ECE264 Summer 2013
Exam 1, June 20, 2013

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.

One learning objective (recursion) is 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

Total Score:

1 Recursive Formula (5 points)

For a given positive integer, we want to partition it into the sum of some positive integers, or itself. For example, 1 to 4 can be partitioned as

```
1 = 1        2 = 1 + 1      3 = 1 + 1 + 1      4 = 1 + 1 + 1 + 1
               = 2            = 1 + 2            = 1 + 1 + 2
                              = 2 + 1            = 1 + 2 + 1
                              = 3                = 1 + 3
                                                 = 2 + 1 + 1
                                                 = 2 + 2
                                                 = 3 + 1
                                                 = 4
```

- One way to partition 1.
- Two ways to partition 2.
- Four ways to partition 3.
- Eight ways to partition 4.

In general, there are 2^{n-1} ways to partition value n. You do not have to prove this.

In the above examples,

- when partitioning 1, "2" is never used
- when partitioning 2, "2" is used once.
- when partitioning 3, "2" is used in two different lines.
- when partitioning 4, "2" is used in four different lines.

Question: When partitioning value n ($n > 2$), how many lines contain "2"?

Let $f(n)$ be the number of lines containing "2" used for partitioning n. $f(1) = 0$ and $f(2) = 1$. Write down the general rule expressing $f(n)$. You may use $f(n-1)$, $f(n-2)$ ..... and other terms related to n, such as 2^n or n. You need to write down a recursive form; you do not need to write the closed form.

You **must** explain your answer. An answer without explanation will receive no point.

This question asks you to write a mathematical formula, not a C program.

2 Partition Using Odd Numbers (5 points)

Consider the following program.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 #define MAXLENGTH 8
 5
 6 static void printPartition(int * arr, int len)
 7 {
 8   int ind;
 9   printf("= ");
10   for (ind = 0; ind < len - 1; ind ++)
11     {
12       printf("%d + ", arr[ind]);
13     }
14   printf("%d\n", arr[len - 1]);
15 }
16
17 static void partitionHelper(int value, int * arr, int ind)
18 {
19   if (value == 0)
20     {
21       printPartition(arr, ind);
22       return;
23     }
24   int nextVal;
25   for (nextVal = 1; nextVal <= value; nextVal ++)
26     {
27       arr[ind] = nextVal;
28       partitionHelper(value - nextVal, arr, ind + 1);
29     }
30 }
31
32 void partition(int value)
33 {
34   printf("partition %d\n", value);
35   int arr[MAXLENGTH];
36   partitionHelper(value, arr, 0);
37 }
38
39 int main(int argc, char * * argv)
```

```
40 {
41   int val;
42   for (val = 1; val <= MAXLENGTH; val ++)
43     {
44       partition(val);
45     }
46   return EXIT_SUCCESS;
47 }
```

Modify the program so that it generates the partitions that use odd numbers (1, 3, 5, ...) only. The program must *generate* only the valid partitions. For example, when partitioning 4, the following are valid:

```
1 + 1 + 1 + 1
1 + 3
3 + 1
```

The following are invalid because at least one even number is used in each line:

```
1 + 1 + 2
1 + 2 + 1
2 + 1 + 1
2 + 2
4
```

You will not receive points if the program generates partitions with even numbers, even though the program does not print these invalid partitions (by checking before printing).

3 Trace Recursive Function (5 points)

Consider the following program.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int f(int n, int * sum)
4  {
5    (* sum) ++; /* how many times f is called? */
6    if ((n == 0) || (n == 1))
7      {
8        return 1;
9      }
10   int i;
11   int val = 0;
12   for (i = 0; i < n; i ++)
13     {
14       int a = f(i, sum);
15       int b = f(n - i - 1, sum);
16       val += a * b;
17     }
18   return val;
19 }
20
21 int main(int argc, char * * argv)
22 {
23   int sum = 0;
24   int val = 3;
25   int fv = f(val, & sum);
26   printf("f(%d) = %d, sum = %d\n", val, fv, sum);
27   return EXIT_SUCCESS;
28 }
```

What are the values of `fv` and `sum` printed by this program? Explain how you get the answers.

You need to write **two** answers.

You can write an arithmetic expression without calculating the result. For example, if the answer is 17, you can write $1 + 5 + 11$.

4 Change Recursive Function (5 points)

The following is a **correct** program to permute a set of characters.
If the second call of the `swap` function is removed (marked by the comment), what will the program print?
Explain your answer.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #define MAX_LENGTH 4
 4
 5 static void printPermutation(char * charset, int len)
 6 {
 7   int ind;
 8   for (ind = 0; ind < len; ind ++)
 9     {
10       printf("%c ", charset[ind]);
11     }
12   printf("\n");
13 }
14
15 static void swap(char * a, char * b)
16 {
17   char t = * a;
18   * a = * b;
19   * b = t;
20 }
21
22 static void permuteHelper(char * charset, int len, int ind)
23 {
24   if (ind == len)
25     {
26       printPermutation(charset, len);
27     }
28   int pos;
29   for (pos = ind; pos < len; pos ++)
30     {
31       swap(& charset[pos], & charset[ind]); // first call of swap
32       permuteHelper(charset, len, ind + 1);
33       // remove the following line (second call of swap)
34       swap(& charset[pos], & charset[ind]); // <--- remove this line
35       // remove the previous line
36     }
```

```
37 }
38
39 void permute(char * charset, int len)
40 {
41   permuteHelper(charset, len, 0);
42 }
43
44 int main(int argc, char * * argv)
45 {
46   char set[MAX_LENGTH] = {'A', 'B', 'C', 'D'};
47   permute(set, MAX_LENGTH);
48   return EXIT_SUCCESS;
49 }
```

A correct permutation program prints $n!$ lines for a set of n elements and each line is unique.

4.1 Number of Lines

How many lines does this program print after removing the second call of `swap`?
If the program does not print $n!$ lines, how many lines does this program print?

4.2 Unique Lines

Does this program produce unique lines after removing the second call of `swap`?
If the lines are not unique, how many unique lines are printed?