Dans ce TP, nous supposerons travailler avec l'alphabet Σ qui est l'ensemble des caractères ASCII.

1 Introduction

Très tôt dans l'histoire de l'informatique, l'action de rechercher certains motifs dans une collection de fichiers textes est apparue comme importante. Dans ce but, l'informaticien Keneth Lane Thompson (co-créateur de la première bersion du système Unix, du standard UTF-8, et de nombreux autres logiciels) développe à Bell Labs l'outil `grep`. Ce dernier permet de rechercher dans des fichiers textes toutes les occurrences des chaines appartenant au langage d'une expression régulière donnée. Toutes les lignes des fichiers dont une sous-chaine vérifie l'expression régulière sont alors affichées sur la sortie standard.

Les expressions régulières reconnues par `grep` sont une généralisation des expressions régulières formelles vues en cours. Le tableau ci-dessous résume la syntaxe POSIX BRE (**P**ortable **O**perating **S**ystem **I**nterface for Unix ; **B**asic **R**egular **E**xpressions), et donne leur encodage en expression régulière simple lorsque cela est possible.

expression POSIX	signification	expression formelle
`a`	caractère fixe `a`	a
`.`	n'importe quel caractère	$a_1 \vert \cdots \vert a_n$ pour $a_i \in \Sigma$
`[c1 ... cn]`	n'importe quel des caractères `ci`	$c_1 \vert \cdots \vert c_n$
`[^c1 ... cn]`	n'importe quel caractère autre que l'un des `ci`	$a_1 \vert \cdots \vert a_k$ pour $a_i \in \Sigma \setminus \{c_1, \ldots, c_n\}$
`[c1-cn]`	n'importe quel caractère de l'intervalle `c1-cn`	$c_1 \vert \cdots \vert c_n$
`[^c1-cn]`	n'importe quel caractère hors de l'intervalle `c1-cn`	$a_1 \vert \cdots \vert a_k$ pour $a_i \in \Sigma \setminus \{c_1, \ldots, c_n\}$
`e1e2`	concaténation	$e_1 e_2$
`e1 | e2`	union	$e_1 \vert e_2$
`e*`	étoile de Kleene	e^*
`e\+`	répétition au moins une fois	ee^*
`e\?`	répétition au plus une fois	$\varepsilon \vert e$
`e\{m,n\}`	répétition entre m et n fois	$e^m \cdot (\varepsilon \vert e)^{(n-m)}$
`\(e\)`	parenthèses autour d'une expression	(e)
`^`	début de ligne	pas d'équivalent
`$`	fin de ligne	pas d'équivalent

Aux expressions régulières simples, la norme POSIX ajoute de nombreuses facilités syntaxiques :

- le caractère joker "`.`" qui peut remplacer n'importe quel caractère ;
- les intervalles de caractères. Par exemple :
  - `[a-z]` désigne l'ensemble des lettres minuscules ;
  - `[A-Z]` désigne l'ensemble des lettres majuscules ;
  - `[0-9]` désigne l'ensemble des chiffres.

Concernant les caractères `(`,`)`,`|` : utilisés seuls, ils sont considérés comme une lettre de Σ. Si on veut les utiliser comme connecteur dans l'expression régulière, ils faut les "échapper" avec un `\`. Par exemple :

- `(ab)*` reconnait les mots de la forme $(ab)))\cdots)$;
- `\(ab\)*` reconnait les mots de la forme $abab\cdots ab$;

Concernant les caractères `[`,`]`,`-`,`^`,`*`,`.`,`$`, c'est l'inverse : utilisés seuls, ils correspondent à un connecteur dans l'expression, et il faut les "échapper" avec un `\` pour chercher la lettre correspondante. Par exemple :

- `a$` cherche les lignes finissant par un a ;
- `a\$` cherche les lignes contenant a\$.

Dans un terminal, on pourra ainsi écrire `grep 'expression_posix' nom_du_fichier`

Remarque. Même si ça ne sera pas utile pour ce TP, on peut demander à `grep` de chercher dans plusieurs fichiers, grâce au symbole `*` qui sert de joker dans les noms de fichiers. Par exemple :

- `grep 'expr' *` va chercher dans tous les fichiers du dossier courant ;
- `grep 'expr' tp*.c` va chercher dans tous les fichiers du dossier courant dont le nom commence par '`tp`' et fini par '`.c`'.

2 Exercices

Télécharger le fichier `tp02.zip` sur la page web du cours. Il contient plusieurs fichiers sur lesquels on va travailler avec `grep` pour ce TP. On peut extraire ces fichiers avec la commande : `unzip tp02.zip`.

1. Chercher, à l'aide de `grep`, toutes les lignes du fichier `binaire.txt` qui commencent par un entier écrit en binaire sans zéro non significatif.

   On peut passer des options à la commande `grep`, pour modifier son comportement.

   - L'option `-v` permet de n'afficher que les lignes **ne** contentant **pas** l'expression donnée.
     Utilisation : `grep -v expression fichier`
   - L'option `-o` permet de n'afficher que le morceau de la ligne satisfaisant l'expression donnée.
     Utilisation : `grep -o 'expression' fichier`
   - L'option `-c` permet de compter combien de lignes vérifient l'expression donnée.

2. Chercher, à l'aide de `grep`, toutes les lignes du fichier `binaire.txt` n'ayant pas été sélectionnées par la question 1.
3. Renvoyer, à l'aide de `grep`, la liste de tous les identificateurs C (noms de variables ou mots clés) du fichier `area.c`.

On peut demander à `grep`, plutôt que de travailler sur un fichier, de travailler sur le texte que va renvoyer une autre commande, à l'aide du symbole `|` ("pipe"), via la syntaxe suivante :

```
commande | grep expression
```

En particulier, on peut composer les commandes `grep` entre elles :

```
grep 'expr1' fichier.txt | grep 'expr2' | grep 'expr3' | ... | grep 'exprn'
```

4. Reprendre la question 3, mais faire en sorte d'ignorer les lignes contenant un commentaire (`//`).
5. Lister, à l'aide de `grep`, les variables du fichier `area.c`.
6. Lister, à l'aide de `grep`, les constantes flottantes du fichier `area.c`.
7. Trouver toutes les adresse email valides du fichier `mail.csv`.
8. Combien y a-t-il d'adresses email valides ?

Lorsqu'une expression est mise entres `\(e1\)`, on peut indiquer que la sous-chaine capturée par e_1 doit réapparaitre plus tard : pour cela, on utilise `\1` pour la désigner si c'est la première utilisation de `\(...\)`, `\2` si c'est la seconde utilisation de `\(...\)`, etc.

9. Trouver toutes les adresses emails valides du fichier `mail.csv` qui sont de la forme `prenom.nom@...`.
10. Combien y en a-t-il ?