Interlude: PA3c - Solutions

These Practice Exercises are meant to help you review for our next IE.

Grab bag

Here is a text file with a series of sentences, one per line. As you see, the word "this" appears on every single line.

Determine, by hand, which line(s) are matched by each of the following Regexs:

- This
- [Tt]his
- this
- this$
- ^this
- ^this$
- \bthis\b
- \<this\>

```
This is a good idea
What about this?
this goes here...
...here goes this
Middle of this...
In java this.methodname works
```

Then test them with *egrep*

Phone Numbers

- We want to write a regular expression which will *only* match what we define to be valid phone numbers. While the digits themselves are irrelevant, each being between 0 to 9, it is their grouping and the various symbols used to do the grouping that are of interest to us.
- We have 2 lists of phone numbers in two separate text files respectively named *positive.txt* and *negative.txt*.
- Use egrep to design a Regex to match all lines in *positive.txt*, and none in *negative.txt*.
- Feel free to add both positive and negative examples.

positives.txt

```
555-667-7088
555 667 7088
(555)667-7088
(555) 667-7088
(555) 667 7088
555	667-7088
```

uses a single tab to separate 555 from 667.

negatives.txt

Mismatched parentheses

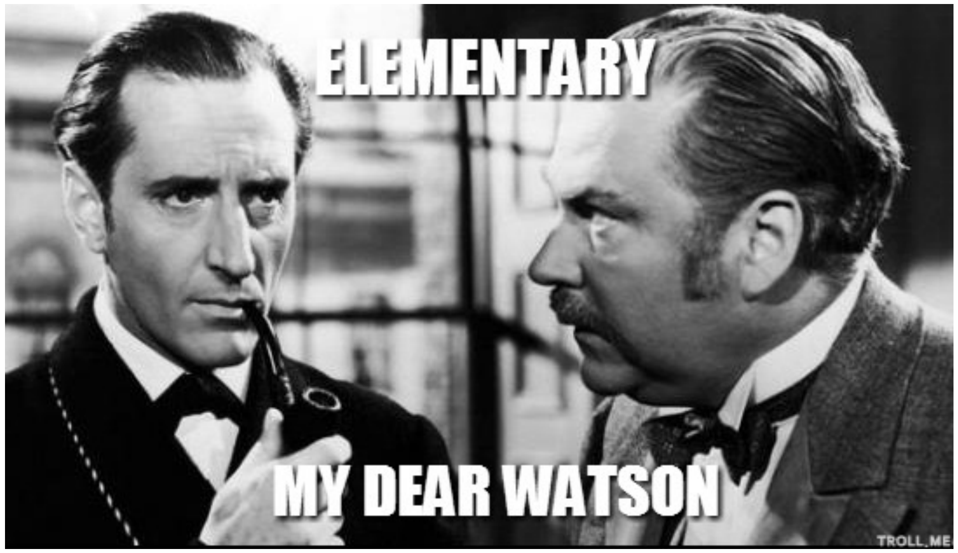
```
(555 667-7088
(555)-667-7088
555     667-7088
```

No dash symbol after parentheses

Multiple spaces or tabs are not allowed to separate the groups of digits.

Solution

```
^(\([[:digit:]]{3}\)([[:blank:]]|)|[[:digit:]]{3}([[:blank:]]|-))[[:digit:]]{3}(\2|\3|-)[[:digit:]]{4}$
```



```
^(\([[:digit:]]{3}\)([[:blank:]]|)|[[:digit:]]{3}([[:blank:]]|-))[[:digit:]]{3}(\2|\3|-)[[:digit:]]{4}$
```

```
^    (
          \([[:digit:]]{3}\)   <------------------------------   When we have
          (    [[:blank:]]                                       parentheses
          |
          )

     |

          [[:digit:]]{3}   <----------------------------------   When we don't
          (    [[:blank:]]
          |
               -
          )
     )
     [[:digit:]]{3}
     (\2|\3|-)
     [[:digit:]]{4}
$
```

URLs

Use *egrep* to match all the lines of a text file which contain a URL.

We define what a URL is explicitly as follows;

- A URL starts with the keyword *http* or *https* written in either all upper or all lower case.
- These are then followed by *://* and a hostname itself followed by a */*.
- After the hostname, we have an optional pathname with a filename at the end.

We will assume the following;

- The hostname ends with a valid domain name; i.e. *com*, *org*, *edu*, or *info*
- Before that, it is made of one or more alphanumerical names, each followed by a single dot
- A pathname is made of one or more folder names, each separated by */*
- A folder name made of one or more alphanumerical characters. Any alphabetical part of the filename might be in upper or lower cases
- A filename is made of one or more alphanumerical characters, followed by a single dot, followed by exactly 3 alphabetical characters. Filenames might be in upper or lower cases

Here are examples of badly formatted URLs that you should not match:

- A URL which starts with a mix of upper and lower case letters, like hTTpS.
- A valid URL start, followed by :/ instead of ://
- A URL without a valid domain name at the end, or a domain name with non-alphanumeric characters like !com&
- A pathname where folders are separated by \ rather than /

This is not an exhaustive list, but it should give you an idea of some of the things you will want to test.

```
(http|https|HTTP|HTTPS)://([[:alnum:]]+\.)+(com|org|edu|info)/(([
[:alnum:]]|_|-|\.)+/)+([[:alnum:]]|_|-|\.)*
```

```
(http|https|HTTP|HTTPS)://([[:alnum:]]+\.)+(com|org|edu|info)/(([
[:alnum:]]|_|-|\.)+/)+([[:alnum:]]|_|-|\.)*
```

Breaking it down

```
(http|https|HTTP|HTTPS)
://
([[:alnum:]]+\.)+
(com|org|edu|info)
/
(     ([[:alnum:]] | _ | - | \. )+
/
)+
([[:alnum:]] | _ | - | \.)*
```

Email Addresses

- Use *egrep* to extract all lines from a plain text file which contain a syntactically valid email address.
- We define a syntactically valid email address as follows;
  - An email address is made of a username followed by the symbol @ and followed by a hostname.
- We will assume that;
  - The username is made of upper and lower cases letters only.
  - Usernames are at least one character long.
  - The hostname ends with a valid domain name
  - Before that, it is made of one or more alphanumerical names, each followed by a single dot
  - A valid domain name is one of the following; com, org, edu, info

```
[[:alnum:]]+@([[:alnum:]]+\.)+(com|org|edu|info)
```

Java Comments

There are two forms of comments in Java.

- The one-line comments start anywhere in a line with *//* and end at the end of the line
- The multi-lines comments start anywhere in a give line with */** and end anywhere in a following line with a **/*

However, sometimes, */*...*/* are used as one-line comments; e.g.,

- `int data = 42; /* this is the magic number */`
- The closing */ might be followed by spaces or tabulations instead of being right at the end of the line
- In these situations, we might have used a *//* comment instead in such situations.

- Use *egrep* to match lines from a java file with such single-line comments.
- Then pipe the result to another tool to count how many such lines were found.
- Your java file will be free of syntax errors and run without issues.

```
grep .*/\*.*\*/[[:blank:]]*$ something.java |wc -l
```

Simple #define

The C and C++ languages are known for their pre-compilation directives. One of them, #define allows to define macros and symbolic constants.

I'd like to extract from my .c files all the lines which are defining an integer constant.

The syntax of #define might get pretty involved, however, in order to define a symbolic constant, one has only to do the following;

- #define RESPONSE   42
- #define   INTERGER_PI   3

Simple #define

Keep in mind the following rules;

- The #define might start at the beginning of the line or be preceded by an arbitrary number of spaces or tabs.
- It is never preceded by anything else.
- The spacing between the constant name and its integer value is also variable.
- We will restrict ourselves to integer values, up to 6 digits only.
- By convention, all programmers working with me are naming their symbolic constants only using upper case letters

Use egrep to extract all the lines which define such integer symbolic constants.

```
grep
[[:blank:]]*\#define[[:blank:]]+([[:upper:]]|_)+[[:blank:]]+[[:di
git:]]{1,6} something.cpp
```