Being ignorant is not so much a shame, as being unwilling to learn.
~ *Benjamin Franklin*

CS1010 Programming Methodology

Week 11: Recursion (Selected Answers)

---

1. **Tracing recursive codes**

   (a) [AY2010/2011 Semester 1 Exam, Q1.2]
   Given the following function, what does **f(5)** compute?

```
// Precond: n >= 0
int f(int n) {
    if (n == 0)
        return 0;
    else
        return (2 * n + f(n-1));
}
```

Answer: See **q1a.c**
30

For such question, I would recommend students to trace from bottom up. That is, find out what is f(0), then f(1), then f(2), and so on. In this way, less mistake will be made.

f(0) → 0
f(1) → 2 * 1 + f(0) → 2
f(2) → 2 * 2 + f(1) → 6
f(3) → 2 * 3 + f(2) → 12
f(4) → 2 * 4 + f(3) → 20
f(5) → 2 * 5 + f(4) → 30

(b) Trace the function below manually, and write out the return value of **q(12)**.

```
// Precond: n >= 0
int q(int n) {
   if (n < 3)
      return n+1;
   else
      return q(n-3) + q(n-1);
}
```

Exploration: Would you be able to write an iterative version? Run both versions on large input, such as 50. What do you observe?

Answer: See **q1b.c**
129
The recursive version is slower than the iterative one – same reason as for Fibonacci.

(c) [AY2011/2012 Semester 1 Exam, Q1.5]
What does following function compute?

```
int mystery(int x, int y) {
   if (x == 0)
      return y;
   else if (x < 0)
      return mystery(++x, --y);
   else
      return mystery(--x, ++y);
}
```

A. It returns the value of y.
B. It returns the value of x – y.
C. It returns the value of x + y.
D. It returns the value of x * y.
E. It will give compile-time error.

Answer: See **q1c.c**
(C) It returns the value of x + y.

6. **Reversing an Array**
Write a program **reverse_array.c** to randomly assign values into an integer array, and then reverse the array using recursion.

For example, if the array contains { 6, 3, 0, 6, 8, 1, 5 }, then the reversed array is { 5, 1, 8, 6, 0, 3, 6 }. You should not use any additional array. You may assume that the array contains at most 15 elements.

Answer: See **reverse_array.c**

7. [AY2012/2013 Semester 1 Exam, Q4]
A positive integer can always be expressed as a product of prime numbers. For instance,

$$60 = 2 \times 2 \times 3 \times 5 = 2^2 \times 3^1 \times 5^1$$
$$78 = 2 \times 3 \times 13 = 2^1 \times 3^1 \times 13^1$$
$$13 = 13 = 13^1$$

The following function **countPrimes()** takes a positive integer and counts the number of (possibly duplicate) prime numbers required to form the given integer.

`countPrimes(60)` returns `4` (two 2's, one 3 and one 5)
`countPrimes(78)` returns `3` (one 2, one 3 and one 13)
`countPrimes(13)` returns `1` (one 13)

```
int countPrimes(int number) {
   return countPrimesRec(number, 1, 0);
}
```

This function in turn calls a recursive function **countPrimesRec**() that does the job of counting primes. The partial code for **countPrimesRec**() is given. You are to complete it by filling your code and the pre-conditions in the dashed boxes.

Note that function **getPrime(*i*)** is considered given, which returns the *i*-th smallest prime number. For instance:

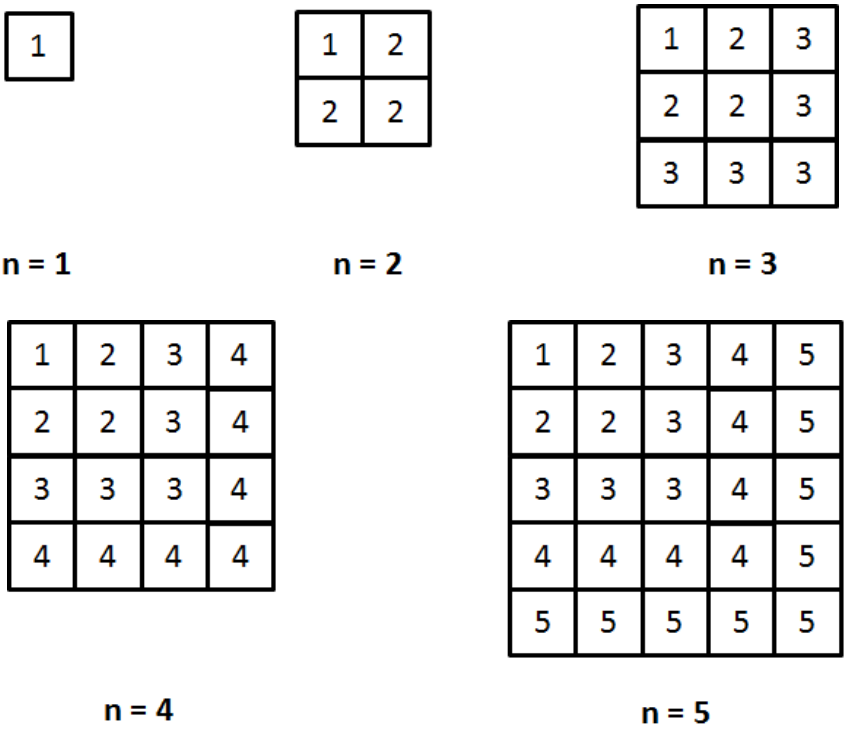
`getPrime(1)` returns `2`; `getPrime(2)` returns `3`; `getPrime(3)` returns `5`

```
// Pre-conditions:
//   number > 0; index > 0; count >= 0
//
int countPrimesRec(int number, int index, int count) {
   int prime;

   if (number == 1)
      return count;
   prime = getPrime(index);
   if (number % prime) {
      return countPrimesRec(number,
                            index+1, count);
   }
   else {
      return countPrimesRec(number/prime,
                            index, count+1);
   }
}
```

(See count_primes.c)

8. [AY2015/2016 Semester 1 Exam, Q13]
Observe the pattern in the following 2D arrays.



Write a recursive function **fill(int arr[][20], int n)** to fill a **n** x **n** 2D array **arr** with the pattern shown above.

You may assume that **n** is an integer in [1, 20].

```
void fill(int arr[][20], int n){
     if (n == 1)
      arr[0][0] = 1;
     else {
      int i;
           for (i=0; i<n-1; i++)
           arr[n-1][i] = n;

      for (i=0; i<n; i++)
           arr[i][n-1] = n;

      fill(arr, n-1);
     }
}
```