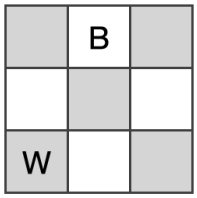
Assignment 4: Fun with Languages (30%)

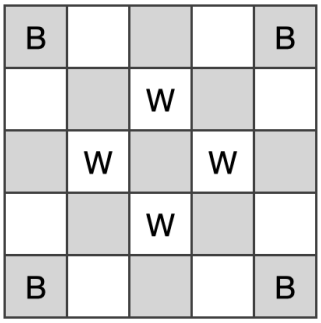
Choose **one** of the two following topics.

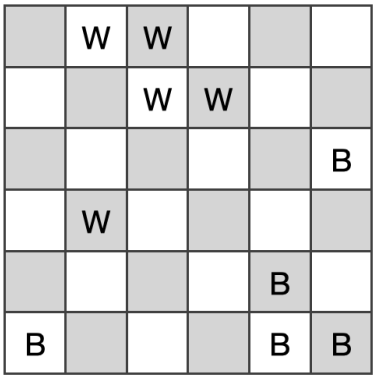
Requirements for the assignment are specified at the end of each topic.

1. CHESS ARMIES OF QUEENS

In chess, a queen attacks positions from where it is, in straight lines up-down and left-right as well as on both its diagonals. It attacks only pieces not of its own colour. The goal of Peaceful Armies of Queens is to arrange **m** black queens and **m** white queens on an n-by-n square grid, (the board), so that no queen attacks another of a different colour. Here are three examples: m=1,n=3; m=4,n=5; and m=5,n=6.







The earliest known reference to this problem was by Stephen Ainley in his 1977 book *Mathematical Puzzles*. Ainley found layouts for square chessboards up to size 30×30.

TASK

Design and implement an Ada program, **`queenarmies.adb`**, to represent two-colour queen armies on a 2-D board.

- Create an algorithm to generate at least two solutions to placing **m** equal numbers of black and white queens on an **n** square board. The program should work for **m** up to a value of 4 and **n** up to a value of 10.

- Include appropriate functionality to decide whether the inputs are valid. For example, m=3, and n=3 is not possible.

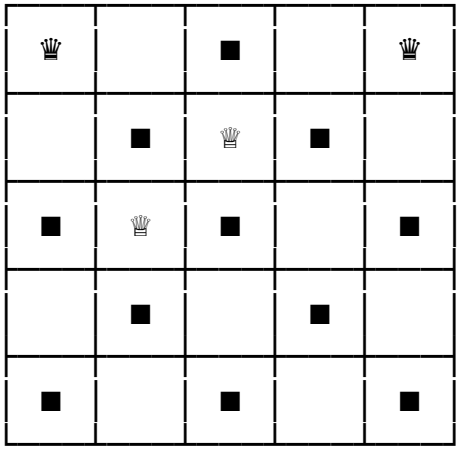
- Allow the user to input the values **m** and **n**, and output the result generated in the form of a graphic.

- The visualization should be intuitive. You can choose to use standard ASCII characters, as in the example shown in the first figure below - For example, for m=2 , and n=5 board, where "x" and "o" represent the alternating colours of the chess board. Alternatively you could embellish the visualization using Unicode chess pieces etc. as shown in the second example. The level of visualization quality will be reflected in the rubric.
    - The code for the visualization should exist in an Ada package: **`showboard.adb`**.

ASCII

```
+---+---+---+---+---+
| B | o | x | o | B |
+---+---+---+---+---+
| o | x | W | x | o |
+---+---+---+---+---+
| x | W | x | o | x |
+---+---+---+---+---+
| o | x | o | x | o |
+---+---+---+---+---+
| x | o | x | o | x |
+---+---+---+---+---+
```

Unicode



Use appropriate modularity in the program. Please refrain from writing the entire program in one long piece of code.

REFERENCE MATERIAL

- *Peaceably Coexisting Armies of Queens* by Robert A. Bosch. Optima, the Mathematical Programming Society Newsletter, Issue 62. (1999) https://www.mathopt.org/Optima-Issues/optima62.pdf

- van Bommel, M.F., MacEachern , K.T., "Armies of Chess Queens", *The Mathematical Intelligencer,* 40, pp.10-15 (2018)

- Literature links: https://oeis.org/A250000

DELIVERABLES

The submission should consist of the following items:

- The reflection report (PDF).
- The code (well documented and styled appropriately of course):
  `queenarmies.adb`
  `showboard.adb`
- Both the code and the reflection report should be submitted as a ZIP, TAR, or GZIP file.

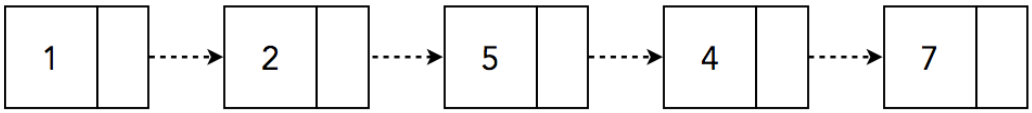
2. UNBOUNDED INTEGERS

The problem with integers is, that regardless of the system, they are bounded, making it difficult to calculate large factorials or higher Fibonacci numbers. Unbounded arithmetic operations are quite important in modern mathematical languages like Matlab, Mathematica, and Maple. In those systems unbounded signed integers are represented as *linked lists* of machine size integers. For example, consider the following series from the Factorial:

```
18 6402373705728000
19 121645100408832000
20 2432902008176640000
21 -4249290049419214848
```

When !21 is calculated, an overflow occurs. Using a linked list would mean that the largest numbers can easily be stored, and manipulated.

Using a linked list to store an integer, for example, the number 12547 would be represented as:



For example, adding two unbounded integers would require adding the associated linked list nodes in some manner.

TASKS

Design and implement a Fortran program, **`unbounded.f03`**, to deal with unbounded integers.

- The program should allow the standard operations of addition, subtraction, multiplication, and division.
- The program should verify that the number input is valid, e.g. only contains digits, and negative/positive signs.
- The program should store the numbers using a *dynamic linked list*. The linked list should be provided in the form of a module, **`dynllist.f03`** - the usual functions to build a linked list, and print it out will be required.
- Implement a user interface like the one shown in the next section to make it work.

- Implement a means of calculating any size Factorial (up to 42!), outputting it in its entirety. For example the value of 42! is:

```
1405006117752879898543142606244511569936384000000000
```

Use appropriate modularity in the program.

THE USER INTERFACE

The user interface will be very simple. It will be terminal input and output. For example:

```
> Enter an operation: + - * / or !
 *
> Enter first operand:
-57890223456789999999999999999
> Enter second operand:
37894289377777777222222222222
> The result is:
-219370887981581921271387585733497081501009602227777777777777778
```

DELIVERABLES

The submission should consist of the following items:

- The reflection report (PDF).
- The code (well documented and styled appropriately of course):
  `unbounded.f03`
  `dynllist.f03`
- Both the code and the reflection report should be submitted as a ZIP, TAR, or GZIP file.