ENBISYS
ENTERPRISE BUSINESS SYSTEMS

+7 (3822) 99-00-49
info@enbisys.com
enbisys.com

634055, г. Томск,
пр-т Академический,
8/8, оф. 612

Part 1: Language Challenge

This Language Challenge consists of three exercises involving the use of a few operators to create expressions representing possible ways a user might say something in a particular language.

The four operators you will be using are the grouping operators ‘ () ’ and ‘ [] ’, and the connecting operators ‘ . ’ and ‘ | ’. Basic words can be specified by using ‘tokens’ (literal words surrounded by quotation marks). More complicated patterns can be specified by grouping a series of tokens and operators into ‘blocks’ using the grouping operators and then joining blocks and tokens using the connecting operators. There is a detailed explanation below of how to use these operators, and some example of English expressions.

You will be responsible for creating and submitting similar expressions in Russian language.

Explanation of Operators:

‘ () ’ - REQUIRED grouping operator, must surround token(s) that are required to be present
‘ [] ’ - OPTIONAL grouping operator, must surround token(s) that are optionally present
‘ . ’ - CONCATENATE connecting operator, specifies that two tokens or blocks need to be present in sequence
‘ | ’ - OR connecting operator, specifies that one or the other but not both of the two tokens or blocks should be present

Simple example:

Suppose there is an address which begins with the number “1206” (e.g. “1206 Main Street”) and we want to create an expression in English that matches the ways users may say this number.

Some of the most common possibilities would be:

1. one two zero six
2. one two o six twelve o six
3. one thousand two hundred six

If we want to match only the first item on the list, our expression would be:

(" one " . " two " . " zero " . " six ")

This specifies that it's required to find these four words in sequence.

If we want to match both the first item and the second item on the list in a single expression, our expression would be:
(" one " . " two " . (" zero " | " o ") . " six ")

This specifies that the first two words "one" and "two" need to be found in sequence, then a block that specifies either "zero" or "o" needs to be found, and then "six" needs to be found in sequence after that. This example shows that blocks (in this case required blocks) can be nested within each other, and also shows how the OR operator is used.

If we want to extend this expression to add the third item on the list, our expression would become:
(((" one " . " two ") | " twelve ") . (" zero " | " o ") . " six ")

Note that this expression would also match the phrase "twelve zero six" which is good as this is another possible way a user might reasonably say this number.

Optional blocks (using the ' [] ' operator) are similar to required blocks but, as the name suggests, they connote phrasing that is optional (i.e. can be omitted).

For instance, if we have the expression:

(" one " . " thousand " . " two " . " hundred " . [" and "] . " six ")
This expression would match the phrase "one thousand two hundred six" or the phrase "one

thousand two hundred and six".
Please note that the following expression would have a 'bug':

(" one " . " two " . [" zero " | " o "] . " six ")

This is because this expression could match the phrase "one two zero six", "one two o six", or "one two six", and that final phrase corresponds to a different number (126) than the others (1206) which is not the intended behaviour.

Please note that there is no limit on the number of things that can be OR'd or CONCATENATE'd together (e.g. [" a " | " b " | " c " | " d " ...]) and no limit to the number of times that blocks (either required or optional) can be nested within each other.

That means there is a great deal of expressiveness to match non-trivial language patterns using using only these 4 operators.

Also, please note that while it's often convenient to have a token for a single word, it's possible to have tokens that are multiple words like "washing machine" if desired.

Whitespace (spaces and newlines) are meaningless in expressions, and it can often be useful to use lines and indentation to organise a long expression.

Finally, please note that in situations where the order of the connecting operators is potentially ambiguous or confusing, it's possible to use blocks to make it clear to readers which behaviour you want. The ' . ' operator does have a higher precedence than the ' | ' operator, so:
("one" . "two" | "three")

matches "one two" and "three" rather than "one two" and "one three". However, it's often best to write an expression where the intent is clear, like:
(("one" . "two") | "three") .

Here is a more complicated example, in English, of the ways a user may say something equivalent to "thank you":

```
["I" . ("wanted" | "want" | "would like") . "to" . "say"] . (["I" . ("wanted" | "want" | "would like") .
"to"] .
"thank" . "you" . [[("ever" . "so") | "so" | "very"] . "much"]
          |  ["many"] . "thanks"
          |  "thanks" . ["a" . ("lot" | "bunch" | "ton" | "million")]
          |  (["I"] . ["really"] . "appreciate" .
       ("it"
        | ("everything" . [("you've" | ("you" . "have")) . "done"]))
          |  "I" .
((("can't" | "cannot" | "can" . "not") .
"thank" . "you" . "enough") | "owe" . "you" . ["one"]))
```

Here is a list of 'matching phrases' for the expression above:
I wanted to say thanks
thank you so much
I want to say thank you so much

I want to say thanks a lot
I can't thank you enough
many thanks
thanks a ton

thanks a million
thanks
thank you
thank you much
thank you ever so much I owe you one

I really appreciate it
I appreciate it
appreciate it
really appreciate everything I wanted to thank you

I want to say I would like to thank you

Here is a list of ‘non-matching phrases’ for the expression above (note that they are on the

non-matching list because they either mean something different or are not linguistically meaningful

nor something the user would be likely to say): I can’t thank you so much
thank you ever much
I appreciate

really appreciate
I want to many thanks
I wanted to many thanks
owe you
many thanks a lot
I wanted to I really appreciate it

We find it helpful when designing expressions like this to create to a ‘matching’ and ‘non-matching’ phrase list, and then creating an expression that behaves accordingly.

If the explanation above is unclear or you have additional questions about how the 4 operators work, please feel free to e-mail us for clarification.

Language Challenge Exercises:

Your will be responsible for creating and submitting, in Russian expressions (along with ‘matching’ and ‘non-matching’ phrase list) for the following use cases:

- Ways a user might say the number “ 1206 ” as part of a street address
- Ways a user might say something equivalent to “ thank you ” (should ideally be similar in coverage/complexity to the example above)
- Ways a user might say something equivalent to the time of day “ 1:45pm ”. (Examples of how users may say this in English include “one forty five p m”, “quarter till two in the afternoon”, etc).

In each case, your ‘matching’ and ‘non-matching’ phrase lists do not need to be exhaustive, but should provide a similar level of detail to our example above.