Engineering Design for Embedded Systems: Assignment 4, Extra Questions

Not for credit; recommended date February 4, 2013.

1 Debugging Selection Sort

You have been given responsibility for a number of programs developed by a fellow programmer who is no longer with the company. One of the programs is shown below. It implements selection sort. The notes left by the programmer indicate that the program does not produce correct results.

Task. State whether this is indeed the case. If yes, show an input that produces an incorrect result. Describe the steps that you used to identify the bug. Show a change to the program that fixes the bug.

```
public class SelectionSort
{
  public static void sort (int[] data)
  {
    int minIndex;
    for (int index = 0; index < data.length - 1; index++)
    {
      minIndex = min(data, index);
      swap(data, minIndex, index);
    }
  }

  public static int min(int[] data, int start)
  {
    int minIndex = start;
    for (int index = start; index < data.length - 1; index++)
    {
      if (data[index] < data[minIndex])
      {
        minIndex = index;
      }
    }
    return minIndex;
  }

  public static void swap(int[] data, int index1, int index2)
  {
    int tmp = data[index1];
    data[index1] = data[index2];
    data[index2] = tmp;
  }
}
```

2 Debugging a Binary Search Implementation

ECE150 introduced the topics of searching and sorting. One of the searching algorithms discussed was the binary search algorithm, applicable to sorted arrays. The code below shows an implementation of the algorithm written by programmer X. The implementation takes two parameters: a reference to an array of `int`s, and the `int` to be searched for. It should return the index of the array element that contains the `int`, or -1 if the `int` is not found in the array.

The implementation was given to the test group. The test group reported that the program failed on test case 09. In this test case, the test array was 99 integers long:

```
int[] testArr09 = {1,2,3, ... ,98,99};
```

Invoking `binaryX(testArr09, 2)` failed to return the index of the element, i.e. 1.

The algorithm is simple and the method is small enough that programmer X could identify the bug by carefully reading the code. Assume that he did not.

Questions for Thought. What debugging tactics would you recommend to him/her to localize and identify the fault (bug)? What diagnostic information should be obtained? How would its analysis help?

Task. After having written down answers to the questions above, apply the tactics that you have suggested. Keep a record of the changes that you apply to the implementation and the test cases that you investigate. Identify and correct the bug, explaining how the information that you collected helped.

```
public static int binaryX(int[] a, int target) {
    if (a == null) {
        return -1;
    }
    int lo = 0;
    int hi = a.length - 1;
    int mid;
    while (hi >= lo) {
        mid = lo / 2 + hi / 2;
        if (a[mid] == target) {
            return mid;
        }
        if (a[mid] > target) {
            hi = mid - 1;
        }
        else {
            lo = mid + 1;
        }
    }
    return -1;
}
```

3 Assertions (Insertion Sort)

Consider the insertion sort discussed in ECE150. The source for a static method `sort (int [] data)` implementing this sort is shown below. As you can see, the program does a lot of data shuffling. This may result in errors. There are two principal categories of errors that are of concern:

1. the resulting array may not be sorted, and
2. an item in the original array may get "lost", that is, overwritten by another item.

Task. Add assertions which detect occurences of these errors.

As an additional challenge, show a buggy version of Insertion Sort that could cause (1) and another for (2).

Note: A strict check for (2) would require that a separate copy of the original contents of the array be kept. For larger arrays, this may be expensive both in memory and execution time. In your answer, use a weaker, but less expensive check.

```
class InsertionSort
{
    public static void sort( int[] data )
    {
        for( int index = 1; index < data.Length; index++ )
        {
            int position;
            int tmp = data[index];
            for( position = index; position > 0; position-- )
            {
                if( tmp > data[position - 1] )
                {
                    break;
                }
                else
                {
                    data[position] = data[(position - 1)];
                }
            }
            data[position] = tmp;
        }
    }
}
```