05OGDLP - Algorithms & Programming
Laboratory 8

Learning objectives

- Recursion

1 Multiplication

Implement the recursive function

```
int mult(int x, int y);
```

to multiply two integer numbers `x` and `y`. Notice that the elementary algorithm to perform multiplication is based on a *sum-and-shift* process. Instead, `mult` may adopt the following recursive definition:

```
x * y = (x * (y - 1)) + x
x * 1 = x
```

2 Decimal to binary conversion

Implement the recursive function

```
void d2b(int d, int * b, int * n);
```

to convert the decimal number `d` into a binary one (stored into the array `b`) having length `n` bits. Notice that to perform this conversion, it is necessary to apply the "division" algorithm as follows:

```
   :2
23 |
11 | 1
 5 | 1
 2 | 1
 1 | 0
 0 | 1
```

such that:

```
23 (base 10) == 10111 (base 2)
```

Suppose that the size of the binary array is always sufficient to store all binary digits. Each call to the recursive function must generate and store into the array one single bit of the result.

3 Catalan numbers

Implement the recursive function

```
int catalan(int n);
```

to compute and return the Catalan number of order `n`.

Catalan numbers are a sequence of numbers that can be defined directly or recursively. The recursive definition is the following one:

$$C_0 = 1 \, for \, n = 0 \, , \;\; C_n = \sum_{i=0}^{n-1} (C_i * C_{n-1-i}) \;\; for \, n >= 1 \tag{1}$$

Write a program that, using the recursive function `catalan`, prints out the first 10 Catalan numbers.

Observation

$\sum$ indicates the summation symbol, i.e., $\sum_{i=0}^{n-1}(C_i * C_{n-1-i})$ indicates the addition sequence of the numbers $(C_i * C_{n-1-i})$ for all values of i from $i = 0$ to $i = n - 1$:

$$C_0 = 1 \tag{2}$$

$$C_1 = C_0 * C_0 = 1 \tag{3}$$

$$C_2 = C_0 * C_1 + C_1 * C_0 = 2 \tag{4}$$

$$C_3 = C_0 * C_2 + C_1 * C_1 + C_2 * C_0 = 5 \tag{5}$$

$$C_4 = C_0 * C_3 + C_1 * C_2 + C_2 * C_1 + C_3 * C_0 = 14 \tag{6}$$

$$C_5 = C_0 * C_4 + C_1 * C_3 + C_2 * C_2 + C_3 * C_1 + C_4 * C_0 = 42 \tag{7}$$

$$C_6 = C_0 * C_5 + C_1 * C_4 + C_2 * C_3 + C_3 * C_2 + C_4 * C_1 + C_5 * C_0 = 132 \tag{8}$$

4 White-space count

Implement the recursive function

```
int countSpaces(char * s);
```

which counts and returns the number of white-space characters that appear in a string `s`. Notice that characters are white-space as defined by the `isspace()` function within the `<ctype.h>` library.

Observation

If string `s` is read with the command

```
scanf("%s", s);
```

it cannot include spaces, as the reading operation actually stops when it reaches a space. Use the function `gets()` instead, to read a string with spaces, or define and initialize the string as:

```
char s[] = "This is a string with spaces!";
```

5 Palindromes

Implement the recursive function

```
int isPalindrome(char * s, int l);
```

that expects a string `s` and its length `l` as arguments, and recursively determines whether the string is a palindrome.

Observation

A sequence is a palindrome when the sequence of characters or numbers looks the same forwards and backwards. For example, the strings

```
Madam, I'm Adam
A man, a plan, a canal, Panama
```

are palindrome (excluding spaces and punctuation, and considering uppercase letters equal to lowercase ones), because they are spelled the same way reading them from front to back or from back to front.

The number 12321 is a numerical palindrome.

6 Triangles

Implement the recursive function

```
void triangle(int n, ...);
```

that, given an integer value `n` and any other parameters considered necessary, prints out a triangle containing an increasing sequence of integer numbers up to `n`, arranged in a triangular shape. For instance, given `n = 4`, the program prints out

```
1
2 2
3 3 3
4 4 4 4
```

Suggestion

Use an iterative construct (along each row) plus a recursion mechanism to move on to the subsequent row.

7 Triangles

Modify the previous program so that it does not rely on any iterative construct, i.e., using recursion to "move" along both rows and columns of the triangle.