Programmering for computerteknologi
Hand-in Assignment Exercises

Week 9: Functions that call themselves

Please make sure to submit your solutions by next **Monday** (06-11-2023).

In the beginning of each question, it is described what kind of answer that you are expected to submit. If Text answer AND Code answer is stated, then you need to submit BOTH some argumentation/description and some code; if just Text answer OR Code answer then just some argumentation/description OR code. The final answer to the answers requiring text should be one pdf document with one answer for each text question (or text and code question). Make sure that you have committed your code solutions to your GitHub Repository.

Note: the **Challenge** exercises are *optional*, the others mandatory (i.e. you **have** to hand them in).

Exercises

1)

Text answer We talked about the run-time stack (see e.g. slides from lecture 7). In the lecture, we looked at the Fibonacci numbers and a program to calculate them (`fib.c`).

Draw the stack as it evolves when calculating `fib(4)`

2)

Code answer Summing an array can recursively be described as follows (a is the array, n is the length of the array):

$$\text{sum}(a, n) = \begin{cases} a[n-1] + \text{sum}(a, n-1) & \text{if } n > 0 \\ 0 & \text{if } n = 0 \end{cases}$$

Implement a recursive function with the signature

```c
int sum(const int a[], int n);
```

that sums the integer array a.

3)

Code answer In the lecture, we looked at a recursive binary search. To use binary search, the elements must be sorted. A recursive search function NOT requiring the elements to be sorted could look like (a is the array, n is the length of the array, x is the element to be found):

$$\text{search}(a, n, x) = \begin{cases} \text{true} & \text{if } n > 0 \text{ and } a[n-1] = x \\ \text{search}(a, n-1, x) & \text{if } n > 0 \text{ and } a[n-1] \neq x \\ \text{false} & \text{if } n = 0 \end{cases}$$

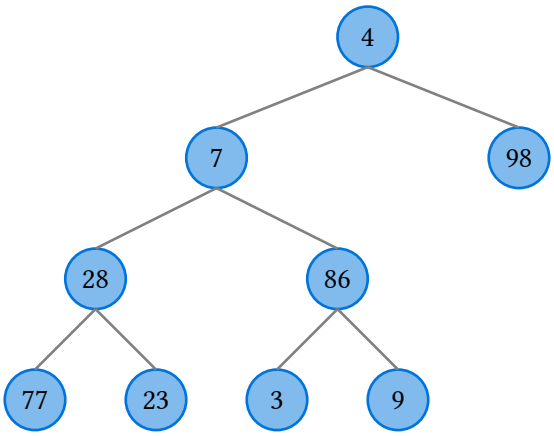
Implement a recursive function with the signature:

```
bool search(const int a[], int n, int x);
```

that searches the integer array a for the element x.

4)

Code answer Implement depth-first search using a **stack** in a fashion similar to as presented in the lectures. Your stack should be implemented as a linked list, and your tree as *tree nodes* that each have an integer as the data item and a left and right child. Given the following tree your DFS code should return the sequence of nodes visited in a linked list.



The correct output for this tree should be $4 \rightarrow 7 \rightarrow 28 \rightarrow 77 \rightarrow 23 \rightarrow 86 \rightarrow 3 \rightarrow 9 \rightarrow 98$

Challenge) (PC-6.6.4)

Let X be the set of correctly built parenthesis expressions. The elements of X are strings consisting only of the characters "(" and ")", defined as follows

- The empty string "" belong to X.
- If A belongs to X, then (A) belongs to X.
- If both A and B belong to X, then the concatenation AB belongs to X.

For example the strings `()(())()` and `(()(()))` are correctly built parenthesis expressions, and therefore belong to the set X. The expressions `(()))(()` and `())(()` are not correctly built parenthesis expressions and are thus not in X. The length of a correctly built parenthesis expression E is the number of single parenthesis (characters) in X. The depth $D(E)$ of E is defined as follows

$$D(E) = \begin{cases} 0 & \text{if } E \text{ is empty} \\ D(A) + 1 & \text{if } E = (A) \text{ and } A \text{ is in} \\ \max(D(A), D(B)) & \text{if } E = AB \text{ and } A, B \text{ are in} X \end{cases}$$

For example, `()(())()` has length 8 and depth 2. Write a program which reads in n and d and computes the number of correctly built parenthesis expressions of length n and depth d.

Input

The input consists of a pair of integers n and d with $2 \leq n \leq 300, 1 \leq d \leq 150$.

Output

For the pair of integers in the input, output a single integer on one line – the number of correctly built parenthesis expressions of length n and depth d.

Example

```
6 2
3

300 150
1
```

The three correctly built parenthesis expressions of length 6 and depth 2 are `(())()`, `()(())`, and `(()())`.