ECE264 Fall 2021 Exam 3

8-10AM, December 16, 2021

Please write your answers on the answer sheets only and **return the answer sheet**. Do not return the questions. Please keep the answer sheet clean. Do not use the answer sheet as your scratch space. The answer sheet should contain **only** the answers.

Do not write anything that is not the answers. Otherwise, you may lose points.

Please use **DARK** ink. If your pen is too light, your answer may not be graded.

When you take this exam, you agree with the Purdue Honor Pledge: **advancing a culture of academic integrity, seeing academic dishonesty and cheating as threats to Purdues reputation**.

Value	Character	Value	Character	Value	Character
48	0	65	A	97	a
49	1	66	B	98	b
50	2	67	C	99	c
51	3	68	D	100	d
52	4	69	E	101	e
53	5	70	F	102	f
54	6	71	G	103	g
55	7	72	H	104	h
56	8	73	I	105	i
57	9	74	J	106	j
		75	K	107	k
		76	L	108	l
		77	M	109	m
		78	N	110	n
		79	O	111	o
		80	P	112	p
		81	Q	113	q
		82	R	114	r
		83	S	115	s
		84	T	116	t
		85	U	117	u
		86	V	118	v
		87	W	119	w
		88	X	120	x
		89	Y	121	y
		90	Z	122	z

1 Binary Tree (30 pts)

Consider a binary tree whose post-order description is
H D E B I J F G C A
The in-order description is
D H B E A I F J C G
Please write down the pre-order descripition of the binary tree and answer these questions.

ANSWER A: (4 pts) What is the root of the tree?

ANSWER B: (6 pts) What is the right child of the root of the tree? Your answer should be a letter between A and H.

ANSWER C: (6 pts) What is the left child of the root of the tree? Your answer should be a letter between A and H.

ANSWER D: (7 pts) If a tree has the same in-order description and post-order description, how large can this tree be (measured by the number of nodes)? Please choose one correct answer. Your answer should be a number 1, or 2, or 3 ... or 9.

1. 0 node
2. 1 node
3. 2 nodes
4. 3 nodes
5. 4 nodes
6. 7 nodes
7. 8 nodes
8. There is no limit
9. None of the above

ANSWER E: (7 pts) If a tree has the same pre-order description and post-order description, how large can this tree be (measured by the number of nodes)? Please choose one correct answer. Your answer should be a number 1, or 2, or 3 ... or 9.

1. 0 node
2. 1 node
3. 2 nodes

4. 3 nodes
5. 4 nodes
6. 7 nodes
7. 8 nodes
8. There is no limit
9. None of the above

For your reference, the methods for generating the three descriptions of a binary tree are shown below. `Node` is the data type for a tree node.

```
void preorder(Node * tn)
{
    if (tn == NULL) { return; }
    print (tn -> data);
    preorder(tn -> left);
    preorder(tn -> right);
}

void inorder(Node * tn)
{
    if (tn == NULL) { return; }
    inorder(tn -> left);
    print (tn -> data);
    inorder(tn -> right);
}

void postorder(Node * tn)
{
    if (tn == NULL) { return; }
    postorder(tn -> left);
    postorder(tn -> right);
    print (tn -> data);
}
```

Answer:
ANSWER A: A
Answer B: C
Answer C: B
Answer D: 8 (no limit)
Answer E: 2 (1 node)

2 Binary Search Tree (25 pts)

Consider the following definition in `tree.h`:

```
1  #ifndef _TREE_H
2  #define _TREE_H
3
4  typedef struct node {
5      int value;
6      struct node * left;
7      struct node * right;
8  } TreeNode;
9
10 #endif
```

and the following functions in `tree.c` for binary search tree:

```
1  #include "tree.h"
2  #include <stdlib.h>
3
4  TreeNode * treenode_construct (int key) {
5      TreeNode * tn = malloc(sizeof(*tn));
6      tn -> value = key;
7      tn -> left = NULL;
8      tn -> right = NULL;
9      return tn;
10 }
11
12 TreeNode * tree_insert (TreeNode * tn, int key) {
13     if (tn == NULL) {
14         return treenode_construct (key);
15     }
16     // There is a bug here: it does not check if key == (tn -> value)
17     // Do NOT fix the bug and answer questions
18     if (key < (tn -> value)) {
19         tn -> left = tree_insert (tn -> left, key);
20     }
21     else {
22         tn -> right = tree_insert (tn -> right, key);
23     }
24     return tn;
25 }
26
27 TreeNode * tree_delete(TreeNode * tn, int key) {
28     // ----Search for the node to be deleted----
```

```
29      if (tn == NULL) {
30          return NULL;
31      }
32      if (key < (tn -> value)) {
33          tn -> left = tree_delete(tn -> left, key);
34          return tn;
35      }
36      if (key > (tn -> value)) {
37          tn -> right = tree_delete(tn -> right, key);
38          return tn;
39      }
40
41      // ----Implement the delete: tn is the node to delete----
42      if (key == (tn -> value)) {
43          // if tn has no child
44          if (((tn -> left) == NULL) && ((tn -> right) == NULL)) {
45              free(tn);
46              return NULL;
47          }
48
49          // if tn has one child: left child
50          if ((tn -> left) != NULL && (tn -> right) == NULL) {
51              TreeNode * sn = tn -> left; // sn is the substitue node
52              free (tn); // free the current root
53              return sn; // return the substitute node
54          }
55          // if tn has one child: right child
56          if ((tn -> left) == NULL && (tn -> right) != NULL) {
57              TreeNode * sn = tn -> right; // sn is the substitue node
58              free(tn); // free the current root
59              return sn; // return the substitute node
60          }
61
62          // if tn have two children
63          // step 1: find sn (substitute node)
64          // sn is the rightmost node on the left subtree of tn
65          TreeNode * sn = tn -> left; // go to the left subtree
66          while ((sn -> right) != NULL) { // get the rightmost node
67              sn = sn -> right;
68          }
69          // step 2: swap the values of sn and tn
70          tn -> value = sn -> value;
```

```
71          sn -> value = key;
72          // step 3: delete sn via recursive function call
73          tn -> left = tree_delete(tn -> left, key);
74          return tn;
75      }
76      return tn;
77  }
```

There is a bug in `tree_insert` function, such that it cannot handle duplicate values properly. Do NOT fix the bug.
We use the given `tree_insert` function to insert nodes 5, 3, 5, 2, 4, 6, 3, 3, 5, 2 (in this order) to an empty binary search tree. Answer questions A and B.

ANSWER A: (5 pts) What is the last number in the pre-order traversal? Your answer should only contain an integer.

ANSWER B: (5 pts) What is the fourth number in the post-order traversal? Your answer should only contain an integer.

After the above insert operations, we then use the given `tree_delete` function to delete nodes 5, 3, 1 (in this order). Answer questions C and D.

ANSWER C: (7 pts) What is the first number in the pre-order traversal? Your answer should only contain an integer.

ANSWER D: (8 pts) What is the fourth number in the post-order traversal? Your answer should only contain an integer.
Answer:
Answer A: 5
Answer B: 3
Answer C: 4
Answer D: 2

3 Bitwise Operation (20 pts)

Consider the following program for bitwise operation.

```
1  #include <stdio.h>
2  #include <stdbool.h>
3
4  // input: arr is an integer array
5  // all but one element in arr occur even times
6  // output: the only integer element that occurs odd times
7  int findOdd (int * arr, int n) {
8      int value = 0;
9      int i = 0;
10     for (; i < n; i++) {
11         value = <--- ANSWER A: fix the code here --->;
12     }
13     return value;
14 }
15
16 // input: an integer value
17 // output: false if value is an odd number, else true
18 bool checkEven (int value) {
19     if (value  <--- ANSWER B: fix the code here --->) {
20         return false;
21     }
22     return true;
23 }
24
25 int main(int argc, char * * argv){
26     int arr[] = {215, 121, 36, 121, 121, 36, 215, 36, 36};
27     int value = findOdd(arr, sizeof(arr)/sizeof(*arr));
28     // print in lower-case hexadecimal
29     // your answer C should only contain a hexadecimal number
30     printf("<--- ANSWER C: %x --->\n", value);
31     if(checkEven(value)) {
32         printf("%d is an even number.\n", value);
33     }
34     else {
35         printf("%d is an odd number.\n", value);
36     }
37     return EXIT_SUCCESS;
38 }
```

The `findOdd` function takes an integer array as input. In the input array, there is one

element that occurs odd times, while all other elements occur even times. The `findOdd` function returns the only element that occurs odd times.

The `checkEven` function takes an integer as input and checks if the integer is an even number.

Hint: the output of the given code is
```
<--- ANSWER C: [ANSWER C] --->
121 is an odd number.
```

ANSWER A: (8 pts) Fill in the code in line 11.
(Hint 1: Think about a case where every number appears in the array exactly two times, except for the number you want, which appears exactly once.)
(Hint 2: Suppose the number you're looking for has a 1 as its seventh bit. What can you say about how many times the seventh bit will be 1 across all the numbers in the array? Suppose the number you're looking for has a 0 as its sixth bit. What can you say about how many times the sixth bit will be 1 across all the numbers in the array?)

ANSWER B: (8 pts) Fill in the code in line 19. For full credit, your answer should only use bitwise operations.

ANSWER C: (4 pts) What does line 30 print? Assume the code works. Your answer should only contain a hexadecimal number.
Answer:
Answer A: value ^ arr[i]
Answer B: & 1
Answer C: 79

4 Linked List Traversals (25 points)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct node
5  {
6    int data;
7    struct node * next;
8  } node_t;
9
10 void printIter(const node_t * n) {
11   while (n != NULL) {
12     printf("%d ", n->data);
13     n = n->next;
14   }
15 }
16
17 void printRecur(const node_t * n) {
18   if (n == NULL) {
19     return;
20   }
21
22   printRecur(n->next);
23   printf("%d ", n->data);
24 }
25
26 void custom(const node_t * n) {
27   if (n == NULL) {
28     return;
29   }
30
31   if (n->next != NULL) {
32     custom(n->next->next);
33   }
34
35   printf("%d ", n->data);
36 }
37
38 int main(int argc, char ** argv) {
39   node_t * head = NULL;
40
```

```
41    int LIST_SIZE = 6;
42
43    // note the index starts at 1 and goes until equal to LIST_SIZE
44    for (int i = 1; i <= LIST_SIZE; i++) {
45      node_t * tmp = head;
46      head = (node_t *)malloc(sizeof(node_t));
47      head->data = i;
48      head->next = tmp;
49    }
50
51    // Visual representation of linked list
52    // head --> 6 --> 5 --> 4 --> 3 --> 2 --> 1
53
54    printIter(head);         // answer Q4.A
55    printf("\n");
56    printRecur(head);        // answer Q4.B
57    printf("\n");
58    custom(head);            // answer Q4.C
59    printf("\n");
60    return EXIT_SUCCESS;
61  }
```

ANSWER A: (5 pts) What does line 54 print in the code above?

ANSWER B: (5 pts) What does line 56 print in the code above?

ANSWER C: (7 pts) What does line 58 print int he node above?

ANSWER D: (8 pts) Suppose line 31 in the code above were changed to `if (1)`. This introduces a bug in the code. Give a value of `LIST_SIZE` in line 41 that would cause a segmentation fault.

Answer:

`A: 6 5 4 3 2 1`
`B: 1 2 3 4 5 6`
`C: 2 4 6`
`D: Any odd number`