Advanced Programming

Lab Sheet 2 – Problem Solving

Introduction

One of the key skills that developers rely upon when writing software is the ability to *break a problem down into its constituent parts*. Typically, users or other stakeholders are able to tell us what they want the software to do in terms of input and output, with developers relied upon to identify the series of steps that need to be followed to achieve this. This skill, sometimes referred to as *problem decomposition*, can be developed by working on open-ended problems with no prescribed way to complete the task. The tasks in this lab sheet require you to figure out *how* you're going to write the program, rather than just adopting techniques you've been taught as-is.

Task 1: Milestone Birthdays

Some of your birthdays are a bit more special than others. We typically think of certain birthdays as key milestones. Your 16th and 18th birthdays mark your ascent to adulthood. In many countries, the 21st birthday is considered a milestone, as well as the 30th, 40th, 50th, 60th etc. Write a program that allows the user to enter their date of birth, and tells them how long until their next milestone birthday.

Too Easy?

Take a look at Java's LocalDate and LocalTime classes to see if you can make your program's output precise, taking into account the current date and time.

Too Hard?

If you're struggling with this task, it indicates that your out of practice at understanding exactly how you design code to address a written problem. This is a skill that takes practice, so the best way to improve it is to keep writing code to solve problems!

The codingbat website that you were introduced to last year serves as useful practice for this sort of things, although the problems are quite small. There are a lot of useful challenges on edabit too, although I have only reviewed a handful of them.

Task 2: Monthly Temperature Statistics

We regularly see news headlines that inform us that last month was the "Hottest April on record", or the "Coldest January since records began". The data used by journalists to write these low-effort articles is published by the met office using some automated weather stations, see for example the data gathered for Bradford here.

The software that generates the temperature data in these reports will likely be fairly simple, searching through a list of temperature readings for the minimum and maximum reported temperatures, and calculating an average of all readings. Write a program that populates an array of temperature values (you can make them up for testing), and outputs the minimum, maximum, and average temperature.

Too Easy?

Mean averages can be skewed by small numbers of extreme values, and in many cases where an average value is sought, a median average is a better choice. Create a second version of the program that calculates the median temperature instead.

Too Hard?

This exercise is quite similar to the work you did in the earlier lab session this week. Take a look back at your work in Lab 1 Task 3 to see if it can help you to get started. You might also wish to look back over the material from last year on arrays and loops.

Task 3: Word Length & Readability Analysis

Although there are a signifiant number of different theories and associated formulae that seek to capture what makes prose easier or more difficult to read, one of the best known is the *Automated Readability Index* [1]. This probably has more to do with it being one of the easiest to calculate metrics, rather than it being the most accurate. The Automated Readability Index for a body of text can be calculated using the formula depicted in Figure 1, below.

$$4.71\left(\frac{characters}{words}\right)+0.5\left(\frac{words}{sentences}\right)-21.43$$

Figure 1: Automated Readability Index Formula

Write a program that prompts the user to paste a block of text into the console, and then calculates the Automated Readability Index score for the block of text. Test your program by calculating the difficulty of a few different passages. Does the calculated difficulty align with your expectations?

Too Easy?

There are several other readability formulae that you could try to calculate in a similar way to the Automated Readability Index, but they often require you to calculate the number of syllables in each word, or to have a pre-compiled list of “difficult” words. You could calculate Flesch-Kincaid score, or the Gunning Fog Index, which serve as well-known examples of each approach.

If you’re interested in readability metrics in general, I stumbled upon an interesting looking research paper [3] comparing various well-known readability metrics when writing this lab sheet. If you’re working on campus, you should be able to access the PDF of the paper because of the University’s subscription to IEEE here.

Too Hard?

The common stumbling block in this task lies in the splitting up of a long String into individual words. There are several standard approaches for this, but I would recommend taking a look at the .split() method in the String class, which would work well here.

References

1. Smith, E. A., & Senter, R. J, “Automated readability index” Aerospace Medical Research Laboratories (Vol. 66, No. 220).
2. S. Zhou, H. Jeong and P. A. Green, "How Consistent Are the Best-Known Readability Equations in Estimating the Readability of Design Standards?," in IEEE Transactions on Professional Communication, vol. 60, no. 1, pp. 97-111, March 2017, doi: 10.1109/TPC.2016.2635720.