[Exercise2]

Attached Files:

- Exercise2.zip (4.94 MB)
- makex2.zip (8.308 MB)

Topic: Basic FP Programming

This exercise assumes you have completed Exercise 1 and understand how to install and use the relevant software.

Instructions:

- Download **makex2.zip**, uncompress (produces **makex2** folder, and use your command-line/terminal window inside that folder to issue the command: **stack install**.
  If that succeeds, then you will have installed an executable called **makex2**.
- Go to the folder (FPCW say) where you put Exercise1. It should still have the **makex.conf** file left over from that exercise. Download and uncompress **Exercise2.zip** here.
- Execute the following in FPCW: **makex2**. You should not need to re-enter any details. It should create file **Exercise2/src/Ex2.hs**.
- Enter Exercise2 and give the command **stack install**, which will install an executable called **ex2**. Running ex2 will result in a **Prelude.undefined** error.
- Your task is to edit **Ex2.hs** (only) so that it produces the correct output - this is defined in the comments in **Ex2.hs**.
  You can re-compile your code using **stack install**, and the re-run **ex2**.
- Your submission is ONLY the file **Ex2.hs** (as is, do not compress/tar it in any way)

Unlike Exercise 1, this exercise asks you to write five functions.

- Functions **f1**, **f2** and **f3** are exercises in basic list processing.
- Function **f4** is a more complex exercise involving an imaginary fault-prone mid-20th century list handling processing unit (it is described more fully below)
- Function **f5** is a function that keeps repeating the use of **f4** on an input as long as it is possible.

Function **f4** --- see week 4 tutorial slides

Clarifications/Hints

- Taking every 3rd element of list [1..16] returns [3,6,9,12,15]
- If the **f2** list is too short it should return 0
- For **f2** you can use **add** or **(+)** to add the numbers - my tests will still work
- If the **f3** list is too short it should return 1
- For **f3** you can use **mul** or **(*)** to multiply the numbers - my tests will still work
- **f4** will initially skip any number that is not an opcode
- **f4 []** will return (0,[])
- In **f4**, if no following numbers are found, then it should return (0,[]) for **add**, and (1,[]) for **mul**.
- In **f4**, if the list ends midway during processing an opcode then the result so far is returned.
- In **f4**, if the **stop@** number is the same as the number specified to replace **Nothing**, then when a **Nothing** is encountered, its value is added/multiplied as appropriate, and it is ***not*** treated as a stop number.
- In **f4** if a **Nothing** is **skip**ped for a **fixed N** opcode, that Nothing does **not** contribute to the count.
- When building a list for test purposes, remember a value of type **Maybe a** needs to be built/specified using the two data constructors of the **Maybe** type.