ECE264 Spring 2014
Exam 3, April 10, 2014

In 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 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.*

Read the questions carefully.
Some questions have conditions and restrictions.

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 this exam.

Contents

Learning Objective 1 (Recursion)	Pass	Fail
Learning Objective 2 (Structure)	Pass	Fail
Learning Objective 3 (Dynamic Structure)	Pass	Fail

Total Score:

This page is intentionally left blank. You can write answers here if you need more space.

1 Recursion: Recursive Function (8 points)

There are unlimited red ($\mathbb{R}$), green ($\mathbb{G}$), and blue ($\mathbb{B}$) balls. Below you are given a program that will select a combination of n balls and print their colors. Each selection sequence will be printed as a sequence of characters, e.g. "`RGB`", followed by a space. The program encodes certain rules about which balls can be selected depending on the color of the previous ball.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void red(int, int, char *);
5  void green(int, int, char *);
6  void blue(int, int, char *);
7
8  void printBuffer(char *buffer, int limit)
9  {
10     int i;
11     for (i = 0; i <= limit; i++) printf("%c", buffer[i]);
12     printf(" ");
13 }
14
15 void red(int num, int pos, char *buffer)
16 {
17     buffer[pos] = 'R';
18     if (num == 1) printBuffer(buffer, pos);
19     else {
20         blue(num - 1, pos + 1, buffer);
21     }
22 }
23
24 void green(int num, int pos, char *buffer)
25 {
26     buffer[pos] = 'G';
27     if (num == 1) printBuffer(buffer, pos);
28     else {
29         green(num - 1, pos + 1, buffer);
30         blue(num - 1, pos + 1, buffer);
31     }
32 }
33
34 void blue(int num, int pos, char *buffer)
35 {
36     buffer[pos] = 'B';
```

```
37      if (num == 1) printBuffer(buffer, pos);
38      else {
39          red(num - 1, pos + 1, buffer);
40          green(num - 1, pos + 1, buffer);
41          blue(num - 1, pos + 1, buffer);
42      }
43  }
44
45  void genColor(int num)
46  {
47      char *buffer = malloc(sizeof(char) * num);
48      red(num, 0, buffer);
49      green(num, 0, buffer);
50      blue(num, 0, buffer);
51      free(buffer);
52  }
53
54  int main(int argc, char **argv)
55  {
56      genColor(3);
57      return EXIT_SUCCESS;
58  }
```

Given the above program, please write the resulting output for the call `genColors(3)` in the main function on the provided lines below (14 in total, 0.5 point per ball sequence, 1 extra point if all sequences in the correct order). The order of the outputs is important; if your output is in the incorrect order even if the ball sequences are correct, you will lose 1 point.

- - - - - - - - - - - - - - - - - - - - -

- - - - - - - - - - - - - - - - - - - - -

2 Structure: Swaps (6 points)

The source code below represents a small string library that uses character data and a length in a structure instead of the normal null-terminating scheme that C normally uses. Fill in the blanks in the sample output below. Please note that we are using "dummy" addresses (e.g. `0x440`) to represent pointer values; real pointer addresses are 32 or 64 bits long. (6 points, 1 per line; no partial credit for a line)

```
 1 #include <stdio.h>
 2 #include <string.h>
 3 #include <stdlib.h>
 4
 5 typedef struct {
 6     char *str;
 7     int len;
 8 } String;
 9
10 String *String_create(const char *str)
11 {
12     String *s = malloc(sizeof(String));
13     s->len = strlen(str);
14     s->str = malloc(sizeof(char) * s->len);
15     memcpy(s->str, str, s->len);
16     return s;
17 }
18
19 void String_destroy(String *s)
20 {
21     free(s->str);
22     free(s);
23 }
24
25 void String_println(String *s)
26 {
27     int i;
28     for (i = 0; i < s->len; i++) {
29         printf("%c", s->str[i]);
30     }
31     printf(" (s: %p, str: %p, len: %d)\n", s, s->str, s->len);
32 }
33
34 void String_swap1(String a, String b)
35 {
```

```
36      char *ts = a.str;
37      int tl = a.len;
38      a.str = b.str;
39      a.len = b.len;
40      b.str = ts;
41      b.len = tl;
42  }
43
44  void String_swap2(String *a, String *b)
45  {
46      char *ts = a->str;
47      int tl = a->len;
48      a->str = b->str;
49      a->len = b->len;
50      b->str = ts;
51      b->len = tl;
52  }
53
54  void String_swap3(String **a, String **b)
55  {
56      String *tmp = *a;
57      *a = *b;
58      *b = tmp;
59  }
60
61  int main(int argc, char **argv)
62  {
63      String *s1 = String_create("leia");
64      String *s2 = String_create("luke");
65      String *s3 = String_create("mario");
66      String *s4 = String_create("luigi");
67      String *s5 = String_create("gandalf");
68      String *s6 = String_create("saruman");
69
70      printf(" -- Swap 1:\n");
71      String_println(s1);
72      String_println(s2);
73      printf(" -vs.-\n");
74      String_swap1(*s1, *s2);
75      String_println(s1);
76      String_println(s2);
77
```

```
78      printf("\n");
79
80      printf(" -- Swap 2:\n");
81      String_println(s3);
82      String_println(s4);
83      printf(" -vs.-\n");
84      String_swap2(s3, s4);
85      String_println(s3);
86      String_println(s4);
87
88      printf("\n");
89
90      printf(" -- Swap 3:\n");
91      String_println(s5);
92      String_println(s6);
93      printf(" -vs.-\n");
94      String_swap3(&s5, &s6);
95      String_println(s5);
96      String_println(s6);
97
98      String_destroy(s1);
99      String_destroy(s2);
100     String_destroy(s3);
101     String_destroy(s4);
102     String_destroy(s5);
103     String_destroy(s6);
104
105     return EXIT_SUCCESS;
106 }
```

Sample output with blanks on the next page.

```
 -- Swap 1:
leia (s: 0x440, str: 0x460, len: 4)
luke (s: 0x480, str: 0x4a0, len: 4)
 -vs.-

_______ (s: 0x______, str: 0x______, len: 4)

_______ (s: 0x______, str: 0x______, len: 4)

 -- Swap 2:
mario (s: 0x4c0, str: 0x4e0, len: 5)
luigi (s: 0x500, str: 0x520, len: 5)
 -vs.-

_________ (s: 0x______, str: 0x______, len: 5)

_________ (s: 0x______, str: 0x______, len: 5)

 -- Swap 3:
gandalf (s: 0x540, str: 0x560, len: 7)
saruman (s: 0x580, str: 0x5a0, len: 7)
 -vs.-

______________ (s: 0x______, str: 0x______, len: 7)

______________ (s: 0x______, str: 0x______, len: 7)
```

3 Dynamic Structure: Expression Trees (6 points)

An important part of both compilers and interpreters for a programming language is to parse and evaluate arithmetic expressions. Many times, the parsed source code is turned into a tree to enable evaluation and/or machine code generation. Below we provide some very basic source code for representing and evaluating arithmetic operations.

(a) Please fill in the resulting values for this evaluation below. (5 points, 1 per line)

(b) The tree traversal used here visits both of the children before visiting the node itself to calculate the value of the operation. What is this type of traversal called? (1 point)

```
eval(expr1) = ____

eval(expr2) = ____

eval(expr3) = ____

eval(expr4) = ____

eval(expr5) = ____
```

Being unable to calculate these numbers without a calculator is **not** a valid excuse. These are simple calculations. If you do not provide a single number, you will receive zero points for that line.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 #define ADD 0
 5 #define SUB 1
 6 #define MUL 2
 7 #define DIV 3
 8
 9 typedef struct Expr_t {
10     int val;
11     struct Expr_t *left;
12     struct Expr_t *right;
13 } Expr;
14
15 Expr *Expr_num(int val)
16 {
17     Expr *expr = malloc(sizeof(Expr));
18     expr->val = val;
```

```
19      expr->left = NULL;
20      expr->right = NULL;
21      return expr;
22  }
23
24  Expr *Expr_op(int op, Expr *left, Expr *right)
25  {
26      Expr *expr = malloc(sizeof(Expr));
27      expr->val = op;
28      expr->left = left;
29      expr->right = right;
30      return expr;
31  }
32
33  void Expr_destroy(Expr *expr)
34  {
35      if (expr == NULL) return;
36      Expr_destroy(expr->left);
37      Expr_destroy(expr->right);
38      free(expr);
39  }
40
41  Expr *Expr_add(Expr *left, Expr *right)
42  {
43      return Expr_op(ADD, left, right);
44  }
45
46  Expr *Expr_sub(Expr *left, Expr *right)
47  {
48      return Expr_op(SUB, left, right);
49  }
50
51  Expr *Expr_mul(Expr *left, Expr *right)
52  {
53      return Expr_op(MUL, left, right);
54  }
55
56  Expr *Expr_div(Expr *left, Expr *right)
57  {
58      return Expr_op(DIV, left, right);
59  }
60
```

```
61 int Expr_eval(Expr *expr)
62 {
63     if (expr->left == NULL || expr->right == NULL) return expr->val;
64     int v1 = Expr_eval(expr->left);
65     int v2 = Expr_eval(expr->right);
66     switch (expr->val) {
67     case ADD: return v1 + v2;
68     case SUB: return v1 - v2;
69     case MUL: return v1 * v2;
70     case DIV: return v1 / v2;
71     default: return 0;
72     }
73 }
74
75 int main(int argc, char **argv)
76 {
77     Expr *expr1 = Expr_add(Expr_num(1), Expr_num(5));
78     Expr *expr2 = Expr_mul(Expr_num(10), Expr_num(4));
79     Expr *expr3 = Expr_div(Expr_num(100), Expr_sub(Expr_num(30),
80                                                    Expr_num(5)));
81     Expr *expr4 = Expr_mul(Expr_add(Expr_num(4),
82                                     Expr_mul(Expr_num(10),
83                                              Expr_num(2))),
84                            Expr_div(Expr_num(40),
85                                     Expr_sub(Expr_num(6), Expr_num(2))));
86     Expr *expr5 = Expr_mul(Expr_mul(Expr_mul(Expr_num(2), Expr_num(5)),
87                                     Expr_num(2)), Expr_num(3));
88
89     printf("eval(expr1) = %d\n", Expr_eval(expr1));
90     printf("eval(expr2) = %d\n", Expr_eval(expr2));
91     printf("eval(expr3) = %d\n", Expr_eval(expr3));
92     printf("eval(expr4) = %d\n", Expr_eval(expr4));
93     printf("eval(expr5) = %d\n", Expr_eval(expr5));
94
95     Expr_destroy(expr1);
96     Expr_destroy(expr2);
97     Expr_destroy(expr3);
98     Expr_destroy(expr4);
99     Expr_destroy(expr5);
100
101    return EXIT_SUCCESS;
102 }
```