ECE264 Spring 2013
Exam 1, February 14, 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 also declare that I will not discuss/share this exam with anybody on February 14, 2013. 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.

You must return all pages before you leave the room. Otherwise, your exam will not be graded and you will receive a zero score.

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.

Two learning objectives (recursion and file) are tested in this exam. To pass an objective, you must receive 50% or more points in the corresponding question.

Questions (Total 15 points)

1. recursion (6 points)
2. file (6 points)
3. pointer (3 points)

Contents

1 Recursion (6 points) **3**

1.1 Recursive Relation (1 points) 3

1.2 Trace Recursive Function (2 points) 4

1.3 Permutation (3 points) 5

2 File (6 Points) **8**

3 Pointers (3 points) **11**

Learning Objective 1 (Recursion)	Pass	Fail
Learning Objective 2 (File)	Pass	Fail
Total		

1 Recursion (6 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
```

We observe that there exists

- 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.

Also by observation, we find

- One "1" is used to partition 1.
- Two "1" is used to partition 2.
- Five "1" are used to partition 3.
- Twelve "1" are used to partition 4.

1.1 Recursive Relation (1 points)

How many "1" are used for partitioning value n? Let $f(n)$ be the number of "1" used for partitioning n. $f(1) = 1$ and $f(2) = 2$. Write down the general rule. You can write down a recursive form. You do not need to write the closed form.

1.2 Trace Recursive Function (2 points)

Consider the following program.

```c
#include <stdio.h>
#include <stdlib.h>
int f(int n, int * max, int * sum, int depth)
{
  int fnm1; /* f(n - 1) */
  int fnm2; /* f(n - 2) */
  int fsum; /* f(n - 1) + f(n - 2) */
  (* sum) ++; /* how many times f is called? */
  if ((*max) < depth)
    {
      * max = depth;
    }
  if ((n == 0) || (n == 1))
    {
      return 1;
    }
  fnm1 = f(n - 1, max, sum, depth + 1);
  fnm2 = f(n - 2, max, sum, depth + 1);
  fsum = fnm1 + fnm2;
  return fsum;
}

int main(int argc, char * * argv)
{
  int max = 0;
  int sum = 0;
  int val = 6;
  int fv = f(val, & max, & sum, 1);
  printf("f(%d) = %d, max = %d, sum = %d\n", val, fv, max, sum);
  return EXIT_SUCCESS;
}
```

For your reference, the values of `f(n)` is

n	0	1	2	3	4	5	6	7	8	9	10
f(n)	1	1	2	3	5	8	13	21	34	55	89
sum	1	1	3								

What are the values of `max` and `sum`? (0.5 point each) You can write an arithmetic expression without calculating the result. For example, if the answer is 17, you can write $1 + 5 + 11$.

Explain your answers. (1 point)

1.3 Permutation (3 points)

In PA03, you wrote a program to permute a set of characters.
Suppose the input array contains both digits (1, 2, 3, 4) and alphabets (A, B, C, D):

```
#define MAX_LENGTH 8
...
char set[MAX_LENGTH] = {'1', 'A', '2', 'B', '3', 'C', '4', 'D'};
```

We want to permute the elements. There are two restrictions: (1) The first element (index is 0) must be a digit. (2) Two digits must be separated by one and only one alphabet. Thus, the following are valid permutations:

```
1 C 2 D 4 A 3 B
4 B 1 C 3 A 2 D
3 D 2 B 1 A 4 C
```

The following are invalid permutations

```
1 A 2 B C D 4 3
B A C D 1 3 4 2
A B 4 D 1 C 2 3
```

Make necessary changes to the program below so that only valid permutations are printed.

Hint: Generating only valid permutations is easier (fewer lines of code) than generating all possible permutations and then eliminating the invalid ones.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_LENGTH 8
void printPermute(char * charset, int length);

void swap(char * c1, char * c2)
{
  char t = * c1;
  * c1 = * c2;
  * c2 = t;
}

void recursivePermute(char * charset, int ind, int length)
{
  int iter;
  if (ind == length)
    {
      printPermute(charset, length);
    }
  for (iter = ind; iter < length; iter ++)
    {
      swap(& charset[iter], & charset[ind]);
      recursivePermute(charset, ind + 1, length);
      swap(& charset[iter], & charset[ind]);
    }
}

void permute(char * charset, int length)
{
  recursivePermute(charset, 0, length);
}

int main(int argc, char * * argv)
{
  char set[MAX_LENGTH] = {'1', 'A', '2', 'B', '3', 'C', '4', 'D'};
  permute(set, 8);
  return EXIT_SUCCESS;
}

void printPermute(char * charset, int length)
{
  int iter;
  for (iter = 0; iter < length; iter ++)
    {
      printf("%c ", charset[iter]);
    }
  printf("\n");
}
```

2 File (6 Points)

The following is an *attempt* for the `countString` function in the fourth programming assignment. This function, unfortunately, does not work. In fact, it has many problems. You need to identify the problems and describe the solutions. The problems are

1. The `countString` always returns `EXIT_FAILURE` regardless of the arguments `file1` and `file2`.
2. The program never stops.
3. The output file is empty.
4. The program's output "`appears ?  times`" sometimes is a garbage value.
5. The program always reports `appears 0 times` even though the input file (whose name is given by the argument `file1`) contains the string `str`.
6. The program never stops if the input file contains the string.

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#define LINE_LENGTH 80
/* length of a line. Each line has at most 80 characters, including \n
   and \0 */

/*
 * The function counts the occurrences of a string in the input file.
 * The string is case-sensitive. "ECE" and "ece" are different.
 *
 * file1: name of the input file
 * file2: name of the output file
 * str: the string to search
 *
 * This function returns EXIT_SUCCESS if it finishes without
 * problem. The function returns EXIT_FAILURE if there is a problem
 * in opening the file, reading from the input file, or writing
 * to the output file.
 *
 * It is possible that the string may appear multiple times in the
 * same line. If this occurs, the function should count all
 * occurrences.
 *
 * For simplicity, we assume that each line contains at most 80
 * characters.  Use LINE_LENGTH and do not use 80.
 *
 */
```

```


int countString(char * file1, char * file2, char * str)
{
  FILE * fptrin  = fopen("file1", "r");
  if (fptrin == NULL)
    {
      return EXIT_FAILURE;
    }
  FILE * fptrout = fopen("file2", "w");
  if (fptrout == NULL)
    {
      fclose (fptrin);
      return EXIT_FAILURE;
    }
  char oneLine[LINE_LENGTH];
  int counter;
  char * cptr;
  fclose (fptrin);
  fclose (fptrout);
  while (fgets(oneLine, LINE_LENGTH, fptrin) == NULL)
    {
      cptr = oneLine;
      while (cptr != NULL)
        {
          cptr  = strstr(cptr, str);
          if (cptr != NULL)
            {
              counter ++;
            }
        }
    }
  fprintf(fptrout, "%s appears %d times.\n", str, counter);
  return EXIT_SUCCESS;
}

int main(int argc, char** argv)
{
  /*
     argv[1]: input file name
     argv[2]: output file name
     argv[3]: string to search
    */

  return countString(argv[1], argv[2], argv[3]);
}
```

3 Pointers (3 points)

For the following program,

```
#include <stdio.h>
#include <stdlib.h>
void swap1(int a, int b)
{
  int t = a;
  a = b;
  b = t;
}

void swap2(int * a, int * b)
{
  int * t = a;
  a = b;
  b = t;
}

void swap3(int * a, int * b)
{
  int t = * a;
  a = b;
  * b = t;
}

void swap4(int * a, int * b)
{
  int t = * a;
  * a = * b;
  * b = * t;
}

void swap5(int * a, int * b)
{
  int t = a;
  a = b;
  b = t;
}

void swap6(int * a, int * b)
{
  int t = * a;
  a = b;
  b = t;
}

void swap7(int a, int b)
{
  int * t = a;
  * a = b;
  b = t;
}

int main(int argc, char * * argv)
{
  int u;
  int t;
  u = 264;
  t = 2013;
  swap1(u, t);
  printf("swap1: u = %d, t = %d\n", u, t);

  u = 264;
  t = 2013;
  swap2(& u, & t);
  printf("swap2: u = %d, t = %d\n", u, t);

  u = 264;
  t = 2013;
  swap3(& u, & t);
  printf("swap3: u = %d, t = %d\n", u, t);

  u = 264;
  t = 2013;
  swap4(& u, & t);
  printf("swap4: u = %d, t = %d\n", u, t);

  u = 264;
  t = 2013;
  swap5(& u, & t);
  printf("swap5: u = %d, t = %d\n", u, t);

  u = 264;
```

```
82   t = 2013;
83   swap6(u, t);
84   printf("swap6: u = %d, t = %d\n", u, t);
85
86   u = 264;
87   t = 2013;
88   swap7(u, t);
89   printf("swap7: u = %d, t = %d\n", u, t);
90   return EXIT_SUCCESS;
91 }
```

Functions `swap1` - `swap3` have no warning or error from `gcc`. What are the outputs after running these three functions? Remember to write the values of **both** `u` and `t` (0.5 points for each pair of answers)

```
swap1: u =     , t =
swap2: u =     , t =
swap3: u =     , t =
```

Functions `swap4` - `swap7` have warnings or errors from `gcc`. The warnings or errors may be inside the functions or when the functions are called in `main`. Write down the line numbers and the reasons. You can mark on the lines and explain why these lines cause warnings or errors. There are more than three problems but you need to write only three. (0.5 points for each answer that includes both the line number and the reason)