Computer Science I – Summer 2011
Recitation #3: Recursion (Solutions)

1) Write a **recursive function** that calculates a^b, where a and b are positive integers.

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

int power(int a, int b) {
  if (b == 0) return 1;
  return a*power(a,b-1);

}
```

2) Write **a recursive function** that calculates the sum $1^1 + 2^2 + 3^3 + ... + n^n$, given an integer value of n in between 1 and 9. Make sure to call your function power from question one in your implementation below:

```
int crazySum(int n);

int crazySum(int n) {

  if (n == 1)
    return 1;
  else
    return power(n,n) + crazySum(n-1);
}
```

3) Given the function below, what would the function call question3(357, 8264) return?

```
int question3(int a, int b) {

  if (a == 0) return b;
  if (b == 0) return a;

  return question3(10*a+b%10, b/10);
}
```

```
question3(357, 8264) =
  question3(3574, 826) =
    question3(35746, 82) =
      question3(357462, 8) =
        question3(3574628, 0) =
          3574628
```

4) A maze can be encoded as a two dimensional array of integers by storing various numbers in each array element. A -1 represents an invalid square, while all positive integers less than 4 represent valid squares along with the possible directions you can move from those squares. Here is a chart that outlines the possible directions for each square:

Value	Possible Moves
1	Left and Right
2	Up and Down
3	Left, Right, Up and Down

A prize in a square is denoted by a 9. Your goal is to write a function that takes in a maze as input, as well (x,y) coordinates for a starting position, and returns 1 if there is a path from the starting position to a square with a prize and returns a 0 if there is no such path. The function is outlined on the next page and will work recursively. The idea to solve the problem is as follows:

1) If the given square has the prize, return 1 immediately.
2) If the given square has a -1, return 0 immediately.
3) Otherwise, check the value in the given square, based upon this value recursively start searching for a prize square from the appropriate adjacent squares. (Thus, if the value in the square is 1, start a search from the left of this square, and another search from the right of this square. Return true if either search is fruitful.) Before you recursively search however, to prevent coming "back" to the original square, change its value to -1.

Assume that the maze is a 10x10 maze and that no valid search path leads out of the maze. (Thus, no need to check for ANY array out of bounds errors.) Also, assume that the x coordinate governs left-right movement in the maze while the y coordinate governs up-down movement in the maze.
Fill in the blanks in the function below:

```
int pathToPrize(int maze[][10], int x, int y) {

  if ( maze[x][y] == 9 )
    return 1;
  else if ( maze[x][y] == -1 )
    return 0 ;

  if ( maze[x][y] == 1) {
    maze[x][y] = -1;
    return pathToPrize( maze , x-1 , y ) ||
           pathToPrize( maze , x+1 , y );
  }

  else if ( maze[x][y] == 2) {
    maze[x][y] = -1;
    return pathToPrize( maze , x , y-1 ) ||
           pathToPrize( maze , x , y+1 );
  }

  else if ( maze[x][y] == 3) {
    maze[x][y] = -1;
    return pathToPrize( maze , x , y-1 ) ||
           pathToPrize( maze , x , y+1 ) ||
           pathToPrize( maze , x-1 , y ) ||
           pathToPrize( maze , x+1 , y );
  }
}
```