# Lab session 1: Imperative Programming

**Warning for experienced programmers: In lab session 1 you are not allowed to make use of loop-constructs nor recursion (which are discussed in weeks 2 and 5).**

## Problem 1: Hello world!

The first exercise is to make a small computer program that outputs `Hello world!` on the screen. The goal of the exercise is to get acquainted with the programming environment on the lab computers and the automatic assessment system *Themis*.

1. Log on to the lab system. After log in, open a terminal window. In this terminal you can type commands. Start by creating a directory `impprog` that you will use for the course Imperative Programming. You create this directory with the command:

   ```
   mkdir impprog
   ```

2. Navigate to this directory using the command `cd` (change directory).

   ```
   cd impprog
   ```

3. Create another directory `week1`, which is a subdirectory of `impprog`. Create in the directory `week1` another subdirectory `hello` and navigate to this directory:

   ```
   mkdir week1
   cd week1
   mkdir hello
   cd hello
   ```

   Create the file `hello.c` using an editor (for example `geany`).

   ```
   geany hello.c &
   ```

   Type the following program, replace `...` by the right information, and save it.

   ```
   /* file:    hello.c               */
   /* author:  ...... (email: ....) */
   /* date:    ......               */
   /* version: .....                */
   /* Description: This program prints 'Hello world' */

   #include <stdio.h>
   #include <stdlib.h>
   #include <math.h>

   int main(int argc, char *argv[]) {
     printf("Hello world!\n");
     return 0;
   }
   ```

4. Compile the program using the command:

   ```
   gcc -std=c99 -Wall -pedantic hello.c
   ```

5. On successful compilation, an executable file `a.out` is produced. Run the program:

   ```
   ./a.out
   ```

6. If you are convinced that your program works well, you can submit it to the online assessment system Themis. You can reach Themis via the link `https://themis.housing.rug.nl` using your favourite webbrowser. You have completed this first task once Themis accepted your submission.

## Problem 2: Camping

The Jones family has set up the tent on a camping site. It's hot and for the kids they want to fill a bath with water. Father has two jerrycans, each having a volume of 12 litres. He is strong enough to walk with the jerrycans to the nearest water tap and carry them back completely filled.

Write a program that asks how many litres of water is needed to fill the bath. You may assume that this number is an integer. The program must print how often Dad must walk up and down to fill the bath.

**Example 1:**
**input**:
```
240
```
**output**:
```
10
```

**Example 2:**
**input**:
```
480
```
**output**:
```
20
```

**Example 3:**
**input**:
```
2400
```
**output**:
```
100
```

## Problem 3: Palindromic numbers

A *palindromic number* is a number that remains the same when its digits are reversed. For example, 11 and 1221 are palindromic numbers. Write a program that reads a positive integer `n` from the input (you may assume that $10 \leq \texttt{n} < 1000$), and outputs whether the number is palindromic or not.

**Example 1:**
**input**:
```
23
```
**output**:
```
23 is not a palindromic number.
```

**Example 2:**
**input**:
```
11
```
**output**:
```
11 is a palindromic number.
```

**Example 3:**
**input**:
```
12
```
**output**:
```
12 is not a palindromic number.
```

## Problem 4: Order

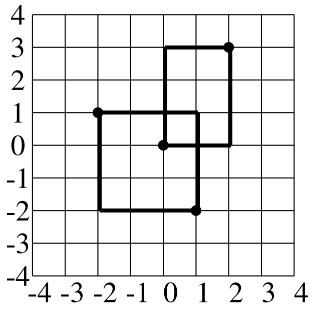
Write a program that reads four integers from the input and outputs them in ascending order. [Hint: a heavily nested if-else-statement is not a good way to solve this problem.]

**Example 1:**
**input**:
```
1 2 3 4
```
**output**:
```
1 2 3 4
```

**Example 2:**
**input**:
```
4 3 2 1
```
**output**:
```
1 2 3 4
```

**Example 3:**
**input**:
```
1 3 2 4
```
**output**:
```
1 2 3 4
```

## Problem 5: Overlap



In the figure on the right, you see a grid containing two rectangles. One rectangle is defined by the coordinates of the corner points $(0, 0)$ and $(2, 3)$. The other rectangle is determined by the coordinates $(-2, 1)$ and $(1, -2)$. Of course, the grid in the figure is finite, but in reality the grid is assumed to be infinitely large. From the figure it is clear that the two rectangles overlap. We call two rectangles *overlapping* if they have at least one point in common, so two rectangles that touch each other are also overlapping.

Write a program that first reads from the input the coordinates of the two corner points of a rectangle, and then reads the two corner points of a second rectangle. You may assume that the corner points are grid points, i.e. their coordinates are integers. The program must print on the screen whether the two rectangles overlap. Make sure that your program produces output in exactly the same format as given in the following examples.

**Example 1 (see figure):**
**input**:
```
0 0 2 3
-2 1 1 -2
```
**output**:
```
overlap
```

**Example 2:**
**input**:
```
0 0 2 3
-1 0 -2 -2
```
**output**:
```
no overlap
```