

ICS H23 Spring 2008  
Honors Introduction to Computer Science III  
Midterm 1 Answers

Comments on the grading are given in smaller type after the answers.

**Part I**

Version A: 1. B 2. A 3. E 4. B 5. A 6. E 7. D 8. D 9. C 10. C

Version B: 1. A 2. B 3. E 4. B 5. A 6. E 7. B 8. D 9. C 10. C

Each question was worth 4 points.

**Part II**

11. Here is one example of a valid answer.

```
if (L ≠ null) {
    x = L;
    while (x.next ≠ null) {
        x.next.prev = x;
        x = x.next;
    }
    // Now x is the last node in the list
    x.next = L;
    L.prev = x;
}
```

Examples of deductions are as follows.

- 3: not handling the case of an empty list.
- 3: dereferencing the **null** pointer; I just took off 2 points if this only happened in the case of a one-node list.
- 3: not correctly initializing the pointer that walks through the list. (But I just took off 2 points if the initialization statement was present but inside the loop.)
- 3: not correctly advancing the pointer that walks through the list.
- 4: not setting the prev fields correctly as the pointer walks through the list; if just one of the prev fields was missed, I took off 2 points.
- 3: not correctly determining when to terminate the loop.
- 4: not doing the final two pointer changes to make the list circular; if one was done correctly but the other was not, I took off just 2 points.

I tried not to deduct twice for the same mistake; for example, if the code did not check for the empty list this could cause it to dereference the **null** pointer, but I just took off a total of 3 points for this. I did not take off points for extra code that did not hurt the correctness of the algorithm, as long as it did not increase the  $\Theta$ -notation for the time complexity. Regardless of the deductions, I did not go below a total score of 2 for an answer that had any good ideas, or below 3 for an answer that had any good ideas and some explanation. One answer left the original list unchanged but made a copy of one meeting the new requirements, and received 11 points.

12. **\*\*B\*DDBA\*\*E\*FFECCA**

Answers that were right except that one letter was wrong or omitted received 13 points. Some answers seemed to skip the last print statement in the given code (so the data was printed in inorder but not postorder), which would give a string of **\*\*B\*DA\*\*E\*FC**. An answer that was like this but had one extra **\*** received 7 points. Answers that were like this but had the wrong number of **\***'s and the letters not in inorder received 2 or 3 points; e.g., **\*\*B\*D\*A\*C\*E\*F** got 3 and **\*\*BDA\*F\*E\*C** got 2.

13. Here is one example of a valid answer. It keeps track of the depth of the nodes and only adds in the data of nodes at an even distance from the root.

```
int addEven(x) {
    return add(x, 0);
}

int add(y, d) {
    /* d will be the depth of y in the tree */
    if (x == null) return 0;
    if (d mod 2 == 0)
        sum = x.data;
    else sum = 0;
    sum = sum + add(x.left, d + 1) + add(x.right, d + 1);
}
```

Here's a second approach based on your solutions. It has two cooperating procedures `addEven` and `addOdd`; one is called at nodes an even distance from the root and the other is called at nodes an odd distance from the root. Only the one for nodes at an even distance adds in the data field of the current node.

```
int addEven(x) {
    if (x != null)
        return x.data + addOdd(x.left) + addOdd(x.right);
    else return 0;
}

int addOdd(x) {
    if (x != null)
        return addEven(x.left) + addEven(x.right);
    else return 0;
}
```

A third approach some people tried was to have `addEven` take care of two levels at a time. With this approach, a single procedure can do the job but one has to be especially careful not to dereference **null**.

```
int addEven(x) {
    if (x != null) {
```

```

    sum = x.data;
    if (x.left ≠ null)
        sum += addEven(x.left.left) + addEven(x.left.right);
    if (x.right ≠ null)
        sum += addEven(x.right.left) + addEven(x.right.right);
    }
else return 0;
}

```

Here are examples of point deductions, assuming the answer seemed to suggest a reasonably sound approach to the problem.

- 5 to -6: if a fundamentally incorrect approach was used to calculate the depth.
- 4: having a good basic structure but then not nesting the **if** statements properly, so that only the top two levels of the tree are examined.
- 3: having a good basic structure that is set up to handle the case of even depth and odd depth correctly, but then adding in the data in either case, or not adding it in at all.
- 3: using the third approach above, but missing some of the extra checks to avoid dereferencing the **null** pointer; however, if the code looked like you did some extra tests to try to avoid this I just took off 2 points.
- 2: summing the values in nodes at levels 0,1,3,5,7,...
- 2: giving some procedures, but never saying how they are used by addEven.
- 1.5: doing the problem correctly but accidentally adding in the data field of the root twice.
- 1: doing the problem correctly but leaving out “**else return 0**” once or twice.
- 2: writing  $\text{addEven}(x + y)$  when  $\text{addEven}(x) + \text{addEven}(y)$  was meant; however I only took off one of these points if other deductions had already taken away a substantial part of the 15 points.

### Part III

14. We can color each component independently. For each component, we can arbitrarily choose the color of one vertex in that component, but then the colors of all the other vertices in that component are determined. Thus if there are  $c$  components, the number of possible colorings is the number of ways to make  $c$  independent 2-way choices, which is  $2^c$ . On Version A  $c$  was  $q$ , and on Version B  $c$  was  $p$ .

Only a few people got this one. I gave very little partial credit, but I did give 2/15 points for giving an example with one component and saying it had two colorings, or noting that you could get another coloring by swapping the colors.