TP 1
Rational Expressions

Version du 21 septembre 2020

Objective

The goal of this TP is to write rational expressions that can be applied to extract text snippets from files and manipulate them. We will use the language Perl [1] that should already be installed on most operating systems.

Unix Command Line Cheat Sheet

Here is a quick reminder of a few useful shell command lines.

`mkdir` *`dir`*	create a directory *`dir`* (MaKe DIRectory)
`ls`	list files in the current directory (LiSt)
`cd` *`dir`*	change directory to *`dir`* (Change Directory)
`cd ..`	moves back to the parent directory
`cd`	moves back to the home directory *HOME*
`cp` *`src dest`*	copy a file (CoPy)
`cp -R` *`src dest`*	copy a directory (CoPy)
`mv` *`src dest`*	move and rename a file or a directory (MoVe)
`cat` *`file`*	display the content of file *`file`* on the standard output

At any time, the `man` (MANual) command can be used to display the user manual of any command that we can run on the shell. As an example, if you want to lean how to use `mkdir`, type '`man mkdir`'.

Try to order your file directories in such a manner you can review your previous works later. As an example, create a parent directory ('`mkdir tps`'), move to this directory ('`cd tps`'), then create a new directory `thlr` ('`mkdir thlr`'), as well as a new directory for each TP. You can later copy and paste the rational expressions you designed in text files.

Note that the command `history` displays a history of executed commands and may prove useful if you need to retrieve previous inputs.

Using Rational Expressions in Perl

The Perl language uses its own syntax for regular expressions that differs from the conventional notation we use during TDs. In the following table, $c_1, c_2, \ldots, c_n$ represent letters in Σ, and $L(e)$ is the language matched to the rational expression e.

1. http://www.perl.org. Use `man perlretut` to display a Perl tutorial on rational expressions, then type '`q`' to exit.

language	Perl rational expression
Σ	`.`
$\{c\}$	`c`
$\{c_1, c_2\}$	`[`c_1c_2`]`
$\{c_1, c_2, \ldots, c_n\}$	`[`c_1`-`c_n`]`
$\Sigma \setminus \{c_1, c_2\}$	`[^`c_1c_2`]`
$L(e)L(f)$	ef
$L(e) \cup L(f)$	$(e \mid f)$
$L(e) \cup \{\varepsilon\}$	e`?`
$L(e)^\star$	e`*`
$L(e)^+$	e`+`

Note that you can consider the union of character classes. As an example, `[a-zA-Z]` stands for lower or upper case letters. Special characters such as `\ . ? * [ ] | ( )` should be preceded by a '`\`'. Hence, if you are looking for the character dot '`.`', you should write '`\.`'.

In order to test a rational expression with Perl, use the following command line in the shell :

```
perl -0777 -pe 's/rational-expression/text-replacement/gsm' file
```

The variable *`text replacement`* replaces each occurrence of the rational expression. As an example, applying '`perl -0777 -pe 's/[a-z]/@/gsm'`' to a file whose content is '`@@@@@@AZERTY@@@@`' would yield the output '`@@@@@@AZERTY@@@@`' : each lower case letter had been replaced by a '`@`'. Note that the original file has not been modified by this operation.

Files Needed For This TP

For each question, you will need a test file on which you can apply the rational expression you designed. These files can be found in the directories '`TP1_files/exo1`', ..., '`TP1_files/exo4`' and are numbered according to their matching question (preceded by the symbol '`q`'). You should therefore test the question 2 of exercise 1 on the file '`TP1_files/exo2/q2`'. For a given question of a given exercise, don't forget to check if your rational expression also works on the files matched to the previous questions.

In order to copy the aforementioned files, type the following command lines in your shell :

```
wget http://www.lrde.epita.fr/~akim/thlr/TP1_files.tar.bz2
tar -xjvf TP1_files.tar.bz2
```

Exercice 1 – Comments in Pascal programs

We want to find comments of the form `{`*`text`*`}` in Pascal files, where *`text`* stands for any sequence of characters.

1. First, we assume that there is only a single comment per file. Use Perl to replace it with the text `COMMENTAIRE`.
2. What happens if there are more than a single comment ? Write a new rational expression in order to fix this issue, while still replacing every single comment with the text `COMMENTAIRE`.

Exercice 2 – Strings in C programs

We are looking for strings in C source files. Strings are arbitrarily long sequences of characters between quotation marks : "*`texte`*", where *`texte`* can be any sequence of characters.

1. Write a rational expression in order to find and replace each string with the text `CHAINE`.
2. Special characters can only be used in strings if they are 'escaped' first, i.e. preceded by the character \. As an example, in order to print the character ", one must write `"...\"..."` to avoid the character " being mistaken for the end of the string. Change your rational expression in order to take these special characters into account.

Exercice 3 – Mass modifications of identifiers

It is possible to refer to patterns matched to a rational expression by using the variables `$1`, `$2`, etc. Theses variables refer to sets of parentheses in the rational expression. Hence, `$n` refers to the n-th pair of parentheses. As an example, applying `perl -0777 -pe 's/([a-z])/$1@/gsm'` to a file whose content is `'azertyAZERTY'` will yield the output `'a@z@e@r@t@y@AZERTY'` : we added a character `'@'` after each lower case letter. Theses references can be very useful if we need to perform mass substitutions on a text file.

1. We want to modifiy function identifiers of the form `get_`*`Identifiant`*`()` or `set_`*`Identifiant`*`(arguments)`, where *`Identifiant`* can be any sequence of alphanumerical characters, by removing the character '_'. Keep in mind that we only want to modify function identifiers whose names are preceded by a blank space then '`get_`' or '`set_`', and followed by an alphanumerical sequence of arguments between a pair of parentheses.
2. We want to modify each function call starting with an upper case letter by replacing said letter with its matching lower case letter. As an example, we should replace `Apply()` with `apply()` and `Process(int i)` with `process(int i)`. Function identifiers are still alphanumerical sequences of characters preceded by a blank space and followed by an alphanumerical sequence of arguments between a pair of parentheses.
   In order to turn an upper case character into a lower case character, one should put `\l` before said character [2] in the substitution pattern.

Exercice 4 – Nested comments

1. We now allow nested comments of the form `{Text {Nested text}}` in Pascal source files. Look for a regular expression that would allow us to replace such a sequence with `@COMMENTAIRE@COMMENTAIRE@@`, in a similar manner to exercise 1.

2. '`man perlre`'.