Lab Exercise: Recursion

In this exercise you will gain some routine in design using recursion and the implementation of recursive functions. The problems below are relatively simple in nature, but require some thought if they are to be solved with recursion (which, of course, they are!)

For each exercise you are required to document that you are following the recursion recipe. This means that you must:

1. State the problem in terms of its size/complexity
2. Find, state and handle the base case (BC).
3. Find, state and handle the recursive case (RC) and ensure progress towards the BC
4. Ensure that the RC reaches the BC

Exercise 1: Triangle

Design, implement, and test the function `triangle()` below.

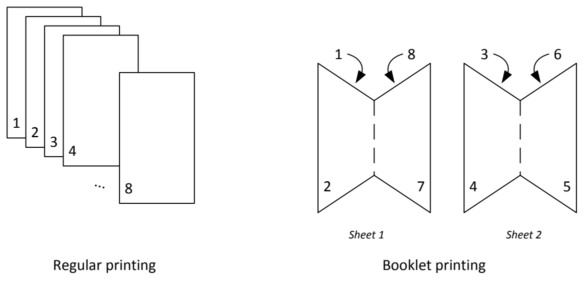
```
// Precondition: m <= n
// Postcondition: The function has printed a pattern of lines so that the
// first line contains m stars, the next line contains m+1 stars, and so on
// up to a line with n asterisks.
// Then the pattern is repeated backwards, going n back down to m.
// Example output:
// triangle(4, 6) will print this:
// ****
// *****
// ******
// ******
// *****
// ****
*/
void triangle(unsigned int m, unsigned int n)
```

Exercise 2: Searching an array

Design, implement, and test a *recursive* function that searches through the first `n` elements of an array `ar` of integers for a given element `x`. If `x` is found in `ar[0..n)` the element is found, the function should return `true`, else the function should return `false`.

Exercise 3: Booklet printing

Many printers allow *booklet printing* of large documents. When using booklet printing, 4 pages are printed on each sheet of paper, so that the output sheets can be folded into a booklet, see below



Design, implement and test a function `void bookletPrint(int startPage, int endPage)` that outputs the pages on each sheet (You may assume that the total number of pages is a multiple of four). E.g. for `bookletPrint(1,8)`:

```
Sheet 1 contains pages 1, 2, 7, 8.
Sheet 2 contains pages 3, 4, 5, 6.
```

Exercise 4: A pattern of stars

Examine this pattern of stars and blanks, and design, implement, and test a recursive function that can generate patterns such as this:

```
*
* *
  *
* * * *
    *
    * *
      *
* * * * * * * *
        *
        * *
          *
        * * * *
            *
            * *
              *
```

Your prototype should look like this:

```
  // Precondition: ‘n’ is a power of 2. ‘n’ >= 0, e.g. 1,2,4,8,16,32,....
  // Postcondition: A pattern based on the above example has been printed. The longest line of
  // the pattern has ‘n’ stars beginning in column ‘i’ of the output.

  void pattern(unsigned int n, unsigned int i);
```

For example, the above pattern is produced by the call `pattern(8, 0)`.Think about how the pattern is recursive. Hint: Can you find *two* smaller versions of the pattern within the large pattern? With recursive thinking, the function needs only seven or eight lines of code (including *two* recursive calls).

Exercise 5: A recursive Linked List Toolkit

1. Review your linked list toolkit. Which of the functions in the toolkit *may* be implemented recursively? Ask yourself: “Can I solve this problem by solving a smaller version of it?”

2. For each of the functions you found in (1) above, state the base case and the recursive case, and argue that the algorithm is correct (i.e., that it does what it is supposed to do, and that it terminates)

3. Design, implement and test your recursive solutions from (1) and (2) above.

Exercise 6: Binary search algorithm

Design, implement and test binary search of a sorted array as we have discussed in class. Also implement serial search. Run binary and serial search on large arrays (e.g of size 2000 and greater) and compare their execution times – is binary search worthwhile?

Exercise 7: Is my phone number cool?
When you send text messages you may use the mobile phone's digits for letters, i.e. characters 'd', 'e', and 'f' are on digit '3'.

1	2 abc	3 def
4 ghi	5 jkl	6mno
7 pqrs	8 tuv	9wxyz
*	0	#

Write a recursive function that can generate all strings for a given phone number. Try with your own number – does it generate any 'cool' strings? Hint: All strings for a *k*-digit number $n=n_0n_1n_2…n_k$ is the same as (all characters for n_0) combined with (all strings for the number $n'= n_1n_2…n_k$).

Exercise 8: The maze (advanced, optional)
An evil inka king has dropped you into a maze like the one below and slammed the door shut behind you.

Write an algorithm that finds the way out (hint: A "room" can be connected to up to 4 other rooms – check recursively to see if it is the exit or perhaps connects to the exit.