Assignment 2: Ada Programming (25%)

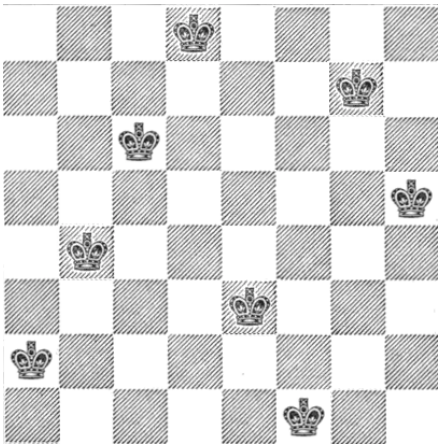
Choose **one** of the following topics.

1. 8-QUEENS

The eight-Queens puzzle was first proposed in 1848 by the German chess player Max Bezzel (1824-1871), and published in the German chess newspaper *Berliner Schachzeitung*. The first to discover a solution was Franz Nauck in 1850, published in *Leipziger Illustrierte Zeitung*. His solution incorporated the 12 fundamental patterns, excluding rotations for 90°, 180° and 270°, and mirror images. That these 12 patterns exhaust all the possibilities was proved in *Philosophical Magazine* in 1874 by English mathematician JWL Glaisher (1848-1928). Carl Friedrich Gauss (1777-1855), also took a passing interest, but did not completely solve it. What seems like a somewhat trivial problem is actually one which may be best solved using a combination of trial-and-error and brute force. The problem is stated as follows:

Place eight Queens on a chess board in such a way that no queen checks against any other queen.

The 8-Queens problem has the following characteristics:
- A queen can attack in any of the compass directions N, NE, E, SE, S, SW, W, NW.
- A queen can attack at any distance as long as it's in a valid direction.
- There are 4,426,165,368 possible arrangements.
- The puzzle has 92 distinct solutions.



Dijkstra used the 8-Queens problem to illustrate the concept of structured programming (Dahl, 1972) (it's a depth-first backtracking algorithm).

TASK

Design and implement two Ada programs for solving the 8-Queens problem:
1. An Ada program that uses a non-recursive back-tracking type algorithm incorporating an ADT such as a stack (implement an Ada package).
2. An Ada program which uses a recursive algorithm (Wirth (1976) or similar).

The programs output should include all 92 distinct solutions, output to file: **queensNR.txt** (non-recursive), and **queensR.txt** (recursive). Visual output should be of the form:

```
. . . Q . . . .
. . . . . . Q .
. . Q . . . . .
. . . . . . . Q
. Q . . . . . .
. . . . Q . . .
Q . . . . . . .
. . . . . Q . .
```

Each program should include a subroutine `solve8Queens()` to solve the 8-Queens problem, and another function `visual8Queens()` that generates the 8-Queens graphics to file.

You can add as many additional functions as you feel relevant.

REFS

- Glaisher, J.W.L., "On the problem of the eight queens", Edinburgh Philosophical Magazine, Vol.4(48), pp.457-467 (1874)
- Nauck, F., "First complete solution of eight queens problem", Leipziger Illustrierte Zeitung, Vol.361, pp.352 (1850)
- Bezzel, F.W.M., "Eight Queens problem", Berliner Schachzeitung, Vol.3, pp.636 (1848)
- Wirth, Niklaus, Algorithms + Data Structures = Programs, Prentice-Hall (1976)
- Wirth, N., "Program development by stepwise refinement", Communications of the ACM, Vol. 14(4), pp.221-227 (1971)
- Dahl, O.J., Dijkstra, E.W., Hoare, C.A.R., Structured Programming, Academic Press (1972) - see pp 72-82 for Dijkstra's solution of the 8 Queens problem.

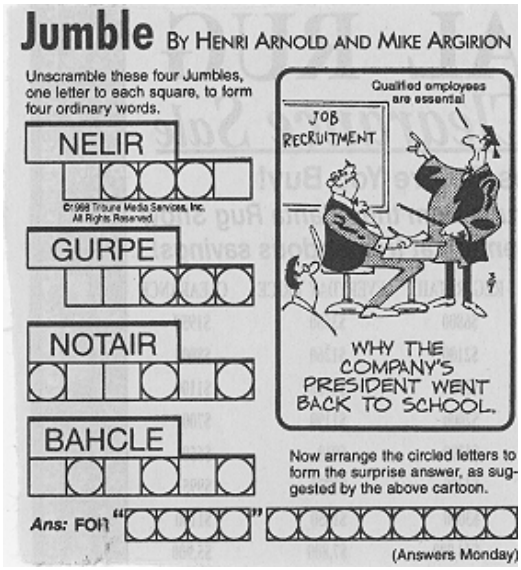
- An excellent reference for n-Queens problems can be found here:
`http://www.liacs.nl/~kosters/nqueens/index.html`

SKILLS

- Ada programming, re-engineering by translation, program comprehension.

2. WORD JUMBLE

An anagram is a type of word play, the result of rearranging the letters of a word or phrase to produce a new word or phrase. For example the anagram of **tea** is tea, tae, eat, eta, aet, ate. Anagrams can be used to solve word jumble problems. If a user provides a series of jumbled letters, all anagrams of the letters can be generated and then checked to see if they exist in a dictionary. The anagrams appearing in the dictionary are printed as potential solutions to the puzzle.



TASK

Design and implement an Ada program (`solveJumble`) which solves the individual word jumbles and uses a recursive (or non-recursive if you like) function to generate anagrams from a series of letters. This involves the following series of modules for each of the *n* jumbles:

1. A subroutine, `inputJumble()`, to obtain user input for the set of jumbles, i.e. jumbled letters NELIR, GURPE.
2. A subroutine, `generateAnagram()`, to generate the anagrams from the letters for each jumbled word.
3. A subroutine, `buildLEXICON()`, which builds a dictionary to search for the anagram in, from the system dictionary (this should be done once for each session). On OSX the dictionary file on OS/X is located in `/usr/share/dict/words`. Use a data structure of some form.
4. A subroutine, `findAnagram()`, to search for the anagrams in a dictionary, printing all results.

The program should run continuously allowing the user to enter as many jumbled words as they wish, providing an option to terminate the program. You may include any other subroutines you deem appropriate beyond those cited above.

Hint: Search for information on permutations.

TESTING

Test your algorithm using the following jumble:

OLIOG
RLCKE
DYOFLN
EMHBUL

The algorithm finds the following words:

```
OLIOG     logoi, igloo
RLCKE     clerk
DYOFLN    fondly
EMHBUL    humble
```

You can stop at this point. However a complete algorithm would choose the appropriate words, and solve the word jumble query from the circled letters in the puzzle.

Using the most appropriate words gives:

```
IGLOO
CLERK
FONDLY
HUMBLE
```

Which gives the following letters:

```
I G E R D B
```

Now the algorithm finds two words:

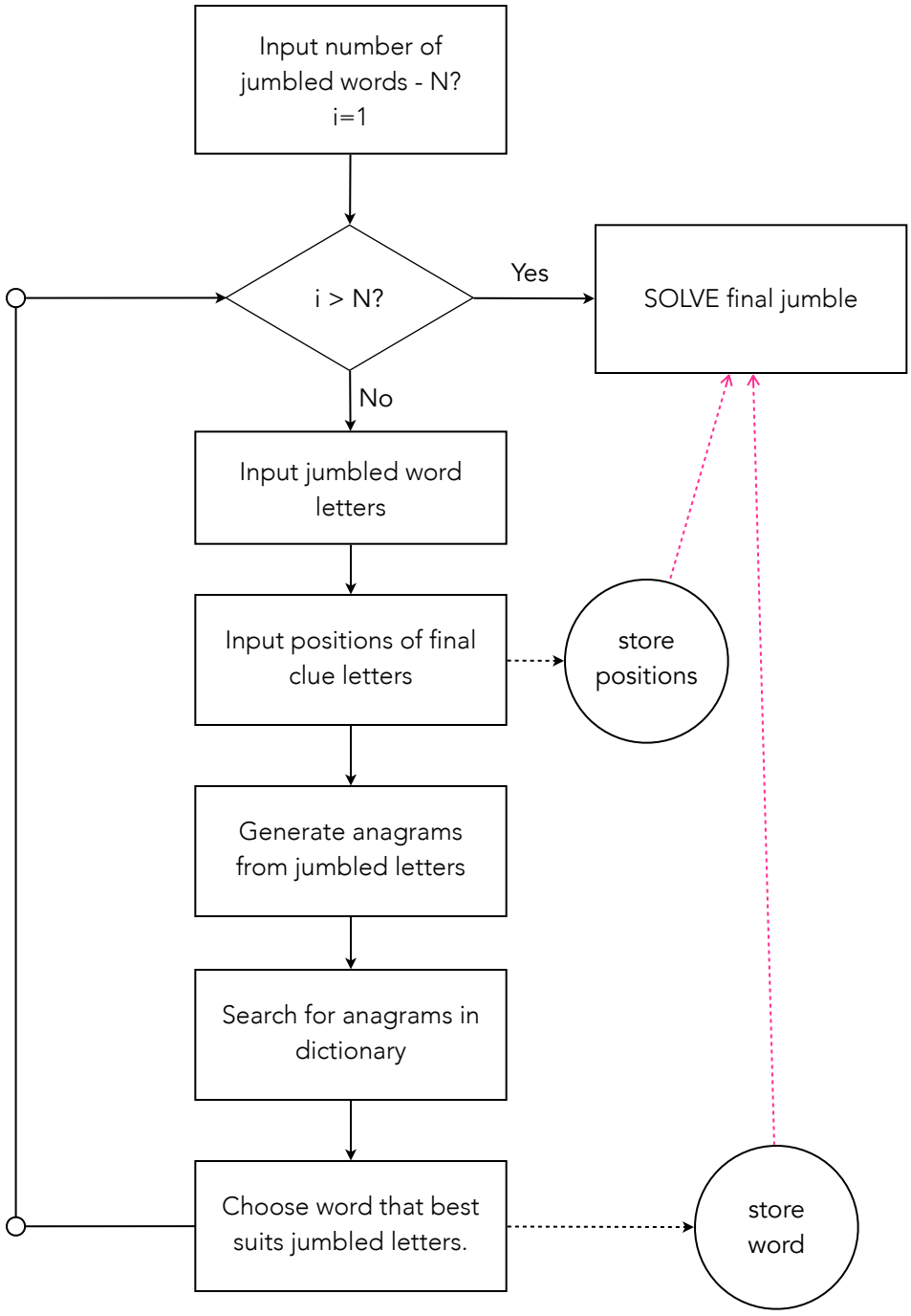
```
BEGIRD, BRIDGE
```

Using the clue: “The creator of Star Trek built one to reach new audiences”, the answer is **bridge**.

REFS

- Knight, D.G., "Anagrams and recursion", *Teaching Mathematics and its Applications*, Vol.5(3), pp. 138-140 (1986).
- Morton, M., "Recursion + data structures = Anagrams", *BYTE*, Vol.12(13), pp.325-334 (1987).

APPENDIX: Flow chart for complete “Jumble” solver



WRITE-UP INFORMATION
(FOR EITHER CHOICE)

REFLECTION REPORT

Describe your Ada program in two (2) page **reflection report**, explaining decisions you made in the process of designing the program. Consider the design document a synopsis of your programming process. One page should include a synopsis of the approach you took to design the program (e.g. it could be a numbered list showing each step in the process).

Some of the questions that should be answered include:

- What structures did you select?
- Was Ada well suited to solving the problem?
- What particular structures made Ada a good choice?
- Benefits / limitations?

DELIVERABLES

Either submission should consist of the following items:

- The reflection report (PDF).
- The code (well documented and styled appropriately of course). The code should be submitted as a ZIP, TAR, or GZIP file.

SKILLS

- Ada programming, program comprehension, problem solving