Program Design & Data Structures (Course 1DL201)
Uppsala University Autumn 2013/Spring 2014
Homework Assignment 3: Cryptography

Prepared by Dave Clarke

Lab	Thursday January 30, 2014
Submission Deadline	**18:00:00, Friday 7 February, 2014**
Lesson	Thursday 20 February, 2014
Resubmission Deadline	Friday 28 February, 2014

Cryptography

Originally used exclusively by spies, governments and in times of war, cryptography has become embedded solidly within our daily lives. Online banking and shopping and accessing websites like Facebook rely on cryptography to operate securely. Cryptography, at least historically, is based on the encryption and decryption of messages. *Encryption* is the process of converting ordinary information (called *plaintext*) into unintelligible text (called *ciphertext*). Decryption is the reverse, converting the unintelligible ciphertext back to plaintext. A *cipher* is a pair of algorithms that perform the encryption and decryption. The detailed operation of a cipher is controlled both by the algorithm and in each instance by a “key”. This is a secret (ideally known only to the communicants), usually short string of characters, which is needed to decrypt the ciphertext.[1]

The *Cards* cipher was designed to be the first truly secure hand cipher, that is, an encryption scheme that can be performed by hand. *Cards* requires only a deck of cards for the encryption and decryption of messages. While it is true that *Cards* is easily computed by hand, using a computer is much quicker and easier. Your task will be to write a Standard ML program that helps compute the *Cards* cipher.

Note: this assignment will be based on the English language alphabet. There is no requirement or guarantee that Swedish characters ÅÄÖåäö will be treated correctly, so avoid them in test cases.

Encryption

The *Cards* encryption algorithm takes the following steps:

1. Preprocessing. Discard all non-alphabetic characters and convert the remaining characters to uppercase. Split the message into five character groups, using Xs to pad the last group, if needed.

   For example, starting with the message “Live long and prosper!”, we would now have:

[1]Adapted from `http://en.wikipedia.org/wiki/Cryptography`.

```
LIVEL ONGAN DPROS PERXX
```

2. Use *Cards* to generate a keystream letter for each letter in the message—this is detailed in Section *The Keystream*. For sake of example, let's assume the result is:

   ```
   DWJXH YRFDG TMSHP UURXJ
   ```

3. Convert the message from step 1 into numbers, using $A = 1$, $B = 2$, and so on:

   ```
   12 9 22 5 12   15 14 7 1 14   4 16 18 15 19   16 5 18 24 24
   ```

4. Convert the keystream letters from step 2 using the same method:

   ```
   4 23 10 24 8   25 18 6 4 7   20 13 19 8 16   21 21 18 24 10
   ```

5. Add the message numbers from step 3 pairwise to the keystream numbers from step 4, subtracting 26 from the result if it is greater than 26. For example, $6 + 10 = 16$ as expected, but $25 + 5 = 4 \quad (= 30 - 26)$:

   ```
   16 6 6 3 20   14 6 13 5 21   24 3 11 23 9   11 26 10 22 8
   ```

6. Convert the numbers from step 5 back to letters:

   ```
   PFFCT NFMEU XCKWI KZJVH
   ```

At this point, the message has been encrypted. We now describe how to reverse the process.

Decryption

Decrypting with *Cards* is even easier. Here are the steps, based on our encrypted example `PFFCT NFMEU XCKWI KZJVH`:

1. Use *Cards* to generate a keystream of the same length as the message to be decrypted. This is the same as the process in step 2 above, ensuring that the sender and receiver will use the same letters:

   ```
   DWJXH YRFDG TMSHP UURXJ
   ```

2. Convert the message to be decoded into numbers:

   ```
   16 6 6 3 20   14 6 13 5 21   24 3 11 23 9   11 26 10 22 8
   ```

3. Convert the keystream letters from step 1 into numbers:

   ```
   4 23 10 24 8   25 18 6 4 7   20 13 19 8 16   21 21 18 24 10
   ```

4. Subtract the keystream numbers (step 3) from the message numbers (step 2) pairwise, adding 26 if the number drops below 1. For example, $22 - 1 = 21$ as expected, but $1 - 22 = 5$ $(27 - 22)$.

```
12 9 22 5 12   15 14 7 1 14   4 16 18 15 19   16 5 18 24 24
```

5. Convert the numbers from step 4 back to letters:

```
LIVEL ONGAN DPROS PERXX
```

That's all there is to decrypting messages. All one needs to do now is insert the word breaks at the appropriate places and drop spurious Xs, obtaining `LIVE LONG AND PROSPER` (*This step is not a part of the assignment.*)

Finally, let's look at the missing piece of the puzzle, generating the keystream letters.

The Keystream

First, let's talk a little about the deck of cards. *Cards* needs a full deck of 52 cards with the two jokers. The two jokers need to be visually distinct. Refer to them here as A and B.

Some of the steps involve assigning a **value** to the cards. In those cases, use the card's face value as a base: ace is 1, two is 2, ..., ten is 10, jack is 11, queen is 12, and king is 13. Then modify the base by the following ordering of suits: clubs (♣) is just the base value, diamonds (♢) is the base value + 13, hearts (♡) is the base value + 26, and spades (♠) is the base value + 39. Either joker has a value of 53.

When the cards must represent a **letter**, club and diamond values are taken to be the same number of the letter (1 to 26), as are heart and spade values after subtracting 26 from their value (27 to 52 drop to 1 to 26). Jokers are never used as letters.

	A♣	2♣	···	K♣	A♢	2♢	···	K♢	A♡	2♡	···	K♡	A♠	2♠	···	K♠
Value	1	2	···	13	14	15	···	26	27	28	···	39	40	41	···	52
Letter	A	B	···	M	N	O	···	Z	A	B	···	M	N	O	···	Z

Now let's put all that to use:

1. Key the deck. This is the critical step in the actual operation of the cipher and the heart of its security. There are many methods for going about this, such as shuffling a deck and then arranging the receiving deck in the same order or tracking a bridge column in a paper and using that to order the cards. *This step will not be a part of the assignment.*

   Because we want to test our answers, though, we'll use an unkeyed deck, cards in order of value. That is, from, top to bottom, we'll always start with the following deck:

   ```
   Ace of clubs
   ... to ...
   King of clubs
   Ace of diamonds
   ... to ...
   King of diamonds
   Ace of hearts
   ... to ...
   King of hearts
   Ace of spades
   ```

```
... to ...
King of spades
"A" joker
"B" joker
```

2. Move joker A down one card. If the joker is at the bottom of the deck, move it to just below the first card. (Consider the deck to be circular.) The first time we do this with an unkeyed deck, it will go from

   ```
   1 2 3 ... 52 A B
   ```

   to

   ```
   1 2 3 ... 52 B A    (*)
   ```

   (Repeating the move will produce deck `1 A 2 3 ... 52 B`.)

3. Move joker B down two cards. If the joker is the bottom card, move it just below the second card, or if the joker is just below the bottom card, move it below the top card. (Again, consider the deck to be circular.) This changes the deck marked `(*)` to the following:

   ```
   1 B 2 3 ... 52 A    (**)
   ```

   (Repeating the move would produce deck `1 2 3 B ... 52 A`.)

4. Perform a triple cut around the two jokers—that is, split the deck into three chunks around the two cards, and then swap the top and bottom chunk: all cards above the top joker move to below the bottom joker and vice versa. The jokers and the cards between do not move. Starting from the deck marked `(**)`, this gives us the following:

   ```
   B 2 3 ... 52 A 1   (***)
   ```

   (A triple cut on deck `1 2 3 A 4 5 ... 50 B 51 52` would result in `51 52 A 4 5 ... 50 B 1 2 3`)

5. Perform a count cut using the value of the bottom card. Count the bottom card's value in cards off the top of the deck, and move them to just above the bottom card. This changes our deck `(***)` to the following:

   ```
   2 3 ... 52 A B 1   (the 1 tells us to move just the B)    (****)
   ```

   (Deck `51 52 A 4 5 ... 50 B 1 2 3` would become `4 5 ... 50 B 1 2 51 52 A 3`.)

6. Find the output letter. Convert the top card to its value and count down that many cards from the top of the deck, with the top card itself being card 1. Look at the card immediately *after* your count and convert it to a letter. This is the next letter in the key stream. If the output card is a joker, no letter is generated this step. This step does not alter the deck. For our example `(****)`, the output letter is as follows:

   ```
   D  (the 2 tells us to count down to the 4, which is a D)
   ```

(For `4 5 ... 50 B 1 2 51 52 A 3` the output letter is, presumably, H (the 8th letter)—understand why!)

7. Return to step 2, if more letters are needed.

For the sake of testing, the first ten output letters for an unkeyed deck are as follows:

```
D (4)
W (49)
J (10)
Skip Joker (53)
X (24)
H (8)
Y (51)
R (44)
F (6)
D (4)
G (33)
```

That's all there is to *Cards*—the explanation is far longer than your code should be.

Some Test Cases

Although a more extensive test suite will be provided, here are a couple of messages for you to test your work with. You'll know when you have them right:

```
CLEPK HHNIY CFPWH CDFEH
GOINB NYXEW BLKCB ZNGIV
```

Work to be done

The goal of the assignment is to implement the *Cards* cipher. This includes preprocessing, encryption and decryption, and the generation of the keystream. You will also analyse the complexity of your functions.

Implement the following functions in a file called `crypto.sml`, making sure that they pass the training test cases in file `assignment3-training.sml` (from student portal). The test cases should **not** be included in the file with your functions.

1. Function `preprocess : string -> char list list` performs the preprocessing step, described above in section *Encryption*, Step 1. The result should consist of lists of length exactly 5 containing only letters A-Z. For example,

```
> preprocess "Live long and prosper!";
val it =
   [[#"L", #"I", #"V", #"E", #"L"], [#"O", #"N", #"G", #"A", #"N"],
    [#"D", #"P", #"R", #"O", #"S"], [#"P", #"E", #"R", #"X", #"X"]]:
   char list list
```

Here `#"D"` is Standard ML's way of writing character D.

2. Function `encrypt : char list list -> char list list` encrypts the input according to the scheme defined above in section *Encryption*, starting from Step 2. The input and the result should consist of lists of length exactly 5 containing only letters A-Z.

3. Function `decrypt : char list list -> char list list` decrypts according to the scheme defined above in section *Decryption*. The input and the result should consist of lists of length exactly 5 containing only letters A-Z.

4. Function `keystream : int -> char list` given an integer $n \geq 0$, return the first n elements of the key stream, as described in section *The Keystream*.

Note that for all inputs `s` satisfying the precondition to `encrypt`, it must be the case that `decrypt (encrypt s) = s`.

In a separate report in PDF format, give explicit reasoning to establish the worst case complexity of these 4 functions. For the purposes of complexity analysis, assume that the input (after preprocessing) has n characters and that there are m cards in the deck, rather than the fixed number 52+2. Thus the run-time complexity will be a function of *two* variables.

Useful Functions

The following two functions will be provide in file `helper.sml` Use these to help debug your own solutions.

- `format : char list list -> string` makes the list of list of characters easier to read.
- `fakekeystream : int -> char list`. This dummy function generates some of the keystream so that you can quickly start doing encryption and decryption. This function will only be helpful for short messages.

In addition, the following Standard ML library functions may come in handy:

- The function `explode : string -> char list` which converts a string into a list of the characters making up the string (See `http://www.standardml.org/Basis/string.html`). Call using `String.explode`.
- The function `implode : char list -> string` which converts a list of characters into a string. Call using `String.implode`.
- Functions on characters (See `http://www.standardml.org/Basis/char.html`).
- Functions on lists (See `http://www.standardml.org/Basis/char.html` and `http://www.standardml.org/Basis/list.html`).

Grading

Your solution will be graded with 0 to 100 points in the following way:

1. If your solution was submitted by the deadline, your program loads under Poly/ML version 5.5.1 and it passes all of the training test cases provided, then you get 20 points; otherwise (including when no solution was submitted by the submission deadline), you get a U grade for this homework.

   Hint: Test your submitted code in a freshly started ML interpreter in order to ensure your code does not work because of some old data or functions lying around in the interpreter.

2. Your program will be run on an unspecified number of orthogonal grading test cases, which cover both valid and invalid problem and solution combinations.

   For each fully correct test result, you get a suitable amount of points, the total being 40 points. We reserve the right to run these tests automatically, so be careful to match exactly the imposed file names, function names, function types and argument orders.

3. Your program will be graded for style and comments. The following criteria will be used:

   - suitable breakdown of solution into auxiliary/helper functions
   - comments describing the purpose of each function, following the coding convention, including function specifications, representation conventions and invariants, and recursion variations)
   - code readability
   - appropriate use of higher order functions.

   This covers 20 points.

4. Your complexity analysis will be graded for correctness of results and explicitness of reasoning. This covers the remaining 20 point

Grades out of 100 are converted to the usual scale U, 3, 4, 5, in the following fashion:

- A grade of less than 30 results in a U.
- A grade between 30 and 69 means that the students are required to attend the lesson discussing the assignment and to subsequently resubmit the assignment. After resubmission, the assignment will be regraded and a grade of 70 or above is required to obtain a 3.
- A grade between 70 and 84 (inclusive) results in a 4.
- A grade of 85 or higher results in a 5.

Modalities

The assignment will be conducted in groups of 2. Groups will be assigned via the student portal. Assignments must be submitted via the student portal. Only one solution per group needs to be submitted.

We assume that by submitting a solution you are certifying that it is solely the work of your group, except where explicitly attributed otherwise. We reserve the right to use plagiarism detection tools and point out that they are extremely powerful.