Université du Maine
P. JACOBONI, E. DELOZANNE

Introduction à La Programmation Orienté Objet.
Application à Ruby

TD n° 4 & 5 : Chaînes et Collections

Exercice 1 : Reprise de l'exo du TP 1 sur les NR.

Exercice 2 : Afficher le nombre d'occurences des lettres d'une chaîne de caractère. On donnera une version correspondant a une programmation "à la pascal et C++", puis on essaiera de trouver un programme plus court en utilisant les possibilités de Ruby.
Rappel, si ch est la chaine "Il est fort ce Jacoboni"
Alors
ch[0] représente le code ascii de "I"

Exercice 3 : Renverser une chaîne de caractères (par caractère/ par mot)
Par caractère : "Ruby est très joli" → "iloj sèrt tse ybuR"
Par Mot : "Rub y est très joli" → "joli très est Ruby"
Principe : "Exploser" la chaine en tableau de caractères le renverser et reconstituer la chaîne.
Les méthodes split, reverse et join permettent de réaliser cela.

Exercice 4 : Ecrire une méthode `enleveStr(s2)` de la classe Chaîne permettant d'enlever toutes les occurrences de la chaîne `s2` dans la chaine receveuse.

Algorithme en C
```
char * enleveStr(char * s1, const char * s2){
   char * p;
      while (p = strstr(s1, s2)){
             strcpy(p, p + strlen(s2));
     }
   return s1;
}
```

Exercice 5 : Ecrire une méthode `enleveChr(s2)` de la classe Chaîne permettant d'enlever toutes les occurrences des charactères de la chaîne `s2` dans la chaine receveuse.
Ex : `"123456789012345678 90".enleveChr("13579")` → `"2468024680"`

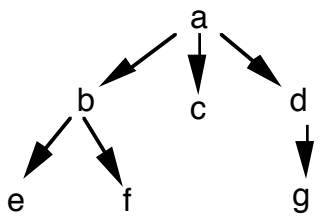
Exercice 5bis : Ecrire une méthode `enlevePasChr(s2)` de la classe Chaine permettant de ne garder de la chaine receveuse que les caractères de la chaine s2
Ex : `"Despres".enlevePasChr("aeiouy")` → `"ee"`

Exercice 6 : Donner une réalisation en Ruby du TDA Pile (sommet, empiler, dépiler, estVide, taille)

Exercice 7 : Donner une réalisation en Ruby du TDA File (entrer, sortir, estVide, taille)

Exercice 8 : Une expression postfixée étant contenue dans une chaîne, écrire le code Ruby permettant de l'évaluer.. ex : '128 12+10-2*' => 260

Exercice 9 (exam de février 1976)



1) On se propose de mettre en œuvre en Ruby les arbres généraux dans une représentation par liste des fils.
Rappel : un arbre général n'est jamais vide.
Un arbre est caractérisé par l'étiquette de sa racine et par la liste de ses fils.
On prévoira :

construction :

- de construire un arbre à partir de la liste de ses fils et de l'étiquette de sa racine (construction à côté)

accès :

- de retourner le premier fils d'un arbre
- de retourner le fils suivant d'un fils d'un arbre
- de retourner l'étiquette d'un arbre

test :

- tester si l'arbre est reduit à une feuille (liste des fils vide)
- tester si le fils d'un arbre est son dernier fils

2) On définira des opérations permettant

- de construire l'arbre donné en exemple sur la figure
- d'afficher les étiquettes d'un arbre par ordre préfixe (racine, premier,… dernier) :
  (a (b (e) (f)) (c) (d (g)))
- d'afficher les étiquettes par niveau :
  a b c d e f g
- de saisir les étiquettes des nœuds

Annexes

Syntaxe de split : **`str.split( pattern=$;, [limit] ) -> anArray`**

Divise str en sous chaines d'après un délimiteur. Retourne un tableau de ces sous-chaînes
Si "pattern" est une chaine son contenus est utilisé comme délimiteur pour découper la chaine.
Si "pattern" est une expression régulière, str est découpé à l'endroit ou le "pattern" est en correspondance.
Si "pattern" n'est pas précisé c'est la variable prédéfinie $; qui est utilisée is used. Si $; est nil (c'est le défaut), la découpe se fait sur l'espace.

```
" now's  the time".split      ->    ["now's", "the", "time"]
" now's  the time".split(' ')->    ["now's", "the", "time"]
" now's  the time".split(/ /)->    ["", "now's", "", "the", "time"]
```

Syntaxe de join : **`arr.join( aSepString=$, ) -> aString`**

Returns a string created by converting each element of the array to a string, separated by aSepString.

```
[ "a", "b", "c" ].join        ->    "abc"
[ "a", "b", "c" ].join("-")   ->    "a-b-c"
```

Syntaxe de gsub, méthode de la classe String : **`str.gsub( pattern, replacement ) -> aString`**

Les méthodes sub et gsub recherche une portion de chaîne dans la le receveur satisfaisant le critère de recherche donné par le premier argument et le remplace par le second. sub effectue un remplacement alors qe gsub remplace toutes les occurrences. Ces deux méthodes renvoient une copie de la chaîne modifiée. Il existe les versions destructives équivalentes sub! et gsub!

```
a = "the quick brown fox"
a.sub(/[aeiou]/,  '*')        ->  "th* quick brown fox"
a.gsub(/[aeiou]/, '*')        ->  "th* q**ck br*wn f*x"
a.sub(/\s\S+/,  '')           ->  "the brown fox"
a.gsub(/\s\S+/, '')           ->  "the"
```