is clear. So-called "modern" programming texts are not an improvement on older texts.

2 Tymann and Schneider

The following is taken from "Modern Software Development Using Java" by Paul T. Tymann and G. Michael Schneider. The book is published by Thomson, with publication date 2004. (In fact, the date of publication is 2003. Every effort has been made to reproduce the program exactly, including the English errors in the comments.)

```
/**
 *  Search the given array for the given integer using a binary
 *  search.  This method assumes that the elements in the array
 *  are in sorted order.  If the element is found, the method
 *  returns the position of the element, otherwise it returns -1.
 *
 *  @param array The array of integers to search
 *  @param target The integer to search for
 *
 *  @return target's position if found, -1 otherwise
 */
public static int search( int array[], int target ) {
    int start = 0;          // The start of the search region
    int end = array.length;  // The end of the search region
    int position = -1;       // Position of the target

    // While there is still something list left to search and the element
    // has not been found
    while ( start <= end && position == -1 ) {
       int middle = (start + end) / 2 ; // Location of the middle

       // Determine whether the target is smaller than, greater than,
       // or equal to the middle element
       if ( target < array[ middle ] ) {
          // Target is smaller; must be in left half
          end = middle - 1;
       } else if ( target > array[ middle ] ) {
          // Target is larger; must be in right half
          start = middle + 1;
       } else {
          // Found it!!
          position = middle;
       }
     }

     // Return location of target
     return position;
}
```

This implementation fails with an array-bound error if the length of the array is 1, and the value of target is greater than the single array element (for example, if the array element is 0 and the target is 1). We leave the reader to check.

3 Winder and Roberts

The following is an implementation of binary search taken from Winder and Roberts, “Developing Java Software”, a second edition of which was published in 2002 by John-Wiley and Sons.

```
/**
 *  The statically accessible sort operation
 *
 *  @param v the sorted array of <code>Object</code>s to be
 *  searched.
 *
 *  @param o the object to be searched for.
 *
 *  @param c the <code>Comparator</code> used to compare the
 *  <code>Object</code> during the search process.  Must either be
 *  "less than" or "greater than" and the same comparator that
 *  defines the order on the array.
 *
 *  @return index of the item or -1 if it is not there.
 */
public static int execute(final Object[] v,
                          final Object o,
                          final Comparator c)
{
   int hi = v.length ;
   int lo = 0 ;
   while (true)
   {
        int centre = (hi + lo) / 2 ;
        if (centre == lo)
        {
           //
           //  Only two items left to test so it is either centre
           //  or centre+1 or it is not in.  This is an exit
           //  point of the infinite loop.
           //
           return ( v[centre].equals(o)
                    ? centre
                    : ( v[centre+1].equals(o)
                      ? centre+1
                      : -1)) ;
         }
         if (c.relation(v[centre], o))
         {
            lo = centre ;
         }
         else if (c.relation(o, v[centre]))
         {
            hi = centre ;
         }
         else
         {
            return centre ;
         }
   }
}
```

The code in this case is much more complicated. The following is a simplified implementation. No essential changes have been made to the program. Changes made are: `(... ? ... : ...)` construct replaced by an `if` statement, and the generic `Comparator` methods instantiated to comparisons of integer values.

```
{
   int hi = v.length ;
   int lo = 0 ;
   while (true)
   {
        int centre = (hi + lo) / 2 ;
        if (centre == lo)
        {
           //
           //  Only two items left to test so it is either centre
           //  or centre+1 or it is not in.  This is an exit
           //  point of the infinite loop.
           //
           if (v[centre] == o)
              { return centre; }
           else if (v[centre+1] == o)
              { return centre+1; }
           else
              { return -1; }
         }
         if (v[centre] < o)
           { lo = centre ; }
         else if (o < v[centre])
           { hi = centre ; }
         else
           { return centre ; }
   }
}
```

This implementation fails if the array v has length two, v[0] is 10, v[1] is 20. Value of object sought o is 30.

4 Discussion

That authors of programming texts are unable to implement binary search correctly is truly lamentable. The danger signs stare you in the face. Tymann and Schneider include verbose comments which add nothing to the program text, since they simply state in words what the program does (e.g. “While there is still something list left to search" (*sic*)). Winder and Roberts have a spurious case analysis on whether there are two items left to search (which, as demonstrated by the example input, they get wrong anyway). Most significantly, both texts use MVN's —which means "*misleading* variable names", although the authors would argue that they are using "meaningful" variable names— . They do not make the function of their variables clear; as a result, they use them inconsistently. Their errors are easily avoided by identifying, as a first step in the design, invariant properties of each of the variables, as every true "computer scientist" should know. Sadly, these texts are typical of "modern" texts on programming.