Algorithms and Data Structures: Week 2

Johnson rev. Bell (Michaelmas 2024)

Before your practical: Revise the content from the lectures in the first week. There are some additional videos on Learn Ultra reviewing the questions from the lecture and looking at the additional problem of writing pseudocode to the find the smallest of a collection of numbers — this will be useful when tackling Question 2.

Question 1

Consider the following pseudocode:

Input: integer L, integer H
Output: ?
  integer $x = L$
  integer $p = 1$
  while $x \leq H$ **do**
    if x is odd **then**
      $p = p \times x$
    end if
    $x = x + 1$
  end while
  print "result is ", p

(a) What does this algorithm do, that is, what is being computed in the variable p?

(b) Rewrite this code to use a **for** loop instead of a **while** loop.

Question 2

Design an algorithm in pseudocode that, when given some number n and numbers $a_0, a_1, \ldots, a_{n-1}$ as input, computes and outputs the second smallest of those numbers. In other words, the input/output specification looks something like

Input: integer n, integers $a_0, a_1, \ldots, a_{n-1}$
Output: second smallest of $a_0, a_1, \ldots a_{n-1}$

Finding the smallest is a problem discussed in a video on `Learn Ultra`. So look back at that (or construct your own solution for that problem) and think about how to modify it?
(Note you should **not** include a line in your solution that says something like

  sort $a_0, a_1, \ldots, a_{n-1}$ so they are in order from smallest to largest

This would not be efficient, as sorting the integers is a more difficult problem than the one you are asked to solve.)

Question 3

Write a simple algorithm in pseudocode that tests whether a given number, i.e., a number that is given as input, is prime.

Recall that a natural number k is called prime if it can be divided without remainder only by one and itself.

If required, you may use a condition in an **if** statement such as "a divides b" (which evaluates to true if a divides b evenly, i.e. without a remainder) and you can also use a command **exit** that will cause the algorithm to terminate: once you have found that k is not a prime you can stop.

Question 4

Write a simple algorithm in pseudocode that takes as input the date (given as three integers for year, month and day) and a further integer k. The output should be the date k days after the input.

You can, if you wish, make the simplifying assumption that there are no leap years and, moreover, that every month has the same number of days (so a year contains 360 days divided into 12 30-day months, say).

You may use the following operations: for two integers a and b, $a // b$ gives an integer and $a \,\%\, b$ gives the remainder. (For example $800 // 360 = 2$ and $800 \,\%\, 360 = 80$ so 800 days from now is 2 years and 80 days away).

Extension

If you finish early in any ADS practical, it is good practice to try to implement any pseudocode solutions you might have. Take Questions 3 and 4. The ideas underlying them are drawn from elementary mathematics. But your first attempt at writing an algorithm in pseudocode may have some intricacies and everyone's attempt will be slightly different. So how can you check your solution for correctness? One way is to convert the pseudocode to code and test it. We will talk about testing your solutions in later practicals.

1. So if you can, have a go at implementing your pseudocode solution to Questions 2–4 in Python.

2. Have a go at implementing your pseudocode solution in another programming language that you are familiar with.

3. Adapt your solution to Question 4 so that it accurately models the traditional calendar correctly. As a reward, you can then read about leap seconds (`https://www.nist.gov/pml/time-and-frequency-division/leap-seconds-faqs`).

4. Have a look at Project Euler. Try Problem 19, for example.

Implementation Notes

I know that you might not have studied programming before: at Durham, learning to program is taught in another module. You already have learned that the devil is in the details of how abstract specifications get implemented in code. Testing your ideas by implementing them is a very useful supplementary activity and as you progress in other modules you will be able to apply the skills you gain there in this module.

Most importantly, the assignment for this module **will require you to write Python code** at the level covered in the first term in COMP1051 Computational Thinking (or equivalent). When we write pseudocode in this module, we often use the conventions of Python, easing the process of converting the pseudocode to code. I will occasionally provide examples using Python and your demonstrators will be familiar enough with Python to provide help if needed.

There is a document on Learn Ultra ("Link to Python environment for practicals") that explains how to get started programming in Python on the departmental compute cluster, NCC. Have a look at this and then see if you can write code for question 2 or 4. You are, of course, welcome to write and execute Python code on your own computer. Another option is the JupyterLite software (`https://jupyter.org/try-jupyter/lab/`), which allows you to run Python code in your browser.

To get you started on implementing your answer to Question 1, here's some scaffolding for a possible solution in Python (don't forget to pay attention to whitespace):

```
def do_something(L, H):
    x = ____________________________
    p = 1

    while ____________________________:
        if ____________________________:
            p *= ____________________________
        ____________________________

    return ____________________________

# Example usage:
L = 1
H = 10
result = do_something(L, H)
print("Result is", result)
```