ECE264 Summer 2013
Final Exam, July, 2013

Name:

By signing this statement, I hereby certify that the work on this exam is my own and that I have not copied the work of any other student while completing it. I also declare that I will not discuss/share this exam with anybody today. I understand that, if I fail to honor this agreement, I will receive a score of ZERO for this exam and will be subject to possible disciplinary action.

Signature:

You must sign here. Otherwise you will receive a **2-point** *penalty.*

This is an *open-book, open-note* exam. You may use any book, notes, or program printouts. No personal electronic device is allowed. You may not borrow books from other students.

Three learning objectives (recursion, structure, and dynamic structure) are tested in this exam. To pass an objective, you must receive 50% or more points in the corresponding question. If you have passed a learning objective, you will not "unpass" later.

Contents

Learning Objective 1 (Recursion)	Pass	Fail
Learning Objective 3 (Structure)	Pass	Fail
Learning Objective 4 (Dynamic Structure)	Pass	Fail
Total		

1 Recursive Function (6 points, Recursion)

The following C function implements the mathematical formula

$$\begin{aligned} f(0) &= 1 \\ f(1) &= 1 \\ f(n) &= f(n-1) + 1, \text{ if } n > 1 \end{aligned} \tag{1}$$

```
1  int f1(int n)
2  {
3    if ((n == 0) || (n == 1))
4      {
5        return 1;
6      }
7    return f1(n - 1) + 1;
8  }
```

1.1 Recursive Formula (1 point)

Write down the mathematical formula for the following function.

Hint: You must write the terminating condition(s) as well as the recursive equation.

```
1  int f2(int n)
2  {
3    if ((n == 0) || (n == 1))
4      {
5        return 1;
6      }
7    int k;
8    int sum = 0;
9    for (k = 0; k < n; k ++)
10     {
11       sum += f2(k) * f2(n - k - 1);
12     }
13   return sum;
14 }
```

1.2 Recursive Formula (1 point)

Write down the mathematical formula for the following function.

```
1  int f3(int n)
2  {
3    if ((n == 0) || (n == 1))
```

```
 4      {
 5        return 1;
 6      }
 7    int i;
 8    int sum = 0;
 9    for (i = 1; i <= n; i ++)
10      {
11        sum += f3(n - 1);
12      }
13    return sum;
14  }
```

1.3 Trace Recursive Functions (3 points)

Write down the values. You will lose points if you leave a value blank.

If a value is zero, you need to write zero or "0".

n	0	1	2	3	4	5	6
f2(n)	1	1					
f3(n)	1	1					

1.4 Compare Recursive Functions (1 point)

Explain whether it is true that $f2(n) \leq f3(n)$ for any positive integer n.

If the answer is false, you need to find only one value of n that makes $f2(n) > f3(n)$.

If the answer is true, you need to explain why this is *always* true.

"Observation" of the answers in the previous question is **not** a valid answer. Your answer needs to be general for any positive integer.

2 Binary Search Tree (7 points, Dynamic Structure)

When values are inserted into a binary search tree, the tree's shape depends on the order of the values. For example, if the values are already sorted, the binary search tree is equivalent to a linked list. If the values are sorted in the ascending order, all nodes' left children are `NULL`. If the values are sorted in the decending order, all nodes' right children are `NULL`.

This question asks you to take a binary search tree of integers and create a **balanced** binary search tree.

This is the algorithm:

1. Count the number of nodes in the binary search tree.
2. Create an array of integers to store the values in the binary search tree.
3. Traverse the binary search tree and fill the array's elements. The array's elements should be **sorted in the ascending order**.
4. Take the element at the center of the array and insert this element into the new binary search tree. Since this element is at the array's center, there will be the same number (or different by at most one) of nodes on the left size and the right side of this newly inserted node.
5. Divide the array into two parts. Recursively repeat the previous step until all elements in the array have been inserted into the new binary search tree.
6. Release memory occupied by the array;
7. Return the new, balanced, binary search tree.

2.1 Balance Function (5 points)

Complete the `Tree_balance` function.

```
1  #include "tree.h"
2
3  void Tree_countNode(TreeNode * root, int * count);
4  void Tree_storeArray(TreeNode * root, int * arr, int * ind);
5  TreeNode * Tree_insertArray(TreeNode * root, int * arr,
6                              int numNode);
7
8  TreeNode * Tree_balance(TreeNode * root)
9  {
10   int numNode = 0;
11   Tree_countNode(root, & numNode);
12   int * arr = malloc(sizeof(int) * numNode);
13   int ind = 0;
14   Tree_storeArray(root, arr, & ind);
15   TreeNode * newroot = NULL;
16   newroot = Tree_insertArray(newroot, arr, numNode);
17   free (arr);
```

```
18   return newroot;
19 }
20
21 /*
22  * count the number of non-NULL nodes in a tree whose
23  * root is given by the first argument. The second
24  * argument is the address of a counter.
25  * ********************************
26  *               1 point
27  * ********************************
28  */
29 void Tree_countNode(TreeNode * root, int * count)
30 {
31   if (root == NULL)
32     {
33       return;
34     }
35   /* fill the necessary code */
36
37
38
39
40
41
42 }
43
44 /*
45  * Store the tree's values in arr
46  * The elements in arr should be ascending.
47  * ind is the index for the next array element
48  *
49  * Hint: inOrder will visit the tree's nodes in the ascending
50  * order
51  *
52  * ********************************
53  *               1 point
54  * ********************************
55  */
56
57 void Tree_storeArray(TreeNode * root, int * arr, int * ind)
58 {
59   if (root == NULL)
```

```
60      {
61        return;
62      }
63
64
65
66
67
68
69
70  }
71
72  /*
73   * insert the array's elements into the binary search tree starting at
74   * root.
75   * The array's minimum valid index is min.
76   * The array's maximum valid index is max.
77   *
78   * The function stops when min > max.
79   *
80   * Hint: This function should be recursive.
81   *
82   * ********************************
83   *               3 points
84   * ********************************
85   *
86   */
87
88  TreeNode * Tree_insertArrayHelper(TreeNode * root, int * arr,
89                                    int min, int max)
90  {
91    /* Terminating condition: min > max */
92
93
94
95
96
97    /* find the element at the center of the array */
98
99
100
101
```

```
102   /* insert the element at the center to the binary search tree */
103
104
105
106
107
108
109
110   /* insert the first half of the array into the binary search tree,
111      the element at the center should be excluded */
112
113
114
115
116
117
118
119   /* insert the second half of the array into the binary search tree,
120      the element at the center should be excluded */
121
122
123
124
125
126
127
128   return NULL;
129 }
130
131 TreeNode * Tree_insertArray(TreeNode * root, int * arr,
132                             int numNode)
133 {
134   return Tree_insertArrayHelper(root, arr, 0, numNode - 1);
135 }
```

2.2 Post-Order Traversal (2 points)

What is the output of this program?

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include "tree.h"
 4
 5 int main(int argc, char ** argv)
 6 {
 7   #define LENGTH 15
 8   int arr[LENGTH] =
 9     {7, 3, 6, 2, 9,
10     8, 13, 1, 12, 10,
11     15, 4, 11, 5, 14};
12   int iter;
13   TreeNode* t = NULL;
14   TreeNode* t2 = NULL;
15   for (iter = 0; iter < LENGTH; iter ++)
16     {
17       t = Tree_insert(t, arr[iter]);
18     }
19   /*
20    * Tree_printPostorder(t);
21    * printf("\n");
22    */
23   t2 = Tree_balance(t);
24   Tree_printPostorder(t2);
25   printf("\n");
26   Tree_destroy(t);
27   Tree_destroy(t2);
28   return EXIT_SUCCESS;
29 }
```

3 Auto-Resize String (7 points, Structure)

The C programming language does not have a built-in type for strings. Instead, C programs use array of characters for strings. There are some problems of this approach. For example, programmers must always ensure that a string has enough memory to store the characters. This may lead to wasting memory when programmers allocate more space than necessary. When programmers allocate too little space, memory may be corrupted. Also, every string must end with the special character `'\0'`. Moreover, this special character is not counted by the `strlen` function. These problems create much confusion to new C programmers.

This question asks you to create a new data type called `ECEString`. This type automatically allocates and releases memory as needed. Please complete the functions for this type.

You will lose points if your program **leaks memory**. You will lose 1 point for each `malloc` that does not have a corresponding `free`. If a pointer **may be** `NULL`, your program must check whether it is `NULL` before you proceed using the pointer's value.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #ifndef ECESTRING_H
 4 #define ECESTRING_H
 5 typedef struct
 6 {
 7   /* an arry of characters to store data.
 8    * This array does *NOT* store '\0'
 9    *
10    * YOU WILL LOSE 3 POINTS IF data stores '\0'
11    */
12
13   char * data;
14   /* length of the array */
15   int length;
16 } ECEString;
17 /* ------------------------------------------------------------- */
18 /* create an ECEstring object from an array of characters. This array
19    does not end with '\0'.  The length of the arry is provided by the
20    second argument */
21 ECEString * ECEString_create(char * arr, int len);
22
23
24 /* ------------------------------------------------------------- */
25 /* copy an ECEString object from src to dest.
26  *
27  * If dest is NULL, this funcion does nothing and returns NULL
```

```
28  *
29  * If dest is not NULL, check dest's length. If it is different from
30  * src's length, set dest's length to be the same as src's length and
31  * resize dest's data, release the memory already allocated for dest's
32  * old data, allocate memory for the new data, copy the data from src
33  * to dest, return dest.
34  *
35  */
36 ECEString * ECEString_copy(ECEString * dest, ECEString * src);
37
38
39 /* ------------------------------------------------------------- */
40 /* append the data of src to dest's data
41  *
42  * If dest is NULL, this funcion does nothing and returns NULL
43  *
44  * If dest's is not NULL, check dest's length. If it is insufficient
45  * for storing src's data, set dest's length to be large enough to
46  * hold the data, append src's data, and return dest.
47  *
48  */
49 ECEString * ECEString_append(ECEString * dest, ECEString * src);
50
51
52 /* ------------------------------------------------------------- */
53 /*
54  * print the data stored in str
55  *
56  * If str is NULL, the function does nothing
57  * Do not print any character that is not in str's data
58  */
59 void ECEString_print(ECEString * str);
60
61
62 /* ------------------------------------------------------------- */
63 /*
64  * release the memory occupied by str
65  */
66 void ECEString_destroy(ECEString * str);
67
68 #endif
```

Hint: You will not be able to use many built-in string functions, such as `strlen` and

`strcpy`, because `'\0'` is not part of the data.

Implement the functions below.

```
1  #include "ecestring.h"
2  /* 1 point */
3  ECEString * ECEString_create(char * arr, int length)
4  {
5
6
7
8
9
10
11
12
13
14
15
16
17
18 }
19
20 /* 2 points */
21 ECEString * ECEString_copy(ECEString * dest, ECEString * src)
22 {
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
```

```
41 }
42
43 /* 2 points */
44 ECEString * ECEString_append(ECEString * dest, ECEString * src)
45 {
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65 }
66
67 /* 1 point */
68 void ECEString_print(ECEString * str)
69 {
70
71
72
73
74
75
76
77
78
79
80
81
82
```

```
 83 }
 84
 85 /* 1 point */
 86 void ECEString_destroy(ECEString * str)
 87 {
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
100
101
102 }
```

The `main` function and its output are provided for your reference.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include "ecestring.h"
 4 #define STR1LEN 9
 5 #define STR2LEN 7
 6 #define STR1DATA "ECEPURDUE"
 7 #define STR2DATA "2642013"
 8 int main(int argc, char ** argv)
 9 {
10   ECEString * str1 = NULL;
11   ECEString * str2 = NULL;
12   ECEString * str3 = NULL;
13   str1 = ECEString_create(STR1DATA, STR1LEN);
14   str2 = ECEString_create(STR2DATA, STR2LEN);
15   ECEString_print(str1);
16   printf("\n");
17   ECEString_print(str2);
18   printf("\n--A--\n");
19   ECEString_copy(str1, str2);
20   ECEString_print(str1);
21   printf("\n");
```

```
22    ECEString_print(str2);
23    printf("\n--B--\n");
24    ECEString_copy(str3, str2);
25    ECEString_copy(str1, str2);
26    ECEString_print(str1);
27    printf("\n");
28    ECEString_print(str3);
29    printf("\n--C---\n");
30    ECEString_append(str1, str2);
31    ECEString_print(str1);
32    printf("\n");
33    ECEString_print(str3);
34    printf("\n---D---\n");
35    ECEString_destroy(str1);
36    ECEString_destroy(str2);
37    ECEString_destroy(str3);
38    return EXIT_SUCCESS;
39  }
40  /*
41    Output:
42
43    ECEPURDUE
44    2642013
45    --A--
46    2642013
47    2642013
48    --B--
49    2642013
50
51    --C---
52    26420132642013
53
54    ---D---
55  */
```