Computer Science 54

Assignment 3

Sunday, September 25 @ 11:59pm

https://xkcd.com/859/

This assignment gives you additional practice with lists and recursion.

Partners On this assignment you may (and I'd encourage you to) work with a partner. If you choose to, you must both be there anytime you're working on the assignment.

Submission When you're all done, run the check file. If all is well, submit your assignment via Gradescope. If you worked with a partner, make sure that both names are at the top of the file. Only one person needs to submit, but make sure you add the other person when you submit.

I Keeping up with the Cartesians

"I recurse, therefore I am."

René Descarte is known for, among other things, developing analytic geometry. He used ordered pairs of numbers to specify points in the plane. We now use the phrase *cartesian product* to mean a collection of ordered pairs in which the components are taken from the specified sets. If A and B are sets, we have

$$A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}.$$

Instead of sets we will use lists, and we will write a function `cartesian` that creates a list of ordered pairs from the elements of two lists. Our function will recurse over both lists, but we will write it in two parts—one for each list.

1. **[2 points]**

a. To start, write a function `precart` which takes an *element* and a list, and forms the list of pairs in which the first component is the given element. For example,`precart 1 [2,3,4]` returns `[(1,2),(1,3),(1,4)]`.

```
precart : 'a -> 'b list -> ('a * 'b) list
```

As shown in the type signature, the components do not have to be integers. In fact, the elements can come from lists of different types.

b. Now use `precart` to create a function `cartesian` that computes the full cartesian product of two lists. For example,

`cartesian [1,2] [3,4,5]` returns
`[(1,3),(1,4),(1,5),(2,3),(2,4),(2,5)]`.

```
cartesian : 'a list -> 'b list -> ('a * 'b) list
```

II Fun with Arithmetic

2. **[2 points]** We can think of a natural number as being represented by a list of digits, with the least significant digit at the front of the list. For example, 47 is represented by `[7,4]`.

Notice that the representation is not unique; 47 is represented by `[7,4]` and also `[7,4,0,0]`. What is the shortest representation for zero?

a. Declare an inverse function `fromDigitList` to convert from lists of digits to non-negative integers. Declare an exception `BadDigit` and raise it when the function finds a number that is not in the range from 0 through 9.

```
fromDigitList : int list -> int
```

Hint: Horner's method is a way to evaluate polynomials efficiently. We can view the multi-digit number $d_{n-1} \ldots d_2 d_1 d_0$, expressed as $[d_0, d_1, d_2, \ldots, d_{n-1}]$ in our list representation, as a polynomial evaluated at $X = 10$.

$$\begin{aligned} d_{n-1} \ldots d_2 d_1 d_0 &= d_0 + d_1 10 + d_2 10^2 + \ldots + d_{n-1} 10^{n-1} \\ &= d_0 + 10 \cdot \left(d_1 + d_2 10 + \ldots + d_{n-1} 10^{n-2}\right) \end{aligned}$$

For example, 342 has $d_2 = 3$, $d_1 = 4$ and $d_0 = 2$, and we can write $342 = 2 + 10 * (4 + 3 * 10)$.

b. Write a function `toDigitList` to convert from non-negative integers to lists of digits. Declare an exception `NegativeNumber` and raise it when the argument is negative.

```
toDigitList : int -> int list
```

III The Luhn Algorithm

The Luhn Algorithm is a way of verifying the validity of credit card and other identifying numbers. Here are the steps to verify a number:

- Write the number as a sequence of decimal digits in the usual left-to-right order. Starting at the right, double every second value. (That is, start by doubling the next-to-last digit.)
- Add the digits of the resulting sequence.
- Compute the remainder when the sum is divided by 10.
- The number is valid if the remainder is 0.

For example, here is what we would do if we start with the number 13573:

separate the digits	1	3	5	7	3
double every other one	1	6	5	14	3
add the digits	1	6	5	1 + 4	3

compute the total 1 + 6 + 5 + 5 + 3 = 20

The number is valid because 20 is a multiple of 10. Usually, the last (rightmost) digit of an account number is the *check digit.* It is chosen to make the Luhn algorithm work out correctly. In the above example, 1357 is the "real" account number and 3 is the check digit.

Most credit card numbers are too long to be represented by SML integers, so we will present the numbers as strings of digits. The functions below permit you to translate back and forth between digits (of type `char`) and integers (of type `int`). Copy them into your file.

Security warning: Be cautious with real credit card numbers and avoid typing them into files. There are websites like `http://www.getcreditcardnumbers.com/` from which you can get "valid looking" numbers for testing.

Life warning: Do not try to buy anything with a card number from one of these sites.

```
exception NotADigit;

fun digitToInt c =
        if Char.isDigit c then
           ord c - ord #"0"
        else
           raise NotADigit;

fun intToDigit k =
        if 0 <= k andalso k < 10 then
           chr (k + ord #"0")
        else
           raise NotADigit;
```

3. **[2 points]** Getting it started

a. The first task is to write a function `stringToIntList` that takes a string and produces a list of the corresponding digit values. Because the Luhn algorithm works from the right of the string, it is most convenient to write the list in reverse order. For example, `stringToIntList "13573"` returns `[3,7,5,3,1]`.

Suggestion: Use `map` and `explode`.

```
stringToIntList : string -> int list
```

b. Write a function `doubleDigitSum` that takes an integer k between 0 and 9, and returns the sum of the digits in $2k$. Your function must give the correct sums for arguments between 0 and 9 inclusive; the results for other arguments will never be used.

Hint: The sum of the digits in $2k$ is either $2k$ or $2k - 9$. Why?

```
doubleDigitSum : int -> int
```

4. **[2 points]** Almost there

Write a function `luhnSum` that takes a list of digit values and calculates the sum of the digits after every second element of the list has been doubled. For example, as computed above, `luhnSum [3,7,5,3,1]` returns `20`.

Several approaches are possible. One is to write a recursive function that processes two list elements at a time. Another approach is to extract two different lists, map `doubleDigitSum` over one of them, and then add the elements of both lists.

```
luhnSum : int list -> int
```

5. **[2 points]** There it is

a. Put everything together and write a function `luhnCheck` that determines if the account number, presented as a string, is valid. The function will simply be a combination of functions that you have already written.

```
luhnCheck : string -> bool
```

b. Finally, the last step is to calculate the check digit. Write a function `calculateCheckDigit` that takes an account number and calculates the check digit for that number. For example, `calculateCheckDigit "1357"` would give us the check digit 3. The final account number would then be the actual account number with this digit at the end. One way to calculate the check digit is to append a zero on the right of the account number and compute the Luhn sum.

```
calculateCheckDigit : string -> int
```

c. The Luhn algorithm will detect any single-digit error, and it will detect *most* transpositions of adjacent digits. Find a valid account number with a pair of adjacent digits d and e for which d and e are different and the number remains valid when d and e are swapped. Put your answer in a comment.

IV Substitution Ciphers, Revisited

On Assignment 2 we wrote a function to encrypt using Caesar ciphers, which employ a simple shift of the alphabet. We now turn to a slightly more sophisticated system.

Suppose you and your fellow spy want to exchange secret messages. You agree on a pangram, a sentence that contains all the letters of the alphabet, to use as a key. The order of the letters in the pangram will specify the letter-for-letter substitution you will use. Here are a few examples of pangrams.

```
A QUICK BROWN FOX JUMPS OVER THE LAZY DOG
CRAZY FREDRICK BOUGHT MANY VERY EXQUISITE OPAL JEWELS
GRUMPY WIZARDS MAKE A TOXIC BREW FOR THE JOVIAL QUEEN
```

Suppose you chose the third of these. Then to encode your message, you would replace `A` with `G`, replace `B` with `R`, and so forth. Pangrams that use each letter only once are rare and hard to remember, so we have to eliminate duplicate letters. For the third "grumpy" pangram, we would have the following translation.

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ␣
GRUMPY␣WIZADSKETOXCBFHJVLQN
```

6. **[2 points]** `uniquify`?

a. Write a function `keepFirst` that removes duplicates from a list, keeping the *first* occurrence of each element.

```
keepFirst : ''a list -> ''a list
```

The function `uniquify` that was presented in class retains the *last* occurrence of each element. For this function, consider using an accumulator, like we did with `revAux`.

b. Use `keepFirst` to write a function `subst` that takes a pangram and returns a list of pairs of characters. Each pair will specify one substitution in the cipher. For example, using the "grumpy" pangram,

`subst "GRUMPY...QUEEN"` would return
`[(#"A",#"G"),(#"B",#"R"),...,(#"␣",#"N")]`.

Get used to using some of the tools from the SML library, such as `ListPair.zip`.

You may assume that the given pangram is presented as a string with only uppercase letters and spaces.

```
subst : string -> (char * char) list
```

7. **[2 points]** Substitution > Caesar

We are now ready to assemble the parts of a substitution cipher. Your function will resemble the one for `caesar` from Assignment 2. Feel free to borrow code from Assignment 2. As in Assignment 2, build up your solution step by step using a `let` expression.

a. Write a function `substEncipher` which takes a pair consisting of two strings, a pangram key and a message, and returns the encrypted string.

```
substEncipher : string * string -> string
```

b. Write the corresponding `substDecipher` which takes a pangram key and an encoded string and returns the plaintext message. The pangram key will be the one used to encipher. All you have to do is invert the substitution.

```
substDecipher : string * string -> string
```

Once your functions are working properly, you will be able to encrypt a message with the key and then decrypt it to get the original message back.

V ***Extra credit:*** *Making Change*

This is the famous *Change Problem,* a tradition in the CS department since 2003.

[0.5 points] Given a list of coin denominations and an amount of money, we want to list all the ways to make change in that amount using the specified coins. For example, there are two ways to make seven cents from nickels and pennies:

```
[ [1,1,1,1,1,1,1], [5,1,1] ]
```

Write an SML function `change` that takes a list of denominations and an amount and produces a list of all the ways to make change in that amount. You may assume that the amount is not negative and that the coin denominations are all positive.

```
change : int list -> int -> int list list
```

This exercise is intended to be an example of a *nonstandard* use of list recursion. Be guided by the following:

- Think about the base cases. What is the result when the amount is 0? or negative? What is the result when the list of coins is empty?
- Here is a strategy to reduce a general case to simpler ones: Let a be the amount and d be the first denomination on the list. There are two possibilities: either you do not use d at all and you have to make change in the amount a from the other denominations, or else you use d once and you have to finish the job by making change in the amount of $a - d$, using the entire list including d.

  A "use-it-or-lose-it" argument.
- The order of coins in a single possibility is not relevant. "Two pennies and a nickel" is the same as "a nickel and two pennies." If you follow the previous suggestion, you will automatically avoid duplicate possibilities. When you are finished, observe that the coins appear in a possibility in the same order—perhaps with repetitions or omissions—as in the original list of coins.
- The possibilities themselves may appear in any order, depending on how you implement the strategy above.
- Use the type information as a guide to constructing the final result. Remember that `change` returns a list of lists.