Assignment 4: Comparing Languages (30%)

Choose **one** of the following topics.

1. ACKERMANN'S FUNCTION

Ackermann's function was originally conceived in 1928 by Wilhelm Ackermann, and has been used extensively in the past for studies in computational efficiency, especially by B.A. Wichmann. Ackermann's function is interesting from the point of view that it is a highly recursive function. Ackermann's function has the following recurrence relation:

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \text{ and } n \geq 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases}$$

How does it work? Consider the calculation of A(1,2):

```
A(1,2)
= A(0, A(1,1))
= A(0, A(0, A(1,0)))
= A(0, A(0, 2))
= A(0, 3)
= 4
```

This seems simple, but the larger *m* and *n* become, the more complex the recursion becomes. The iterative version of Ackermann exists, but relies on the use of stacks to implement recursion.

TASK

The code is provided for a recursive version of Ackermann's function in Ada, and a non-recursive version using stacks, implemented in C. Translate both recursive and non-recursive algorithms into Fortran, Ada, and C, and the non-recursive algorithm into Cobol (as Cobol does not allow recursion), and compare the implementations. This assignment is an exercise in code translation and language bench-marking. If you have a Raspberry Pi, you can easily install compilers on it for most languages, and also compare the performance of the languages on the Raspberry Pi, versus any other operating system. Note, Cobol does not allow recursion, so you

DESIGN DOCUMENT

Include a 4-5 page summary, discussing the benefits/limitations of each of the languages (1 page per language). Which language had the best usability? (e.g. was easiest to program, had the most appropriate structures, is easy to maintain). Was there any difference in efficiency?

REFS

- Wichmann, B.A., “Ackermann’s function in Ada”, *ACM SIGAda Ada Letters*, 6(3), pp.65-70 (1986).
- Wichmann, B.A., “Ackermann’s Function: A study in the efficiency of calling procedures”, *BIT* 16 103-110 (1976)

2. THE FLESCH INDEX

In 1949, Dr. Rudolph Flesch developed a formula for estimating the "readability" of written material based on the average number of words per sentence and the average number of syllables per word. The readability index gives an idea of how easy the material is to understand.

Index	**Readability**
90-100	Very easy
80-90	Easy
70-80	Fairly easy
60-70	"Plain English"
50-60	Fairly difficult
30-50	Difficult
< 30	Very difficult

The *Flesch readability index* for a document can be computed in five steps:

1. Count the number of words in the document.
2. Count the number of syllables in the document.
3. Count the number of sentences in the document.
4. Compute the index as:

$$\text{score} = 206.835 - \left(1.015\frac{\text{words}}{\text{sentences}} + 84.6\frac{\text{syllables}}{\text{words}}\right)$$

score = 206.835 - (1.015 (words/sentences) + 84.6 (syllables/words))

5. Round the index to the nearest integer.

TASK

This task involves using a program to calculate the FLESCH Index for a piece of text written in PL/I for a VAX 11/780 system (yikes old!). The program can be found in the following paper by John Talburt (available from the ACM Digital Library):

Talburt, J., "The Flesch Index: An easily programmable readability analysis algorithm", Proc. Int. Conf. on Systems Documentation, pp.114-122 (1985) http://www.csc.villanova.edu/~tway/courses/csc3990/f2009/papers/nlp/talburt_1986.pdf

Your task is to re-engineer, by means of translation, the PL/I program into an Ada program.

You will likely not have any experience in writing programs in PL/I (although it is still used in certain industries), but there is plenty of online information, and literature available (especially in the library). A handout with the basic syntax of PL/I can be found here:

http://bitsavers.trailing-edge.com/pdf/ibm/360/pli/SC20-1651-1_A_Guide_to_PL_I_for_Commercial_Programmers_Apr68.pdf

You can certainly try and find a PL/I compiler to try and run the code, however a comprehension of the code is all that is needed.

Once you have written the Ada program, write a *second* Ada program which translates the formula used into the *Flesch-Kincaid grade level*. The grade level is calculated with the following formula:

$$0.39\left(\frac{\text{words}}{\text{sentences}}\right) + 11.8\left(\frac{\text{syllables}}{\text{words}}\right) - 15.59$$

Your program should output both the *Flesch readability index* and the *Flesch-Kincaid grade level.*

TESTING

Test the program against the reference passages given, where the indices were calculated through manual counting.

passage 1: index=111.38 (64 syllables in 62 words in 8 sentences)
passage 2: index=65.09 (74 syllables in 55 words in 2 sentences)
passage 3: index=3.70 (110 syllables in 71 words in 1 sentences)

DESIGN DOCUMENT

Write a 4-5 page summary, documenting the process of converting the program from PL/I to Ada. Describe the process of converting a true vintage language such as PL/I, which is little used today (unlike Cobol). For example, discuss the decisions made during the re-engineering process, i.e. what did you change, how, etc.

REFS

- Flesch, R., "A new readability yardstick", Journal of Applied Psychology, Vol.32, pp.221–233 (1948).

- Talburt, J., "The Flesch Index: An easily programmable readability analysis algorithm", *ACM*, pp.114-122 (1986).

3. QUICKSORT IN COBOL

In the early 1960s, C.A. Hoare developed the Quicksort algorithm for sorting. Here is the basic algorithm for the Quicksort.

1. Choose an element in the list - this element serves as the pivot. Set it aside (e.g. move it to the beginning or end).
2. Partition the array of elements into two sets - those less than the pivot and those greater than the pivot.
3. Repeat steps 1 and 2 on each of the two resulting partitions until each set has one or less elements.

However the caveat with Quicksort is that it relies on recursion, and there is no facility for recursion in Cobol. None. Nada. Not one little bit. So adding it is somewhat of a challenge... recursively anyways. It is however possible to create a non-recursive version for Cobol.

TASK

Derive an iterative version of Quicksort (some references are given below). A version of a non-recursive algorithm in Cobol is given, written in Cobol 74. Implement the iterative version of Quicksort in the three course languages: Fortran, Ada, Cobol, and at least one other language, e.g. C/C++, Lua, Python and compare the implementations. The programs should prompt for a filename containing 100,000 integers to be sorted (a C program to generate this file will be provided). Your output from each program should be directed into an ASCII file. Compare execution speeds of each of the languages.

When you have completed this, you have one of two choices:

1. Implement a recursive version of the algorithm in C. Perform the same tests and compare the iterative versions against the recursive algorithm. You can use Hoare's original algorithm if you like, or some derivative.

   OR

2. Have your Cobol program call a recursive version of Quicksort built in C - i.e. the Cobol main program will call a recursive C function to perform the sorting.

DESIGN DOCUMENT

Include a 4-5 page summary, justifying your choice of language, and decisions made during the re-engineering process, i.e. what did you change, how, etc. Which language had the best usability? Was there any difference in efficiency?

REFS

- Hildebrand, K., “Implementierung des Quicksort in COBOL”, Angewandte Informatik (Applied Informatics), Vol.31(1), pp.14-18 (1989).
- Hoare, C.A.R., “Algorithm 63: Partition”, Communications of the ACM, Vol.4(7), pp.321 (1961)
- Hoare, C.A.R., “Algorithm 64: Quicksort”, Communications of the ACM, Vol.4(7), pp.321 (1961)
- Sedgewick, R., “Implementing Quicksort programs”, Communications of the ACM, Vol. 21(10), pp.847–857 (1978).

DELIVERABLES

Either submission should consist of the following items:

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