1 Introduction

You will write some simple programs in the Impcore programming language in order to gain familiarity with the language and to practice writing recursive functions.

Download `prog01.imp` and `prog01_tests.imp` and/or `prog01_tests.zip` (or copy from `/usr/local/pub/mtf/plc/programming/prog01-impcore` on the CS Department Linux systems). The former is a template for your submission and also includes a number of supporting functions. The latter is a test suite for the assignment.

2 Description

Complete the following problems. Each one is to define one or more Impcore functions. See Requirements and Submissions for important restrictions.

A. (5pts) Complete Exercise 4 of Chapter 1 from Programming Languages: Build, Prove, Compare (p. 76).

B. (10pts) Complete Exercise 5 of Chapter 1 from Programming Languages: Build, Prove, Compare (p. 76).

C. (5pts) Complete Exercise 6 of Chapter 1 from Programming Languages: Build, Prove, Compare (p. 76).

D. (5pts) Complete Exercise 7 of Chapter 1 from Programming Languages: Build, Prove, Compare (p. 76).

E. (20pts) Complete Exercise 8 of Chapter 1 from Programming Languages: Build, Prove, Compare (p. 76).

F. (20pts) Complete Exercise 9 of Chapter 1 from Programming Languages: Build, Prove, Compare (pp. 77 – 78).

For Exercise 9b (all-fours?), the presence or absence of a negative sign in the decimal representation of the input number should not affect the answer (which is consistent with how negative numbers are treated in Exercise 9c and Exercise 9d). In particular, the exercise transcripts should read

```
-> (all-fours? -4)
1
```

and the unsatisfying answer is wrong (and not just unsatisfying).

G. (5pts) Complete Exercise 10 of Chapter 1 from Programming Languages: Build, Prove, Compare (p. 78).

H. (5pts) The aliquot sum (see [https://en.wikipedia.org/wiki/Aliquot_sum](https://en.wikipedia.org/wiki/Aliquot_sum)) of a positive integer \( n \) is the sum of the proper divisors of \( n \). A positive integer \( n \) is deficient if the aliquot sum of \( n \) is less than \( n \); for example, 15 is deficient, because the proper divisors of 15 are 1, 3, and 5 and \( 1 + 3 + 5 = 9 < 15 \). A positive integer \( n \) is perfect if the aliquot sum of \( n \) is equal to \( n \); for example, 6 is perfect, because the proper divisors of 6 are 1, 2, and 3 and \( 1 + 2 + 3 = 6 \). A positive integer \( n \) is abundant if the aliquot sum of \( n \) is greater than \( n \); for example, 12 is abundant, because the proper divisors of 12 are 1, 2, 3, 4, and 6 and \( 1 + 2 + 3 + 4 + 6 = 16 > 12 \).

Define a function `aliquot-class` such that `(aliquot-class n)` returns -1 if \( n > 0 \) and \( n \) is deficient, `(aliquot-class n)` returns 0 if \( n > 0 \) and \( n \) is perfect, and `(aliquot-class n)` returns 1 if \( n > 0 \) and \( n \) is abundant. (When \( n \leq 0 \), the behavior of `aliquot-class` is unspecified.)
I. (10pts) The Hailstone sequence (see https://rosettacode.org/wiki/Hailstone_sequence) for positive integer $n$ is the sequence of positive integers $a_0, a_1, a_2, \ldots$ such that

- $a_0 = n$
- $a_i$ is the last element of the sequence if $a_i = 1$
- $a_{i+1} = a_i/2$ if $a_i > 1$ and $n$ is even
- $a_{i+1} = 3 \cdot a_i + 1$ if $a_i > 1$ and $n$ is odd

For example, the Hailstone sequence for 21 is 21, 64, 32, 16, 8, 4, 2, 1. The Collatz conjecture (see http://en.wikipedia.org/wiki/Collatz_conjecture) is that the Hailstone sequence for any positive integer is finite.

Define functions `hailstone-length` and `hailstone-max` such that `(hailstone-length n)` returns the length of the Hailstone sequence for $n > 0$ and `(hailstone-max n)` returns the maximum element in the Hailstone sequence for $n > 0$. For example, `(hailstone-length 21)` should return 7 and `(hailstone-max 21)` should return 64. (When $n \leq 0$, the behavior of `hailstone-length` and `hailstone-max` is undefined. If the Hailstone sequence for some $n > 0$ is infinite, then you have disproved the Collatz conjecture (and will be famous).)

3 Requirements and Submission

In addition to the specifications given in the problems, your functions must use recursion (and must not use iteration via `while`) and must not use global variables.

Your submission must be a valid Impcore program. In particular, it must pass the following test:

```
$ cat prog01.imp | /usr/local/pub/mtf/plc/bin/impcore -q > /dev/null
```

without any error messages. If your submission produces error messages (e.g., syntax errors), then your submission will not be tested and will result in zero credit for the assignment.

Submit `prog01.imp` to the Programming 01 Dropbox on MyCourses by the due date.

4 Hints

- You may define additional helper functions.
- Efficient solutions are not required (although may be entertaining to discover).
- There are various ways in which a recursive function can decompose a natural number:
  - Decrement by one
    Base case: $n = 0$
    Recursive case: $n = m + 1$, recurse on $m$
  - Split into two pieces
    Base case: $n = 0$
    Recursive case: $n = k + (n - k)$ (where $0 < k < n$), recurse on $n - k$
  - Sequence of decimal digits
    Base case: $n = d$ (where $0 < d < 10$)
    Recursive case: $n = 10 \cdot m + d$ (where $0 < d < 10$ and $m > 0$), recurse on $m$

The assignment will require you to use at least one more.
- The reference solution is approximately 150 lines of Impcore.
A Interpreter

A reference Impcore interpreter is available on the CS Department Linux systems (e.g., glados.cs.rit.edu and queeg.cs.rit.edu and ICLs 1 and 2) at:

```
/usr/local/pub/mtf/plc/bin/impcore
```

Use the reference interpreter to check your code.

Source code for the interpreter is available on the CS Department file system at:

```
/usr/local/pub/mtf/plc/src/bare/impcore
```

and

```
/usr/local/pub/mtf/plc/src/commented/impcore
```

A.1 Interactive mode

Simply executing

```
$ /usr/local/pub/mtf/plc/bin/impcore
```

will run the interpreter interactively, but without line editing.

Executing

```
$ rlwrap /usr/local/pub/mtf/plc/bin/impcore
```

or

```
$ ledit /usr/local/pub/mtf/plc/bin/impcore
```

will run the interpreter interactively with line editing.

(See the manual pages for `rlwrap` and `ledit` for more details.)

A.2 Batch mode

Executing

```
$ cat prog01.imp | /usr/local/pub/mtf/plc/bin/impcore
```

will run the interpreter on the contents of the file `prog01.imp`, but with prompts printed.

Executing

```
$ cat prog01.imp | /usr/local/pub/mtf/plc/bin/impcore -q
```

will run the interpreter on the contents of the file `prog01.imp` without prompts printed.

Executing

```
$ cat prog01.imp prog01_tests.imp | /usr/local/pub/mtf/plc/bin/impcore -q
```

will run the interpreter on the contents of the files `prog01.imp` and `prog01_tests.imp` without prompts printed.