Brian Simpson

Professor Vu

COP 4020

03/07/21

HW 5

1. Define an action `adder :: IO ()` that reads a given number of integers from the keyboard, one per line, and displays their sum.

For example:

```
> adder
How many numbers? 5
1
3
5
7
9
The total is 25
```

Hint: start by defining an auxiliary function, `adder’`, that takes the current total and how many numbers remain to be read as arguments. You will also likely need to use the library functions read and show.

```
adder:: IO ()

adder = do

   putStr "How many numbers?"

   x <- adderInteger

   adder' x 0 0
```

Brian Simpson

Professor Vu

COP 4020

03/07/21

```
adder' :: Int -> Int -> Int -> IO ()
adder' x y z =  if y < x                              --haskell equivalent to a while loop
                              then do
                               a <- adderInteger
                               adder' x (y+1) (z+a)
                 else do
                  putStr "The total is "
                  putStrLn (show(z))


adderInteger :: IO Int
adderInteger = do
                                  x <- getLine
                                  return (read x)
```

I chose to implement this across three different functions. The entrypoint function, adder, simply asks the user how many integers they would like to sum up and then calls the third function, adderInteger. adderInteger just takes this single integer of input from the keyboard and returns it to adder. Adder then finally calls the last function, adder' with the total number of integers to be added up as x, and 0 as both y and z. Most of the work is done in adder'. If y is still less than x we know the user has not finished inputting all of their numbers yet so we take another integer of input from them, and then recursively call adder' without ever changing the value of x, incrementing the value of y by 1 each time to track how many numbers have been entered, and using z to keep track of the sum to display it at the end in the else block when y is no longer less than x.

Brian Simpson

Professor Vu

COP 4020

03/07/21

2. Define

I. Define a program
`fibs :: [Integer]`
that generates the infinite Fibonacci sequence
`[0,1,1,2,3,5,8,13,21,34, …`
using the following procedure:

a) The first two numbers are 0 and 1;
b) The next is the sum of the previous two;
c) Return to step 2.

Hint: make use of the library functions zip and tail. Note that numbers in the Fibonacci sequence quickly become large, hence the use of the type Integer of arbitrary-precision integers above.

fibs :: [Integer]

fibs = 0 : 1 : zipWith (+) fibs (tail fibs)

Since 0 and 1 are the base case we default to concatenating them together, followed by recursively calling fibs in a manner that results in the first element of the list being dropped (due to the tail call), and then what remains is zipped together in a manner that causes the summation to proceed down the line infinitely and never stop adding the previous two numbers together to produce a new third output which is then zipped with the next output that is produced until it is done being used for calculations, when it is then dropped from the recursive call through the use of tail.

II. Define a function
`fib :: Int -> Integer`
that calculates the nth Fibonacci number.

fib :: Int -> Integer

fib x = if x == 0 then 0 else if x == 1 then 1 else fib (x-1) + fib (x-2)

I chose to implement this version of fib with if statements instead of using zip or tail like in the previous implementation. Even though I could have simply copied the same strategy

Brian Simpson

Professor Vu

COP 4020

03/07/21

that I used in my adder solution and created a helper function with a y variable in order to track when the nth Fibonacci number is reached, I decided that this solution was quicker and more simple to implement. By simply returning 0 or 1 as the base cases when the value reaches those points, and then calling fib recursively for 1 and 2 numbers back in the sequence respectively (since the next number in the sequence is always the sum of the previous two), we can easily calculate the nth Fibonacci number without needing to make use of those library functions.