Feuille de TD n° 9 : Regexp

Ce TD s'intéresse aux expressions régulières telles qu'elles sont utilisée en programmation. Elles y sont nommé « regexp ». On introduit maintenant la notation utilisées par les regexps. Il faut noter que les regexp varient légèrement d'un langage à l'autre, on introduit ici les notations les plus standard. En particulier, on s'intéressera aux expressions que vous pourrez utiliser dans des programme java.

Comme les expression régulières les regexp sont définies récursivement. Pour r_1 et r_2 deux regexp, alors :

- $r_1 r_2$ reconnait les concaténations d'un mot reconnus par r_1 et d'un mot reconnu par r_2,
- $r_1|r_2$ reconnaît les mots reconnus par r_1 ou par r_2,
- $r_1?$, r_1^+, r_1^*, $r_1\{m,\}$ et $r_1\{m,M\}$ reconnaît une suite de n mots reconnus par r_1, avec $n \leq 1$, $n \geq 1$, $n \geq 0$, $n \geq m$ et $m \leq n \leq M$ respectivement.

Tout comme avec les expressions arithmétiques, les parenthèses sont parfois obligatoires pour lever l'ambiguïté. Ainsi (ab^*) reconnait les mots composés d'une suite de ab, alors que ab^* reconnait les mot composés d'un a, puis d'un nombre fini de b. Ce langage est aussi accepté par $a(b^*)$, on verra plus tard l'intérêt que peuvent avoir ces parenthèses.

- a reconnaît le caractère a,
- (reconnait (,

  reconnait
  et ainsi de suite pour les autres symboles ayant une signification spéciale dans les regexp.
- [*descr*] reconnaît un ensemble de lettre décrit d'une des trois manières suivantes :
  - Si *descr* commence par ^ et est composé d'une suite de caractère, alors l'expression reconnait n'importe quel caractère qui n'est pas dans cette suite.
  - Sinon, si *descr* ne commence pas par ^, alors l'expression reconnait n'importe quel lettre entre les crochets,
  - Enfin, [a-e] reconnaît les lettres entre a et e, et de même pour n'importe quel paire de lettre ou de chiffre.
- Le point reconnaît n'importe quel lettre sauf le retour à la ligne.
- Il y a les abréviations suivantes :
  - \w reconnaît les caractères de mots,
  - \d reconnaît les chiffres,
  - \s reconnaît les caractère d'espaces,

  La même abréviation en majuscule reconnaît son complément.
- ^ reconnaît le début du texte [1].
- $ reconnaît la fin du texte.

Exercice 1 : Quelques regexp

Donner une regexp pour les langages suivants :

1. On rappelle qu'on n'est dans une regex et pas dans un ensemble de caractère.

Numéro de sécurité sociale Le numéro de sécurité sociale est composé de 15 chiffres de la forme suivante :
- 1 pour homme ou 2 pour femme,
- deux derniers chiffre de l'année de naissance,
- mois de naissance,
- département de naissance (01 à 99, sauf 20. Et 2A et 2B)
- commune de naissance (3 chiffres de 001 à 990)
- numéro d'ordre de naissance dans la commune (de 001 à 999)
- Clé de contrôle égal au complément à 97 du nombre formé par les 13 premiers chiffres, modulo 97.

On ignorera la clé de contrôle, qui doit être traité or des regexp. La regexp doit par contre refuser les nombres qui ne respectent pas les autres règles.

Email La norme des adresse email très compliqué, en particulier car elle tient compte des langages utilisant différents alphabets. Ici, on supposera qu'une adresse email contient exactement une arobase, au moins un point à droite de l'arobase, et au moins un caractère avant l'arobase, entre l'arobase et le point, et après le point. Ces autres caractères sont des points, tirets, lettres et chiffres.

Téléphone Les numéros de téléphones sont notés de manières différentes selon les pays, et le nombre de chiffre varie. On cherche ici à reconnaître les suites d'au moins 3 nombres, avec potentiellement entre eux des espaces, parenthèses, point et/ou tiret.

Le second principal intérêt des regexp est de pouvoir réutiliser des facteurs du texte trouvé. Encore une fois, la syntaxe exacte dépend du langage utilisé. Plus précisément, ces facteurs sont appelés des groupes, ce sont les sous-expressions parenthésée. Par exemple l'expression régulière $(a^+)[\hat{}a](a^+)$ reconnaît une suite de a avec une lettre différent de a au milieu. Les groupes 1 et 2 consistent en la première et la deuxième suite de a trouvé. Le groupe 0 consiste en la première suite, la lettre séparant les deux suites, puis la deuxième suite.

En java, la première étape consiste à compiler une rexegp r (intuitivement, transformer une expression régulière en automate), puis créer un objet m de la classe Matcher. Cet objet m va chercher tous les facteurs successifs qui sont accepté par r.

```
Pattern p=Pattern.compile("Une rexegp");
Match m=p.matcher("Le texte dans lequel chercher");
```

La fonction find() permet de trouver un facteur qui est accepté par la regexp. La fonction replaceALl permet de remplacer tous les facteurs accepté par la regexp par une expression. Cette expression peut utiliser le texte contenu dans le ième groupe, grâce à $\$i$.

Un exemple classique d'utilisation des regexp consiste en l'utilisation de fichier au format csv. Un fichier au format csv représente une base de donnée ou la première ligne contient les noms des champs, séparé par des virgules, et où les lignes d'après contiennent les valeurs de ces champs, aussi séparé par des virgule. Par exemple

```
nom,prénom,année de naissance,année de mort
Turing,Alan,1912,1954
Knuth,Donald,1938,
```

Ainsi le code

```
String pattern="([^,]*),([^,]*),([^,]*),([^,]*)";
String remplace="$2 $1 est ne en $3.";
Pattern p=Pattern.compile(pattern);
Match m=p.matcher(fichier.csv);
System.out.print(m.replaceALl(remplace));
```

affiche « Alan Turing est ne en 1912.
Donald Knuth est né en 1938. »

Exercice 2 : Chercher, remplacer

On donne quelques remplacement qu'un programmeur souhaiterait pouvoir faire dans son code, écrivez un morceau de code expliquant comment ce changement implémenté à l'aide de regexp.

1. Dans votre code java, vous décider que tous les if sont suivi d'accolades. On ne se souciera pas d'indentation. Votre expression remplace les « if(cond) expr ; » par « if(cond){expr ;} »
2. Votre site web s'appelle `www.foo.fr`. Dans le code de votre site web, vous souhaitez remplacer tous les liens absolus, de la forme `http://www.foo.fr/bar` par le lien relatif `/bar`. Ce changement ne doit intervenir que dans les balises, et pas dans le texte lui même.

   Rappel : en html, un lien est de la forme « <a [paramètres]*>texte du lien » où les paramètres sont de la forme « nom="valeur" » au moins un nom de paramètre est « a ».

Quelques compléments Vous aurez peut-être compris que, dans le code donné plus haut, un simple chercher-remplacer ne peut pas tenir compte de la présence ou non d'une année de décès. Ça peut toujours être fait via les Pattern grâce au code suivant :

```
Pattern p=Pattern.compile("([\^,]*),([\^,]*),([\^,]*),([\^,]*)");
Match m=p.matcher(fichier.csv);
while(m.find()){
  System.out.print(m.group(2)+" "+m.group(1)+" est n\'e en "+m.group(3));
  if(m.group(4)="") System.out.println(" et est toujours vivant.");
  else System.out.println(" et est mort en "+m.group(4)+".");
}
```

L'exécution de ce code affiche « Alan Turing est né en 1912 et est mort en 1954.
Donald Knuth est né en 1938 et est toujours vivant. »

Plusieurs notions ont été laissé de côté. Vous êtes encouragé à aller vous documenter pour découvrir l'expressivité des regexp. Par exemple :

— Les parenthèse imbriqués,
— Les parenthèses sous des étoiles,
— La commande \1, qui permet de reconnaître exactement le langage reconnu par le premier groupe. (Cette extension permet de reconnaître des langages non réguliers).
— La notion de gourmandise. Est-ce qu'une étoile doit accepter le plus petit ou le plus grand mot ? Cette question n'a pas d'intérêt pour reconnaître un langage mais devient importante pour un chercher-remplacer correcte.

Certaines de ces notions seront plus étudié durant le cours de compilation.