CS1010E Programming Methodology
Semester 1 2016/2017

Week of 10 October – 14 October 2016
Tutorial 7 Suggested Answers
Recursion

1. Given a real number x and a non-negative integer k, we would like to compute the exponent x^k.

   (a) Write a recursive function `exponent1` using the following recurrence relation.

   $$x^k = \begin{cases} 1 & \text{if } k = 0 \\ x \cdot x^{k-1} & \text{if } k > 0 \end{cases}$$

   What is the total number of floating-point multiplications performed by your function to compute x^{13} and x^{128}?

   ANSWER:
   x^{13} requires 13 floating-point multiplications.
   x^{128} requires 128 floating-point multiplications.

   (b) We can reformulate the recursive definition of x^k so that fewer multiplications are required; hence, the efficiency of the function is improved.

   $$x^k = \begin{cases} 1 & \text{if } k = 0 \\ x^{k/2} \cdot x^{k/2} & \text{if } k \text{ is even, } k > 0 \\ x^{\lfloor k/2 \rfloor} \cdot x^{\lfloor k/2 \rfloor} \cdot x & \text{if } k \text{ is odd, } k > 0 \end{cases}$$

   where $\lfloor \cdot \rfloor$ is the floor operator. Write a recursive function `exponent2` that implements the above recursive definition of x^k. Think about how you can save some redundant or repeated recursive calls to improve efficiency. There is no need to use any mathematical functions in the Standard C Library. What is the total number of floating-point multiplications performed by your function to compute x^{13} and x^{128}?

   ANSWER:
   x^{13} requires 7 floating-point multiplications.
   x^{128} requires 9 floating-point multiplications.

```c
#include <stdio.h>

double exponent1(double x, int k);
double exponent2(double x, int k);

int main(void) {
   double x;
   int k;

   printf("Enter x and k: ");
   scanf("%lf %d", &x, &k);

   printf("The result from exponent1 is %f.\n", exponent1(x,k));
   printf("The result from exponent2 is %f.\n", exponent2(x,k));
   return 0;
}

double exponent1(double x, int k) {
   if (k <= 0) {
      return 1.0;
   } else {
      return x * exponent1(x, k-1);
   }
}

double exponent2(double x, int k) {
   double temp;

   if (k <= 0) {
      return 1.0;
   } else {
      temp = exponent2(x, k/2);
      if (k%2 == 0)
         return temp * temp;
      else
         return temp * temp * x;

   }
}
```

2. (a) Define a recurrent relation for $mul(a, b)$ to multiply a by b. Assume that b is a non-negative number. Write a recursive function `mul` to implement the recurrent relation. You are not allowed to use the multiplication `*` operator.

```
int mul(int a, int b);
```

$$mul(a,b) = \begin{cases} 0 & \text{if } b = 0 \\ a + mul(a, b-1) & \text{if } b > 0 \end{cases}$$

(b) Define a recurrent relation for $mod(a, b)$ to find the remainder of a divided by b. Assume that a and b are positive numbers. Write a recursive function `mod` to implement the recurrent relation. You are not allowed to use the modulus `%` operator.

```
int mod(int a, int b);
```

$$mod(a,b) = \begin{cases} a & \text{if } a < b \\ mod(a-b, b) & \text{otherwise} \end{cases}$$

```
#include <stdio.h>

int mul(int x, int y);
int mod(int a, int b);

int main(void) {
   int a, b;
   printf("Enter a and b: ");
   scanf("%d %d", &a, &b);
   printf("mul(%d,%d) = %d\n", a, b, mul(a,b));
   printf("mod(%d,%d) = %d\n", a, b, mod(a,b));
   return 0;
}

int mul(int a, int b) {
   return (b == 0) ? 0 : a + mul(a,b-1);
}

int mod(int a, int b) {
   return (a < b) ? a : mod(a-b,b);
}
```

3. A recursive function can be defined to break up the individual digits of a given integer argument.

   (a) Write a recursive function `printReverse` such that given a positive integer number as argument, the function will print all digits from right to left on separate lines.

   ```
   void printReverse(int n);
   ```

   Write a suitable `main` function such that the input 12345 will result in the output

   ```
   5
   4
   3
   2
   1
   ```

   (b) Write a recursive function `print` such that given a positive integer number as argument, the function will print all digits from left to right on separate lines.

   ```
   void print(int n);
   ```

   Write a suitable `main` function such that the input 12345 will result in the output

   ```
   1
   2
   3
   4
   5
   ```

   (c) Write a recursive function `printEvenOdd` such that given a positive integer number as argument, the function will first print all even digits from right to left, followed by all odd digits from left to right.

   ```
   void printEvenOdd(int n);
   ```

   Write a suitable `main` function such that the input 12345 will result in the output

   ```
   4
   2
   1
   3
   5
   ```

   (d) Write a recursive function `sumDigits` that takes in a non-negative integer as argument and returns the sum of the individual digits. For example, the sum of digits of 12345 is 15.

   ```
   int sumDigits(int n);
   ```

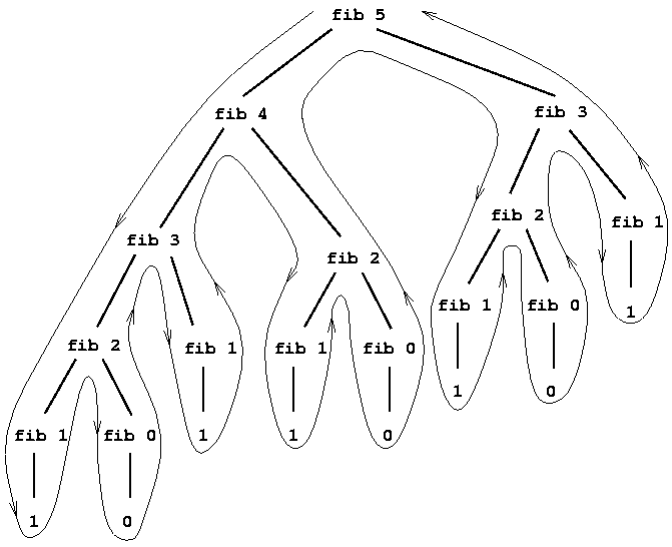
```c
#include <stdio.h>

void printReverse(int n);
void print(int n);
void printEvenOdd(int n);
int sumDigits(int n);

int main(void) {
   int n;

   printf("Enter number: ");
   scanf("%d", &n);

   printReverse(n);
   printf("---\n");
   print(n);
   printf("---\n");
   printEvenOdd(n);
   printf("---\n");
   printf("The sum of digits of %d is %d\n", n, sumDigits(n));

   return 0;
}

void printReverse(int n) {
   int digit;

   if (n > 0) {
      printf("%d\n", n%10);
      printReverse(n/10);
   }

   return;
}

void print(int n) {
   int digit;

   if (n > 0) {
      print(n/10);
      printf("%d\n", n%10);
   }

   return;
}
```

```
void printEvenOdd(int n) {
   int digit;

   if (n > 0) {
      digit = n%10;
      if (digit%2 == 0) {
         printf("%d\n", digit);
      }
      printEvenOdd(n/10);
      if (digit%2 == 1) {
         printf("%d\n", digit);
      }
   }

   return;
}

int sumDigits(int n) {
   if (n < 10) {
      return n;
   } else {
      return n%10 + sumDigits(n/10);
   }
}
```

4. The following function computes the k-th term of the Fibonacci sequence with $k \geq 0$.

```
int fibonacci(int k) {
   if (k <= 1) {
      return k;
   } else {
      return fibonacci(k-1) + fibonacci(k-2);
   }
}
```

The recursive tree for the evaluation of `fibonacci(5)` (abbreviated `fib 5`) is given below.



Rewrite the function to also return the number of times the function is invoked to compute the k-th term. We shall use an output parameter for this purpose.

```
int fibonacci(int k, int *count);
```

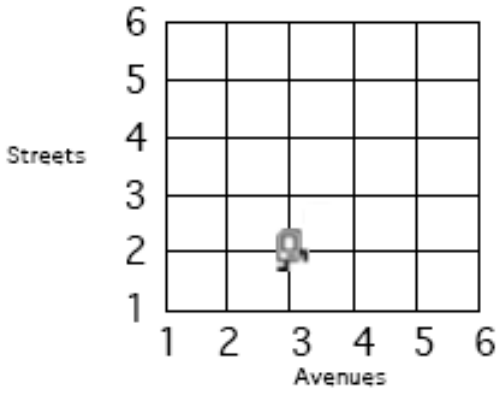
Hint: each recursive function maintains its own `count`(s).

Write a `main` function to output the k-term as well as the number of recursions involved. For example,

```
f(5) = 5
Number of recursive calls: 15
```

```c
#include <stdio.h>

int fibonacci(int k, int *count);

int main(void) {
   int n, count;

   printf("Enter n: ");
   scanf("%d", &n);
   printf("f(%d) = %d\n", n, fibonacci(n,&count));
   printf("Number of recursive calls: %d\n", count);

   return 0;
}

int fibonacci(int k, int *count) {
   int count1 = 0, count2 = 0, result;

   if (k > 1) {
      result = fibonacci(k-1,&count1) + fibonacci(k-2,&count2);
      *count = count1 + count2 + 1;
      return result;
   } else {
      *count = 1;
      return k;
   }
}
```

5. Karel the Robot likes to wander around town that is accessible by steets and avenues laid out on a rectangular grid.



Suppose Karel is stationed at the intersection of 2nd Street and 3rd Avenue as shown. You decide to get her back home (situated at the junction of 1st Street and 1St Avenue), so you call out to her: "Oh! Karel...". Karel would like to get back home with the least number of steps. With one step, Karel can move north, south, east or west to another junction. The following are the three possible minimal movements of Karel.

```
(2,3) --> (1,3) --> (1,2) --> (1,1)
```

```
(2,3) --> (2,2) --> (1,2) --> (1,1)
```

```
(2,3) --> (2,2) --> (2,1) --> (1,1)
```

Write a recursive function `ohKarel` of the following prototype

```
int ohKarel(int street, int avenue);
```

such that the number of minimal paths that Karel can take is returned. We shall assume that Karel is never home at the start. Rather than using combinatorics, simply simulate the possible movements of Karel and count the paths that can lead her home.

```
#include <stdio.h>

int ohKarel(int street, int avenue);

int main(void) {
   int street, avenue;

   printf("Enter street and avenue: ");
   scanf("%d%d", &street, &avenue);

   printf("Number of minimal paths: %d\n", ohKarel(street,avenue));

   return 0;
}

int ohKarel(int street, int avenue) {
   if ((street == 1) || (avenue == 1)) {
      return 1;
   } else {
      return ohKarel(street-1,avenue) + ohKarel(street,avenue-1);
   }
}
```