Introduction to Computer Science

Assignment 11
– xTra marks –

Constructor University,
Bremen, Germany

May 1, 2024

- This assignment counts **15 marks** in total!
- Marks are only awarded for producing the correct solutions and justifying how you arrived at them.
- Please write legibly. Illegible answers will NOT be marked.

Question 1

Using the simple assembly language presented in the lectures (slide 216),

1. Write a program that computes the summation:

$$\sum_{i=1}^{5} i$$

   At the end of the execution the result must be in the accumulator. For answering this question you need to present the assembly code, the machine code is not required. Include a comment at each line of your code describing what occurs at each step.

   [5 marks]

2. Present a trace of the execution of your program using the same notation of the lecture notes, i.e. showing the initial memory content, the machine instructions executed (pc = program counter, ir = instruction register, acc = accumulator), and the final memory content.

   [4 marks]

Question 2

A *recurrence* is an equation according to which the n^{th} term of a sequence of numbers is obtained from some computation of the previous terms. In other words, each element of the sequence is a function of previous elements of the same sequence. One example is the famous *Fibonacci sequence*, which gets defined by:

$$
\begin{aligned}
F_0 &= 0 \\
F_1 &= 1 \\
F_{n+2} &= F_{n+1} + F_n, \quad \text{for } n \geq 0
\end{aligned}
$$

In this manner, the Fibonacci sequence starting from 0 and *followed by* 1 goes like this: $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$, where the next term is computed from the addition of the previous two numbers.

A second example is the *Lucas sequence* which is defined by:

$$
\begin{aligned}
L_0 &= 2 \\
L_1 &= 1 \\
L_{n+2} &= L_{n+1} + L_n, \quad \text{for } n \geq 0
\end{aligned}
$$

The Lucas sequence goes like this: $2, 1, 3, 4, 7, 11, 18, 29, \ldots$

So, the Fibonacci and Lucas sequences satisfy the same linear recurrence but with different initial values. In this exercise we, are going to implement such recurrence using our data-flow (stream-based) programming framework, see file: `stream.hs` part of which is reproduced here for convenience:

```
data Stream a = a :| Stream a

instance (Show a) => Show (Stream a) where
  -- show :: Show a => Stream a -> String
  show (x :| xs) = show x ++ " : " ++ show xs

lift0 :: a -> Stream a
lift0 x = x :| lift0 x

lift1 :: (a -> a) -> (Stream a -> Stream a)
lift1 op (x :| xs) = (op x) :| lift1 op xs
```

```
lift2 :: (a -> a -> a) -> Stream a -> Stream a -> Stream a
lift2 op (x :| xs) (y :| ys) = (x `op` y) :| lift2 op xs ys

lift3 :: (a -> a -> a -> a) -> Stream a -> Stream a -> Stream a -> Stream a
lift3 fn (x :| xs) (y :| ys) (z :| zs) = (fn x y z) :| lift3 fn xs ys zs

-- followed by
fby :: a -> Stream a -> Stream a
fby x ys = x :| ys
```

1. Lift addition of `Integer` up to `Stream Integer`, (represented by `|+|`) as an infix left-associative operator with precedence 6:

   ```
   -- lifted addition
   infixl 6 |+|
   (|+|) :: Stream Integer -> Stream Integer -> Stream Integer
   ```

   [1 mark]

2. Implement the data-flow stream-based function:

   ```
   -- recurrence sequence
   recurrence :: Integer -> Integer -> Stream Integer
   ```

   where the first and second parameters (of type `Integer`) correspond, respectively, to the initial values of the recurrence.
   Thus, for example:

   ```
   > let fibonacci = recurrence 0 1
   > let lucas = recurrence 2 1
   > giveme 10 fibonacci
   [0,1,1,2,3,5,8,13,21,34]
   > giveme 10 lucas
   [2,1,3,4,7,11,18,29,47,76]
   ```

   Note that function `giveme` is already defined in `stream.hs`, for convenience is reproduced here:

   ```
   giveme :: Int -> Stream a -> [a]
   giveme 0 _          = []
   giveme n (x :| xs)  = x : (giveme (n-1) xs)
   ```

   [5 marks]