Tutorial Session 2

Tuesday, 28th of February 2020

Discussion of Exercise 1

Recursion

14.15 – 15.45

Y35-F-32

Agenda

- Administrative Provisions
- Discussion of Exercise 1
- Recap of Recursion
- Preview on Exercise 2

Administrative Provisions

- Some Remarks on Tooling

Some Remarks on Tooling

- Make sure that you have a possibility to write C code. Writing C code yourself is – with all frustrations it comes with – the only way you really will learn C.
- Try to spend most of your time with actual coding. If you're not happy with a particular IDE or setting, just take another one.

Discussion of Exercise 1

- Additional Exercise: Implementing `startswith`
- Task 1: Determining String Length by Foot
- Task 2: Palindromes
- Task 3: Counting the Number of Occurrences of Integers in an Array
- Additional Exercise: Occurrences of Integers in an Arbitrary Array
- Task 4: Sorting Even and Odd Integers Separately
- Tips and Tricks when Devising Algorithms

Additional Exercise: String Manipulation

In Python, there are many functions included natively to deal with strings. For example, there is a function `startswith` which will check whether a given string starts with a given substring. In C, the standard library does also contain some string functions but not as many as Python has. The function named above is not available for example.

Implement in C a function `bool startswith(char fullString[], char startString[])` which checks whether `fullString` starts with `startString` end returns `true` if this is the case.

(Additional exercises for at home: implement endswith, islower, …)

String Manipulation: Additional Exercise 2

Implement in C a function `void zigzag(char input[])` which takes a string as parameter and will print the letters of the string alternately as uppercase and lowercase letter (starting with uppercase and ignoring whether the letter was uppercase or lowercase initially).

Example: `zigzag("something")` should print `SoMeThInG` to the console.

You may use the functions `toupper` and `tolower` from `ctype.h` for this task.

Exercise 1 – Task 1: Determining String Length by Foot

Solution function:

```
int strLength(char s[]) {
	int i = 0;

	while (s[i] != '\0') {
		i++;
	}

	return i;
}
```

Test driver:

```
#include <stdio.h>

int main() {
	char s[1001];
	int length;

	printf("Please enter a string: ");
	scanf("%s", s);
	length = strLength(s);
	printf("String Length: %d\n", length);

	return 0;
}
```

Exercise 1 – Task 2: Palindromes

Solution function:

```
bool isPalindrome(char s[]) {
    int c, i, sLength;
    c = 0;

    sLength = strLength(s);
    for (i = 0; i < sLength/2; i++) {
        if (s[i] == s[sLength-i-1]) {
            c++;
        }
    }

    if (c == sLength/2) {
        return true;
    }

    return false;
}
```

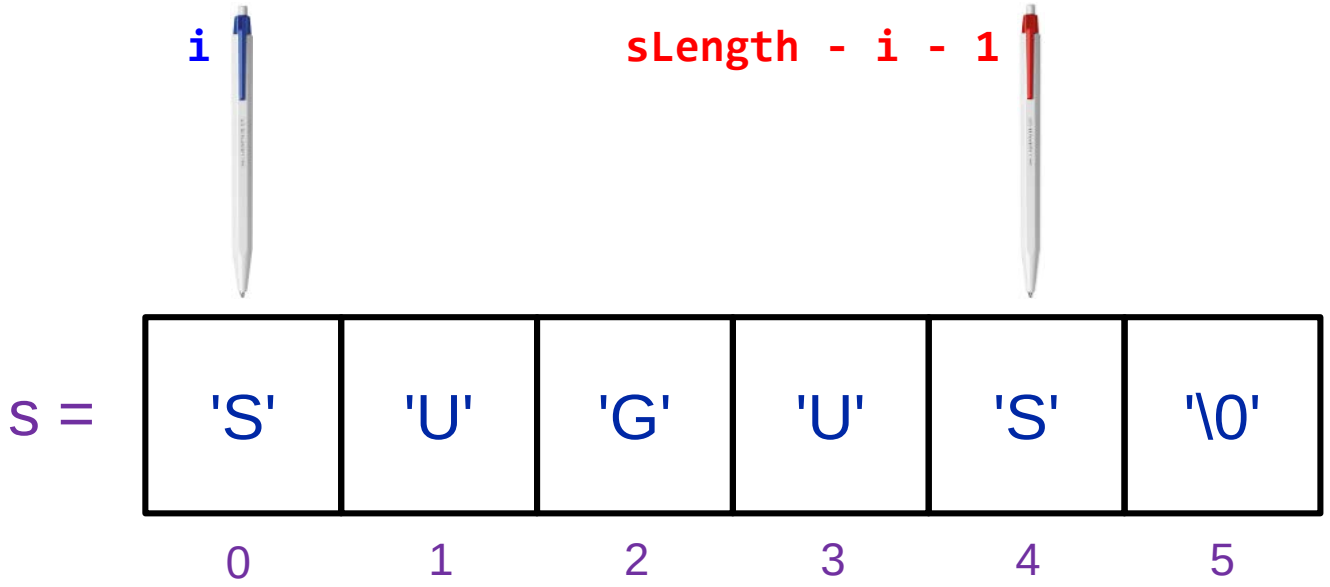
Test driver:

```
#include <stdio.h>
#include <stdbool.h>

void main() {
    bool palindromeCheck;
    char s[1001];
    printf("Please enter a string to check if it is palindrome: ");
    scanf("%[^\n]s", s);
    palindromeCheck = isPalindrome(s);
    if (palindromeCheck) {
        printf("Result string: %s\n", "TRUE");
    }
    else {
        printf("Result string: %s\n", "FALSE");
    }
}
```

Exercise 1 – Task 2: Palindromes

sLength = 5
sLength / 2 = 2



i	*sLength - i - 1*	*c*
0	4	0
1	3	1
2	2	2

Exercise 1 – Task 3: Counting the Occurrences of Integers

```
void countPairs(int A[], int nA, int B[], int nB) {
    int i,j, count;
    i = 0;
    j = 0;
    count = 0;
    printf("Pairs");
    while (i < nA) {

        while (A[i] > B[j] && j < nB) {
            j++;
        }

        while (A[i] == B[j] && j < nB) {
            j++;
            count++;
        }

        printf(" (%d, %d)", A[i], count);
        i++;
        count = 0;
    }
}
```

Additional Exercise: Occurrences of Integers in an Arbitrary Array

Solve the same exercise as task 3 of assignment 1, but this time with a single and unsorted array of integers.

Exercise 1 – Task 4: Sorting Even and Odd Integers Separately

View sample solution code…

Tips and Tricks When Devising Algorithms

- In the beginning, think about a solution on a high level and write it down as a (pseudocode) blueprint which you can later use when you're actually implementing it in C.
- Make drawings!
- Use pens or your fingers to think about iterative algorithms operating over arrays.
- Make a table to analyze loops, keeping track of iteration variables and other relevant values.
- Don't forget to think about potential special cases. Test your code thorougly.
- Execute your code regularly, do not write lots of code and only then try out. Make baby steps in the beginning!

Recursion

- Explaining Recursion
- Example Problems

Recursion

- Function / method which calls itself (directly or indirectly)
- Recursive case and base case
- Can be rewritten as iteration

- Main idea: solve a smaller problem which is an exact copy of the bigger problem but smaller

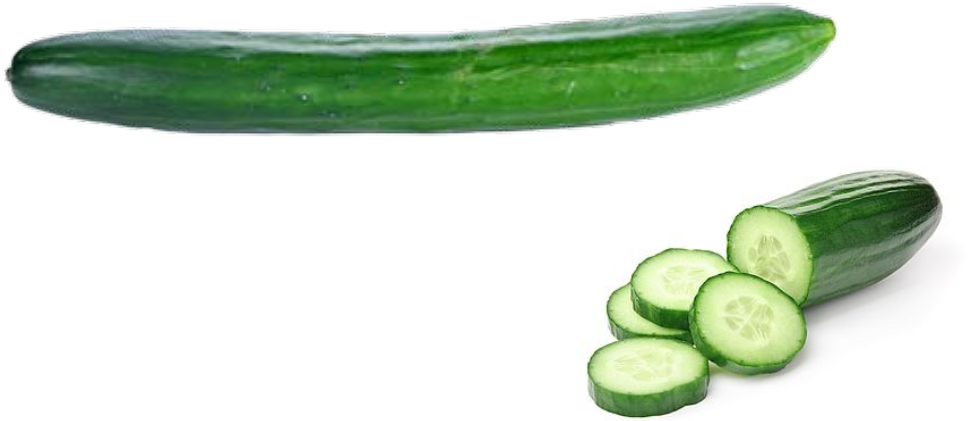
Explaining Recursion

Assume you want to explain recursion to your 10 year old cousin.

How would you do this?

Explaining Recursion: How To Make Cucumber Salad

- How do cut a cucumber to make a salad iteratively and recursively?
- How do you eat a plate of soup recursively?



Exercise Recursion: Sum of an Integer Array

Implement a function in C which recursively calculates the sum of the elements of an integer array.

Exercise Recursion: Average of an Integer Array

Calculate the average of the elements in an array of integers recursively.

Exercise Recursion: Average of an Integer Array

By definition, the average is the sum of all elements divided by the number of the elements:

$$avg(A[1..n]) = sum(A[1..n]) / n$$

this can be rewritten as follows:

$$sum(A[1..n]) = avg(A[1..n]) \cdot n$$

Therefore:

$$sum(A[1..n]) = A[1] + sum(A[2..n])$$
$$sum(A[1..n]) = A[1] + (avg(A[2..n]) \cdot (n - 1))$$
$$avg(A[1..n]) = sum(A[1..n]) / n$$

Recursive formulation: $avg(A, n) = \begin{cases} A[1] & \text{iff } n = 1 \\ (A[1] + avg(A[2..n]) \cdot (n - 1)) / n \end{cases}$

Exercise Recursion: Average of an Integer Array

```
double average(double a[], int n) {
  int idx = n-1;
  if (n == 1) {
    return a[idx];
  } else {
    return (average(a, n-1) * (n-1) + a[n-1])/n;
  }
}
```

base case

recursive call

Recursion: Code Example I

Consider the following recursive function:

```
int rec_fun(int n) {
      if (n < 10) {
            return n;
      } else {
            int a = n / 10;
            int b = n % 10;
            return rec_fun(a + b);
      }
}
```

What is the result of the call `rec_fun(648)`?

A: 8
B: 9
C: 54
D: 72
E: 648

Recursion: Code Example II

Consider the following code snippet. What will be the output?

```c
#include <stdio.h>

void whatDoesItDo(int n) {
    if (n > 0) {
        int a = 1;
        int b = 2;
        int c = a + b;
        whatDoesItDo(c);
    }
    else {
        printf("world!\n");
    }
}
```

```c
int main() {
    printf("Hello ");
    int x = 42;
    whatDoesItDo(x);
    return 0;
}
```

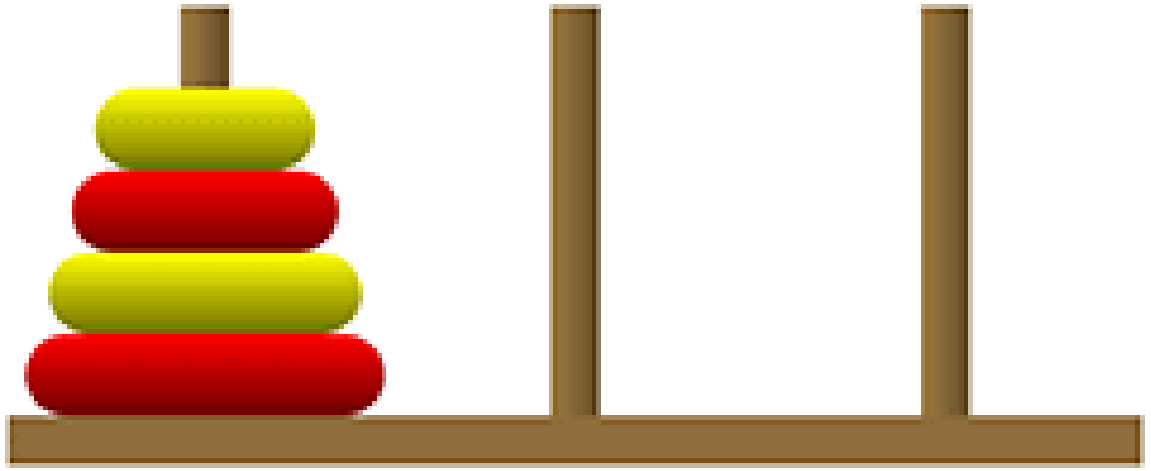
Recursion: Comprehension Question

What is the difference between an infinite loop and an infinite recursion?

Recursion: Comprehension Question

Can there be a meaningful recursive function which has return type `void` and no arguments, e.g. a function with signature `void myRecFun(void)`? Explain your answer.

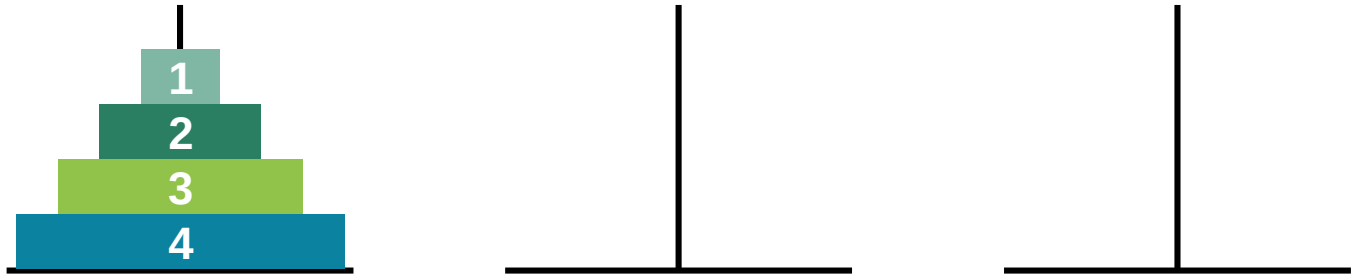
Towers of Hanoi



(from http://larc.unt.edu/ian/TowersOfHanoi/4-256.gif)

Towers of Hanoi: Example with Four Disks

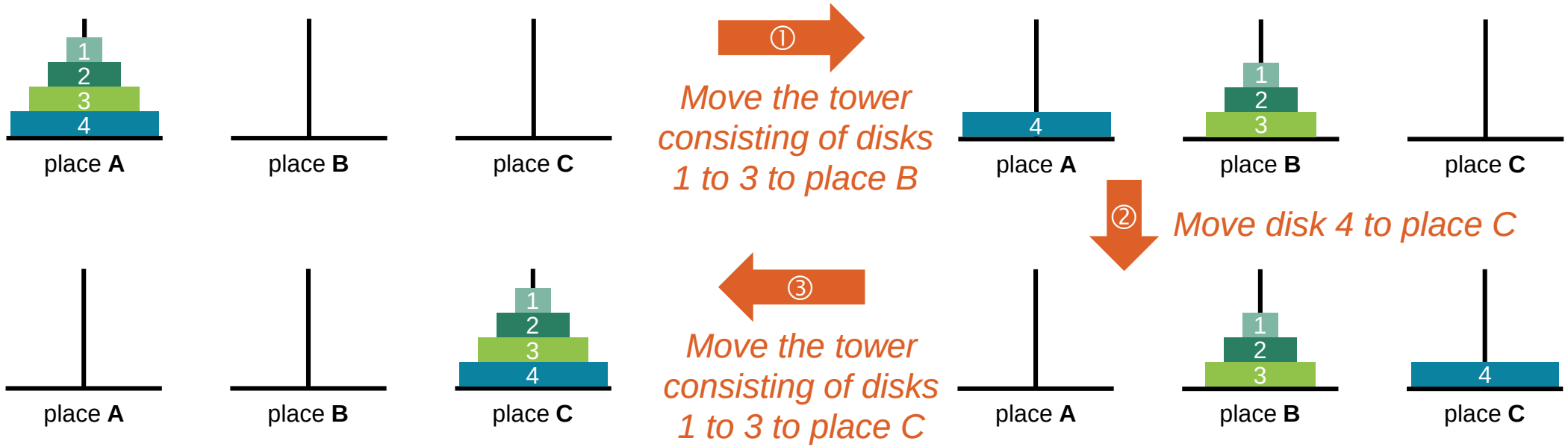
Let's number the disks of the tower from 1 to N, starting with the smallest disk and let's denote the three places as A, B and C. Thus the starting configuration looks like this:



We now are given the task to move the four disks from place A to place C without having a disk with a higher number above a disk with a lower number at any point in time.

Towers of Hanoi: Example with Four Disks

The problem of shifting four disks from place A to place C can be solved in three steps as follows:



So far so good, but now we need a way to move a tower of three disks from one position to another…

How can we solve this? → By applying the same strategy recursively!

Towers of Hanoi: Generalized Recursive Approach

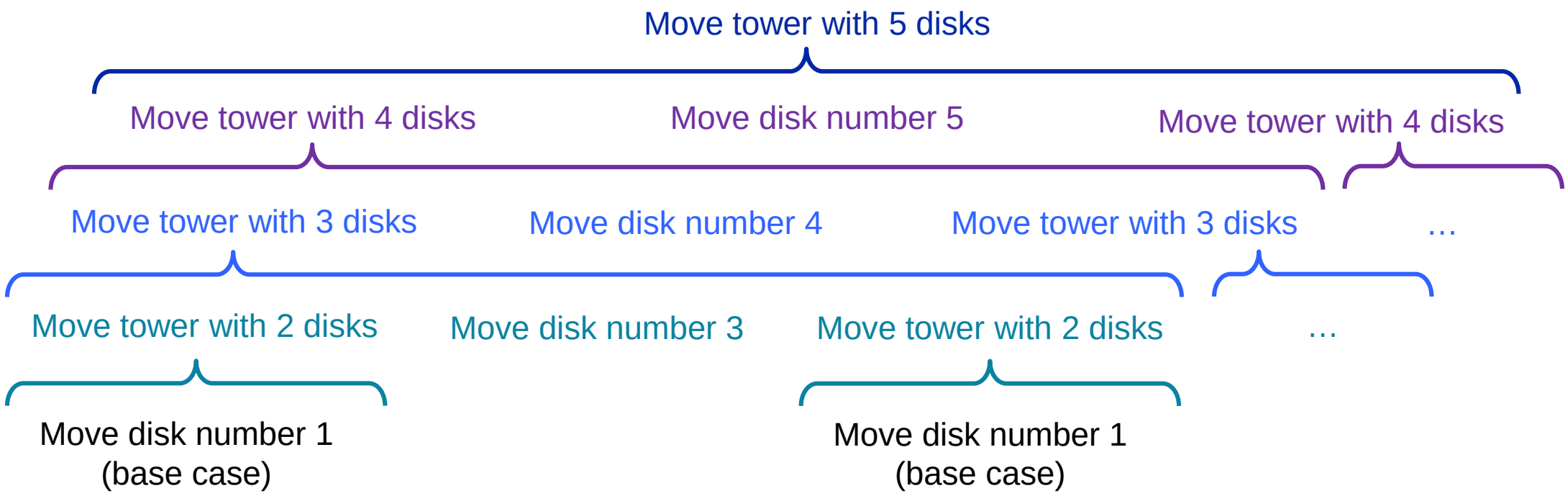
From this, we have a generalized, recursive solution to move a tower with k disks (i.e. disks from 1 to k) from place A to place C:

1) Move the tower with disks 1 to k − 1 from place A to place B
2) Move disk k (biggest disk of the tower of size k) on place C (only remaining valid place)
3) Move the tower with disks 1 to k − 1 from place B to place C

Steps 1) and 3) are applied recursively. The base case is reached when the tasks has reduced to shifting a tower of size 1, i.e. shifting a single disk.

Towers of Hanoi: Example with Five Disks

Graphical depiction of the recursive calls (recursion tree) for the example of five disks:



Towers of Hanoi: C Code Implementation

Possible implementation of a program which prints a sequence of movements which will solve the Towers of Hanoi problem:

```
01 void printHanoiOperations(int numberOfDisks, char fromPlace, char viaPlace, char toPlace) {
02         if (numberOfDisks == 1) {
03                 printf("%c -> %c\n", fromPlace, toPlace);
04         }
05         else {
06                 printHanoiOperations(numberOfDisks - 1, fromPlace, toPlace, viaPlace);
07                 printHanoiOperations(1, fromPlace, viaPlace, toPlace);
08                 printHanoiOperations(numberOfDisks - 1, viaPlace, fromPlace, toPlace);
09         }
10 }
```

Preview on Exercise 2

Exercise 2 – Task 1: Hailstones

Consider a finite sequence $s(n) = a_1, a_2, \ldots, a_m$ with $a_1 = n$ and $a_m = 1$ that is defined as follows:

$$a_{i+1} = \begin{cases} a_i / 2 & \text{if } a_i \text{ is even} \\ 3a_i + 1 & \text{if } a_i \text{ is odd} \end{cases}$$

1. Determine s(3).
2. Write a C program with a *recursive* function `void sequence(int n)` that computes and prints all elements of s(n).

Exercise 2 – Task 2: Set of Binary Strings Without Consecutive 1s

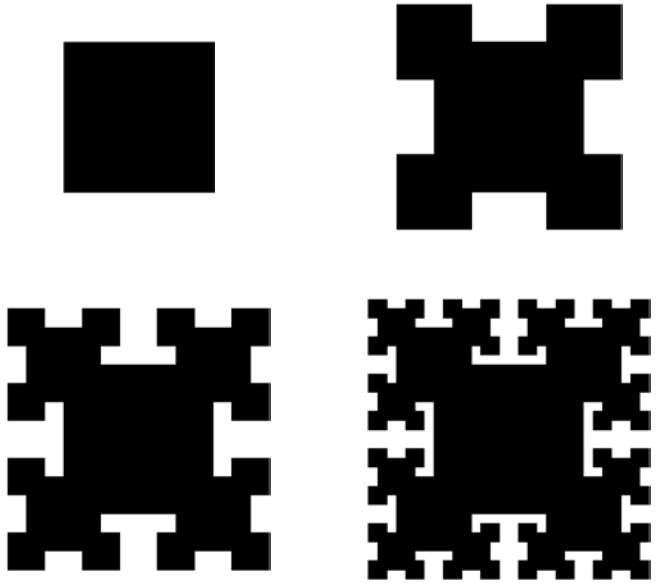
Let B_k be the set of binary strings of length k that do not include consecutive 1s.

1. Determine B_3.
2. Write a C program with a *recursive* function that reads length k from the command line and prints B_k.

Exercise 2 – Task 3: T-Square Fractal

Remark:

– The important thing in this task is that you understand how such figures can be created recursively. Don’t waste too many time on dealing with technical problems like including and linking the graphics library.



Roundup

- Summary
- Outlook
- Questions

Outlook on Next Tuesday's Lab Session

Next tutorial: Tuesday, 06.03.2020, Y35-F-32

Topics:

- Review of Exercise 2
- Preview to Exercise 3
- Algorithmic Complexity
- Running Time Analysis
- … (your wishes)

Questions?

Thank you for your attention.