EDITORIAL TO PRACTICE SHEET: REGULAR EXPRESSION

(some problems are a little hard, but you can do it)

A. Write Regular Expressions for the following languages (Assume that Σ = {0, 1}) -

1. Strings that end in three consecutive 1's.
The answer to this is pretty straight forward. We will have three 1s at the end, and preceding that, there can be any number of zeros and ones.
ANS: (1 | 0)* 1 1 1

2. Strings that have at least one 1.
Any number of zeros and/or ones can appear before and/or after that specific 1.
ANS: (1 | 0)* 1 (1 | 0)*
Alternate ANS: 0* 1 (1 | 0)*

3. Strings that have at most one 1.
There can be one 1 in the string, but there can also be no 1 at all.
ANS: 0* (1 | e) 0* [e means epsilon]
Alternate ANS: 0* 1 0* | 0*

4. Strings that contain the substring "101".
There will be 101 in the string. Anything can come before and/or after that 101.
ANS: (1 | 0)* 1 0 1 (1 | 0)*

5. Strings that do not contain the substring "11".
(The solution to this question is the same as the solution to number 6, as no "11" substring means no consecutive 1s.)

6. Strings that do not have consecutive 1s.
At any position of the string, we can choose a 0 freely, but if we choose a 1, then it must be followed by a 0. Unless the 1 is the last symbol of the string. In that case, the 1 is followed by nothing.
ANS: (0 | 1 0)* (1 | e)
[the (1 | e) part is to cover the case when the last symbol of the string is 1]

7. Strings that have neither consecutive 1s, nor consecutive 0s.
As each 1 must be followed by a 0 and each 0 must be followed by a 1 (except for the last symbol of the string), we have the following expression:
ANS: ((1 0)* (1 | e)) | ((0 1)* (0 | e))
Alternate ANS: (0 | e) (1 0)* (1 | e)

8. Strings that may have consecutive 1s, or consecutive 0s, but not both.
From number 6, we have the expression for strings without consecutive 1s. Similarly, the expression for strings without consecutive 0s can be written as - ***(1 | 0 1)* (0 | e).***
If we OR these two expressions together, we get the expression for strings that either does not

have consecutive 1s, or does not have consecutive 0s. Which is essentially the answer for Q8.
ANS: (0 | 1 0)* (1 | e) | (1 | 0 1)* (0 | e)

9. Strings in which the number of 0s is odd.
Let's first construct the expression where the number of 0s is even. As the zeros will come in pair, and there can be any number of 1s in the beginning/middle/end of the pair, we get the expression - ***(1* 0 1* 0 1*)****
Now if we add one more 0 to this expression for string with even number of zeros, we will get the expression for strings with odd number of zeros. But again, you have to remember that any number of 1s can appear anywhere.
ANS: (1* 0 1* 0 1*)* 1* 0 1*
Alternate ANS: (1* 0 1* 0)* 1* 0 1*

10. Strings in which the number of 0s is divisible by four.
With similar logic, we have the expression for strings where number of 0s come in groups of four, i.e, is divisible by four: (1 0 1* 0 1* 0 1* 0 1*)* But we also have to consider the case where there are no 0s in the string, but only 1s.*
ANS: (1* 0 1* 0 1* 0 1* 0 1*)* | 1*
Alternate ANS: (1* 0 1* 0 1* 0 1* 0 1*)* 1*
Alternate ANS: (1 | 0 1* 0 1* 0 1* 0)* [Carefully examine and try to understand this solution]

B. Write a regular expression for valid email addresses. You have the following information –

Email addresses have two parts: the user ID (the part before the '@' symbol), and the domain name (the part after the '@' symbol). Ex: lex.luthor@gmail.com
[In here, 'lex.luthor' is the user ID, and gmail.com is the domain name.]

● *There are only three domains: gmail.com, yahoo.com, and bracu.ac.bd*
● *All user IDs are made of lower and/or upper class English Letters. A user ID may contain digits, but only if it also contains at least one letter.*
● *Gmail and Yahoo require users to have user IDs that are of length 5 or more. BRACU allows users to have user IDs of length 1 or more.*
● *BRACU does not allow users to have user IDs that start with digits. Google or Yahoo has no such restriction.*

Let's first define some charachter classes: L is the class of all letters of the alphabet, D is the class of all digits, and V is the unified class of letters and digits.
L = [a-zA-Z]
D = [0-9]
V = L U D

As they have different sets of requirements which do not match each other, the expression for bracu domain and yahoo/gmail domain must be different.

Expression for bracu domain:
We know the first symbol must come from the set L, and any number of letters and/or digits can appear after that. Hence the expression is -
L V * @ b r a c u . a c . b d

Expression for yahoo/gmail domain:
In the user ID, there must be at least one letter somewhere, and the length can be five or more. We don't know at which position the letter will appear, so we must consider all cases.

If there is a letter at first position: $\boldsymbol{LV^4V^*}$
If the first symbol is not a letter, but the second one is: $\boldsymbol{DLV^3V^*}$
Similarly, for letters at the third/fourth position, we have $\boldsymbol{D^2LV^2V^*}$ *and* $\boldsymbol{D^3LVV^*}$
Lastly, if the first 4 symbols of the user ID are digits, and there is a letter somewhere after that, we get the expression:
$\boldsymbol{D^4V^*LV^*}$

So the combined answer is:
$\boldsymbol{(LV^*@bracu.ac.bd)|((LV^4V^*|DLV^3V^*|D^2LV^2V^*|D^3LVV^*|D^4V^*LV^*)@(yahoo|gmail).com)}$

C. Write a regular expression for the language of all possible complex numbers.

[Example of some complex numbers: 4+3i, -406-45i, +10+i, -5+1i etc]

Let D be the set of all ten digits, D = {0,...,9}
This is pretty straight forward, we may or may not have a plus or minus sign at the beginning. Then there will be string of integers of length at least one, D^+. *Then there will be either a plus sign or a minus sign followed by another string of integers, but this time, even no integer is also a case, so* ***D**** *it is. After that, there will be an i.*
This is how the expression should look like:

Hence, the answer is:

$$(+|-|e)D^+(+|-)D^*i$$

D. Find the shortest string that IS in the language represented by the regular expression:

$\boldsymbol{(a^*b|ba^*|ab^*|ba^*|(a^*b^*))(c^*d|cd^*|dc^*|d^*c|(c^*d^*))((a^4|c^4)(b^4|d^4)^*)}$
As are are trying to construct the shortest string that matches with this expression, so basically what we will do is NOT take any part of the expression that can be not taken (epsilons, kleen stars etc).
(a*b|ba*|ab*|ba*|(a*b*))
*From this part, we can choose a*b* which generates the empty string epsilon.*
(c*d|cd*|dc*|d*c|(c*d*))

*Similarly from this part, we can choose c*d* which generates the empty string epsilon.*
((a4|c4)(b4|d4)*)
*From this portion, we must take **four a's or four c's**, and as there is a kleen star on **(b4|d4)**, we shall get another empty string epsilon.*

*So the shortest string that matches with this expression is **aaaa or cccc***

***E. Find the shortest string that is NOT in the langua*ge represented by the regular expression**

a*b*((ab)*|(ba)*)b*a*
We are gonna try to construct a string that does not match with this expression using as few symbols as possible.

For the first symbol, we can either choose a or b. Choosing a is not wise, because the a of the expression can consume an infinite number of a's. So we choose b.*

For the second symbol, for the same reason, we shouldn't choose b, as the b can consume as many b's. So we choose a. For the third symbol, if we choose b, the string becomes bab , which can be matched using the green portion of the expression: **a* b*((ab)* |(ba)*)b*a****

Instead, if we choose a, the string becomes baa, which requires the last a of the expression to match:*
a* b*((ab)*|(ba)*)b* a*
or
a*b*((ab)*| (ba)*)b* a*

*Finally, if we take another b as the fourth symbol, then the expression can no longer match our string. So, the shortest string that does not match with this expression is: **baab***

F. Let r1 and r2 be arbitrary regular expressions over some alphabet. Find a simple (the shortest and with the smallest nesting of * and +) regular expression which is equivalent to each of the following regular expressions.
a. (r 1 + r 2 + r 1 r 2 + r 2 r 1)*
b. (r 1 (r 1 + r 2)*)

- *The first expression can be reduced to - (r 1 | r 2)**
  (The sign "+" is also used as OR "|")
- *The second expression can be reduced to - r 1 (r 1 + r 2)**