Computer Science Foundation Exam

December 18, 2015

Section I A

COMPUTER SCIENCE

NO books, notes, or calculators may be used, and you must work entirely on your own.

SOLUTION

<table>
<thead>
<tr>
<th>Question #</th>
<th>Max Pts</th>
<th>Category</th>
<th>Passing</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>DSN</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>ANL</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>ALG</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>ALG</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>ALG</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

You must do all 5 problems in this section of the exam.

Problems will be graded based on the completeness of the solution steps and not graded based on the answer alone. Credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all be neat.
1) (10 pts) DSN (Recursive Functions)

Write a recursive function that takes the root of a ternary tree (a tree where each node has at most three children) and determines whether all the nodes that have a middle child also have both a left child and a right child. If so, return 1. Otherwise, return 0. Note: If the function with a null input, the output should be 1.

The node struct and functional prototype for this question are:

```c
typedef struct node
{
    char *data;
    struct node *left, *middle, *right;
} node;

int hasProperty(node *root)
{
    if (root == NULL)
        return 1;

    else if (root->middle == NULL)
        return hasProperty(root->left) && hasProperty(root->right);

    else if (root->left == NULL || root->right == NULL)
        return 0;

    else
        return hasProperty(root->left) &&
                hasProperty(root->right) &&
                hasProperty(root->middle);
}
```

Grading:

2 pts for root == NULL base case,
3 pts for handling root->middle == NULL case,
2 pts for handling case where exactly middle isn't NULL but left or right is
3 pts for handing case where none are NULL
2) (10 pts) ANL (Summations and Algorithm Analysis)

Give the big-oh runtimes for each of the following functions in terms of \( n \) and/or \( k \) (where \( k \) is the length of string \( s \)), given that \( \text{strlen}(s) \) is an \( O(k) \) function and \( \text{toupper}(c) \) is an \( O(1) \) function. You may assume that \( s \) is non-NULL and contains at least one character (so, it shouldn’t cause any of the following functions to crash).

```c
void uppercase(char *s)
{
    int i;
    for (i = 0; i < strlen(s); i++)
        s[i] = toupper(i);
}
```

uppercase run time: \( O(k^2) \)

```c
void uppercase_remix(char *s)
{
    int i, length = strlen(s);
    for (i = 0; i < length; i++)
        s[i] = toupper(i);
}
```

uppercase_remix run time: \( O(k) \)

```c
void uppercase_unreliable(char *s){
    int i, j = strlen(s) - 1, m;
    while (i <= j)
    {
        m = i + (j - i) / 2;
        if (rand() % 2 == 0)
        {
            s[i] = toupper(s[i]);
            i = m + 1;
        }
        else
        {
            s[j] = toupper(s[j]);
            j = m - 1;
        }
    }
}
```

uppercase_unreliable run time: \( O(k) + O(\log k) = O(k) \)

```c
void mad_scramble(char *s, int n){
    int i;
    for (i = 0; i < n; i++)
        s[strlen(s) - 1] = rand() % 25 + 'a';
}
```

mad_scramble run time: \( O(nk) \)

Grading: 3 pts uppercase, 2 pts uppercase_remix, 3 pts uppercase_unreliable, 2 pts mad_scramble, each part is all or nothing.
3) (10 pts) ALG (Stacks)

Use a stack to convert the following infix expression to a postfix expression. Please show the state of the stack at the exact point in time when the algorithm reaches the marked locations (A, B, C, and D) while processing the expression.

Equivalent Postfix Expression:

\[
8 * 3 / 2 + 9 - 6 + (3 - (8 * 2)) + 7
\]

Grading: 1 pt for each stack, 6 pts for the expression (take off 1 pt per error in the expression, capping at 6)
4) (10 pts) ALG (Binary Search Trees and Hash Tables)

a) (8 pts) Draw a single binary search tree that gives rise to all three of the following tree traversals:

Inorder: 4 7 8 10 27 30 44 56

Preorder: 4 10 7 8 44 30 27 56

Postorder: 8 7 27 30 56 44 10 4

b) (2 pts) If we insert an element into a hash table using quadratic probing to resolve collisions, what two conditions must be met to ensure that if an open spot exists in our hash table, we will find that spot (rather than getting stuck in an infinite loop)?

1. The table size must be a prime number.
2. The table must be at least half empty.

Grading: 1 pt each
5) (10 pts) ALG (Base Conversion)

Write a function that takes a string \(str\) and an integer \(b\) (where \(2 \leq b \leq 10\)), and returns 1 if \(str\) represents an integer in base \(b\) that is a perfect power of \(b\). For example:

\[
isPower("323", 4);  \quad // \text{Return 0. } 323_4 = 59_{10}, \text{ which is not a power of 4}
isPower("27", 3);  \quad // \text{Return 0. } 27 \text{ is not a valid base 3 integer.}
isPower("plum", 8);  \quad // \text{Return 0. } \text{plum is not a valid base 8 integer.}
isPower("1000", 10);  \quad // \text{Return 1. } 1000_{10} \text{ is a power of 10 } (10^3)
isPower("000001", 2);  \quad // \text{Return 1. } 1_2 = 1_{10}, \text{ which is a power of 2 } (2^0)
\]

**Notes:** You may assume \(b\) is always within the range specified above. Your function must return 0 if \(str\) is NULL or the empty string. Strings may be padded on the left with any number of zeros.

```c
// You must use this function signature. You may write helper functions as needed:
int isPower(char *str, int b);

int intCharToInt(char c) { return c - '0'; }

int isPower(char *str, int b)
{
    int i, length, c, flag = 0;
    if (str == NULL || str[0] == '\0')
        return 0;
    for (i = 0; i < strlen(str); i++)
    {
        // Convert this character to an integer.
        c = intCharToInt(str[i]);

        // If this is not even a valid digit in base b, return 0.
        if (c > b - 1 || c < 0)
            return 0;

        // Key insight: For this to be a perfect power of b, we must
        // have '1' followed by '0''s only.
        if (c == 1)
        {
            if (flag > 0)  // If we have seen more than one '1', it's over.
                return 0;
            flag = 1;
        }
        // Can't have anything but 0's and 1's.
        else if (c != 0)
        {
            return 0;
        }
    }
    return flag;
}
```

Grading: 4 pts - for taking care of invalid cases, 1 pt - returns 1 or 0 for all cases, 2 pts - rejects strings that have anything but 0 or 1, 2 pts - rejects any string with > 1 non-zero char, 1 pt - accepts correct strings