Assignment 2: Ada Programming (25%)

Choose **one** of the following topics.

1. HANGMAN - FORTRAN TO ADA

Consider the program attached - an "early" Fortran program that plays the word guessing game Hangman. It has been translated from a BASIC program[1].

The game of hangman may have originated in the Victorian era, and involves trying to guess a word by suggesting letters. The game is played as follows: A word is chosen and the player is invited to guess the word one letter at a time.

When the program is run, it picks a word, tells the player how many letters are in the word, and allows the player to guess a letter in the word. If correct, the program places the letter. If the player's letter is wrong, it begins to hang the stick person. The player is allowed 10 guesses before the stick person is completely hanged:

head
body
right and left arms
right and left legs
right and left hands
right and left feet

A version of hangman using a program would prompt the user by displaying correct guesses in position after each round of guessing. For the first round an underscore character for each letter in the word is displayed. For example, suppose that the computer chooses the word "bench", from the dictionary. The computer would display five dash(-) characters:

```
- - - - -
```

The player then guesses a character that they think is in the word. In the above example, any of the characters b, e, n, c, and h would be correct guesses as they appear in the word bench. When a correct character is guessed, this character is displayed in all the positions it appears in the word. If, for example, the player guessed the letter e, then the letter e would be displayed in position 2:

```
- e - - -
```

---

[1] Aupperle, K., Ahl, D., "Hangman", *BASIC Computer Games*, p.80 (1978)

If an incorrect guess is made the player is informed and the game increments the number of incorrect guesses. As the game progresses, the player gradually guesses more characters that appear in the word. The game ends when either all the characters in the word are guessed and the word is revealed (in this case the player wins), or a specified, maximum number of incorrect guesses are reached (the player loses).

```
THE GAME OF HANGMAN
---
Here are the letters you used:

What is your guess?
g
g--
What is your guess for the word?
get
Wrong. Try another letter
Here are the letters you used:
g,
What is your guess?
m
g-m
What is your guess for the word?
gym
Wrong. Try another letter
Here are the letters you used:
g,m,
What is your guess?
u
You found the word.
Do you want another word? (Y/N)
```

The visual for the program has the following form (shown here with 6 of 10 parts drawn):

```
What is your guess?
q
Sorry, that letter isn't in the word.
This time we draw the left leg.
XXXXXXX
X     X
X    ---
X \ (. .) /
X  \ --- /
X   \ X /
X    \X/
X     X
X     X
X    / \
X   /   \
X
```

TASK

Translate the legacy Fortran program into an Ada program. Properly document the program. You should attempt to modularize the program by decomposing it into smaller subroutines. Your program should compile using the Ada compiler (`gnatmake`).

Make sure to convert/remove any structures which are relevant/irrelevant. The program should be clean and easy to understand (unlike the existing code). The program contains a number of structures which should be modified or removed to make the program more maintainable. Some examples include:

- computed GO TO.
- arithmetic IF statements
- Improved identifier names
- Labels

DESIGN DOCUMENT

Discuss your re-designed program in 3-4 page *design document*, explaining decisions you made in the re-engineering process. Consider the design document a synopsis of your re-engineering process. One page should include a synopsis of the approach you took to translate the program (e.g. it could be a numbered list showing each step in the process). Identify the legacy structures/features and how you modernized them.

Some of the questions that should be answered include:

- Would it have been easier to re-write the program from scratch in a language such as C?
- What were the greatest problems faced during the re-engineering process?
- Is your program shorter or longer? Why?
- Is there a better way of writing the program?
- What particular structures made Ada a good choice?

WANT TO GO BEYOND? (5%)

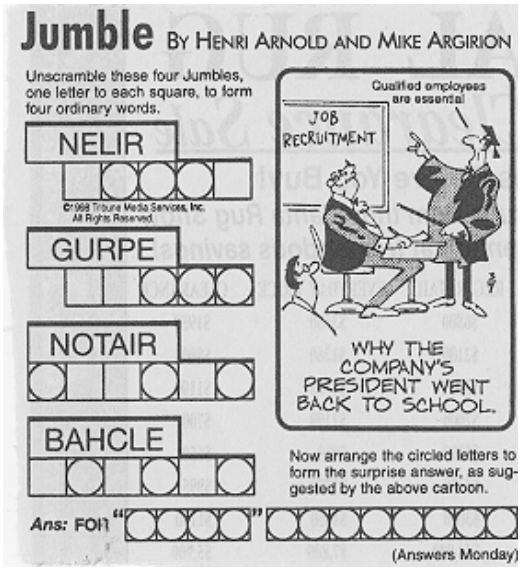
The previous section when completed is worth 95% of the grade for this assignment. To obtain the remaining 5%, you should add to your program the ability to select a word at random from a dictionary file `dict.txt`.

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

Write an Ada program which solves the word jumble 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 to obtain user input for a jumble, i.e. jumbled letters and positions of characters that contribute to the final clue. e.g. O L I O G, and 1, 2 (the latter should be stored somewhere for later).
2. A subroutine to generate the anagrams from the letters.
3. A subroutine to search for the anagrams in a dictionary, printing all results, and allowing the user to choose the appropriate word (and store it somewhere).

You may include any other subroutines you deem appropriate beyond those cited above.

Hint: Search for information on permutations.

Note: For example the dictionary file on OS/X is located in /usr/share/dict/words.

So the algorithm is capable of unscrambling the individual jumbles and then with the next section) finding the surprise answer.

WANT TO GO BEYOND? (5%)

The previous section when completed is worth 95% of the grade for this assignment. To obtain the remaining 5%, you should:

1. Once all the individual clue jumbles have been solved, derive a subroutine to use the letter positions extracted from the user input and the stored un-jumbled words to solve the final jumble (steps 2 and 3 above).

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
```

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**.

DESIGN DOCUMENT

Discuss your re-designed program in 3-4 page *design document*, explaining decisions you made in the process of designing the program. Consider the design document a synopsis of your re-engineering 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?

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).

DELIVERABLES

Either submission should consist of the following items:

- The design document.
- The code (well documented and styled appropriately of course).

APPENDIX: Flow chart for Anagrams