Lab session 6: Programming Fundamentals

Lab session 6 consist of two parts. The first part consists of the exercises 1, 2 and 3, which are programming exercises in C (like previous labs). These exercises are worth 2 points each (totalling 6 points). The remaining exercises are exercises in Dafny, and together are worth the remaining 4 points.

Part I: Programming in C

Problem 1: Balanced splitter (2 points)

Consider the following series of numbers: $1, 5, 3, 2, 7, 5, 6, 8, 9, 3, 3, 5, 4, 2, 6, 7, 8$
Given some value n we can split this series in three groups: group A of numbers less than n, group B of numbers that are equal to n, and group C of numbers that are greater than n. In this example, for $n = 5$, we get the groups $A = [1, 2, 2, 3, 3, 3, 4]$, $B = [5, 5, 5]$, $C = [6, 6, 7, 7, 8, 8, 9]$. Note that the sizes of A and C are equal: both consist of 7 elements. In such a case, we call n a *balanced splitter*.
The input for this problem consists of a series of positive integers, where each integer is at most 100. The series is terminated by a zero. The output should be the smallest balanced splitter if it exists, otherwise it should output `UNBALANCED`.

Example 1:

input:
```
1 5 3 2 7 5 6 8 9 3 3 5 4 2 6 7 8 0
```
output:
```
5
```

Example 2:

input:
```
1 1 5 3 2 7 5 6 8 9 3 3 5 4 2 6 7 8 0
```
output:
```
UNBALANCED
```

Example 3:

input:
```
1 1 2 3 4 0
```
output:
```
2
```

Problem 2: Who's next? (2 points)

A *permutation* of a sequence is a reordering of the elements in the sequence. For example, given the letters A, B, C, then all its permutations in lexicographic (i.e. dictionary) order are: ABC, ACB, BAC, BCA, CAB, CBA. So, given the permutation BAC, its *lexicographic* (i.e. alphabetic) successor in the sequence of lexicographic ordered permutations is BCA.

The input of this problem is a string containing uppercase letters from the alphabet ('A'..'Z'). The string is terminated by a period (`'.'`). The output of this problem must be the lexicographic successor of the input string if it exists, otherwise the output must be the input string itself.

Example 1:
input:
```
BAC.
```
output:
```
BCA
```

Example 2:
input:
```
CBA.
```
output:
```
CBA
```

Example 3:
input:
```
ABCFDDA.
```
output:
```
ABDACDF
```

Problem 3: To the other side (C program, 2 points)

Given is a grid with `R` rows and `C` columns filled with decimal digits. The output should be the *cheapest path* from the first column to the last column, where a path consist of a series of connected grid points `(x,y)`, and the *cost* of the path is the sum of the grid cells along that path. The cheapest path is the path with the lowest sum. There is a restriction on the paths: from location `(x,y)` the path can only go to one of the locations `(x+1,y-1)`, `(x+1,y)`, or `(x+1,y+1)` provided that these locations are within the boundaries of the grid. In other words, eah step of the path goes into the eastern direction, and never goes (back) in the western direction.

The first line of the input contains the numbers `R` (number of rows) and `C` (number of columns), where $1 \leq R \leq 1000$ and $1 \leq C \leq 1000$. Next follows the grid: `R` lines with `C` digits. For the example inputs below, the following figures show cheapest paths. Note that these path need not be unique, but the minimal cost is unique.

4	5	2	1	1
(0)	0	8	6	4
1	(0)	9	(0)	0
5	9	(0)	9	(0)
0	2	9	7	1

4	1	6	7	9
2	8	4	7	3
(1)	6	2	6	(2)
8	(4)	(0)	(2)	4
7	8	4	9	7

8	(0)	9	4	6	7
(0)	1	(2)	8	3	1
5	3	3	(0)	3	(1)
1	2	2	1	(3)	2
8	2	1	9	0	3

Example 1:
input:
```
5 5
4 5 2 1 1
0 0 8 6 4
1 0 9 0 0
5 9 0 9 0
0 2 9 7 1
```
output:
```
0
```

Example 2:
input:
```
5 5
4 1 6 7 9
2 8 4 7 3
1 6 2 6 2
8 4 0 2 4
7 8 4 9 7
```
output:
```
9
```

Example 3:
input:
```
5 6
8 0 9 4 6 7
0 1 2 8 3 1
5 3 3 0 3 1
1 2 2 1 3 2
8 2 1 9 0 3
```
output:
```
6
```

Part II: Dafny exercises

The exercises 4, 5, and 6 are Dafny programming exercises (like we did before in C), and do not require any program verification. The purpose of these exercises is to get comfortable with the compiler and a small IO (input/output) library that we made for Dafny. You include it by placing the line `include "io.dfy"` (which can be downloaded from Themis) at the top of your program. You should copy the file `io.dfy` into the directory in which you are making your lab exercises. Note that you should only submit your program file to Themis. Do not submit the file `io.dfy` to Themis (testing will fail if you do this).

The exercises 7,8,9, and 10 are small problems that involve correctness verification. These exercises are worth 0.5 point each (totalling 2 points).

The following program demonstrates how to use the IO library to read an `int` and a `nat` from the input. It outputs the sum of the two numbers.

```
include "io.dfy"

method Main()
{
    var a : int, b : nat;
    a := IO.ReadInt();
    b := IO.ReadNat();
    print "a+b=", a + b, "\n";
}
```

Problem 4: Missing digit (Dafny program, 0.5 point)

The input for this problem consists of 9 decimal digits, without duplicates. The output should be the missing digit which is not in the input.

Example 1:

input:

```
0 1 2 3 4 6 7 8 9
```

output:

```
5
```

Example 2:

input:

```
1 3 6 4 2 7 9 8 0
```

output:

```
5
```

Example 3:

input:

```
1 3 4 5 9 8 6 0 7
```

output:

```
2
```

Problem 5: Epoch (Dafny program, 0.75 point)

[Note: you may recognize that this problem is taken from an old 1/3rd term (year 2023/24). However, you have to implement it in Dafny this time.]

Standard Unix systems have a built-in function `time()` that returns the time as the number of seconds since the *Epoch*. The Epoch is defined as 1970-01-01 00:00:00 (1 January 1970, 00:00:00). This day was a Thursday.

Write a program that reads from the input a positive integer n, which is the number of seconds since the epoch. The output should be the day of the moment in time t hat is n seconds after the epoch. As an example, For $0 \leq n < 24 * 3600 = 86400$ the output should be `Thursday`, but for $n = 24 * 3600 = 86400$ the output should be `Friday`.

To avoid confusion about the spelling of the days of the week, the output should be any of [`Monday`, `Tuesday`, `Wednesday`, `Thursday`, `Friday`, `Saturday`, `Sunday`].

Example 1:

input:
```
0
```
output:
```
Thursday
```

Example 2:

input:
```
86399
```
output:
```
Thursday
```

Example 3:

input:
```
86400
```
output:
```
Friday
```

Problem 6: Right-truncatable primes (Dafny program, 0.75 point)

[Note: you may recognize that this problem was discussed in one of the tutorials. However, you have to implement it in Dafny this time.]

A *right-truncatable prime* is a prime which remains prime when the least significant digits are successively removed. For example, 7393 is a right-truncatable p rime, because 7393, 739, 73, and 7 are all prime.

The input for this problem is an integer n, where $0 \leq n \leq 10^9$. The output must be `YES` if n is a right-truncatable prime, and `NO` otherwise.

Example 1:

input:
```
7393
```
output:
```
YES
```

Example 2:

input:
```
42
```
output:
```
NO
```

Example 3:

input:
```
3137
```
output:
```
YES
```

Problem 7: Hoare triple (Dafny program, 0.5 point)

From Themis you can download the file `hoare.dfy`. It contains the following two methods (and some auxiliary code).

```
method GivenHoareTriple(X: int, Y: int) returns (x: int, y: int)

method ExerciseHoareTriple(X: int, Y: int) returns (x: int, y: int)
```

The first method (`GivenHoareTriple`) is a complete implementation and proof of the Hoare triple (which was also discussed in the lecture):

```
{x==y*y + X && y + 1 == Y} x := x + 2*y + 1; y := y + 1; {x == y*y + X && y == Y}
```

You are required to make an alternative implementation and annotation (in `ExerciseHoareTriple`) which start with an assignemtn to `y`, followed by an assignment to `x`. In other words,

```
{x==y*y + X && y + 1 == Y} y := ??; x := ??; {x == y*y + X && y == Y}
```

Problem 8: Square n, Cube n (Dafny program, 1.5 point)

Download from Themis the file `sqcube.dfy`. It contains the following (trivial) program. The call `SquareCube(n)` returns $\mathtt{n}^2$ and $\mathtt{n}^3$ (for $n >= 0$).

```
include "io.dfy"

method SquareCube(n: int) returns (sq: int, cb: int)
requires n >= 0
ensures sq == n*n && cb == n*n*n
{
  sq := n*n;
  cb := n*sq;
}

method Main()
{
  // Do not change the following code. It is used for Themis testing

  var n := IO.ReadNat();
  var square,cube := SquareCube(n);
  print n, "*", n, "=", square," ",n, "*", n, "*", n, "=", cube, "\n";
}
```

Replace the body of the function `SquareCube` by a loop that computes the same answer. The only operations that you are allowed to use on the right hand side of an assignment is addition and multiplication by 2 and 3. Annotate your program with a suitable invariant, and a suitable variant function (i.e. `decreases` clause).