Haute École de Bruxelles
École Supérieure d'Informatique

TD Chaines

Résumé

Voyons ici les caractères et les chaines de caractères.

1 Les chaines

Nous avons introduit le type chaine mais nous n'en avons encore fait que des usages basiques, essentiellement des affichages. Il est temps d'aller plus loin et de les manipuler, c'est-à-dire d'examiner et de manipuler le contenu de chaines. Mais procédons par étapes.

Certains langages, comme Java, distinguent clairement le type chaine et le type caractère. Ainsi, en Java, le littéral 'A' représente un caractère qui est différent du littéral "A" qui représente une chaine (composée d'un seul caractère). Cette distinction est essentiellement dictée par des détails d'implémentation ; un caractère pouvant se représenter de façon plus économe qu'une chaine.

En algo, nous introduirons un type caractère mais sans le distinguer clairement d'une chaine. Ainsi, un caractère pourra être utilisé comme une chaine et une chaine d'un seul caractère pourra être considéré comme un caractère. En somme, caractère n'est qu'un raccourci pour dire : une chaine de longueur 1.

Nous avons donc deux types de variables qui permettent de stocker du texte :

- le caractère (pour les différentes lettres et symboles qui apparaissent sur le clavier de votre ordinateur, par exemple 'a', 'B', ' ?', '3', '@', etc.)
- et la chaine (qui est un assemblage de plusieurs caractères)

Observez la petite nuance d'écriture en Java : un caractère sera entouré d'apostrophes (ou simples guillemets) (`' '`) et une chaine de doubles guillemets (`" "`).

La **taille (ou longueur) d'une chaine** est le nombre de caractères qu'elle contient. Pour la connaitre on utilisera la notation `long(uneChaine)`.

Remarquez qu'une chaine peut être vide, mais pas un caractère ! Ne confondez pas la chaine vide (`""`, de taille nulle) avec le caractère blanc (' ') qui contient l'espace séparant 2 mots d'un texte et que vous obtenez en enfonçant la touche d'espacement au bas du clavier.

1.1 Manipuler les caractères

Pour pouvoir manipuler une chaine, il faut pouvoir accéder aux caractères qui la composent. Comme il s'agit d'une opération de base souvent utilisée, nous allons introduire une écriture compacte, empruntée aux tableaux : `nomChaine[i]` désigne le ie caractère de la chaine `nomChaine` (en commençant à 1).

Par exemple :

```
texte : chaine
texte ← "Hello"
afficher texte[5  ] // Affiche "o"
texte[2] ← "a"  // texte vaut "Hallo" ; un caractère doit être remplacé par un seul
```

Écrivons un algorithme qui vérifie si un mot donné contient une lettre donnée
.

```
algorithme contient(mot : chaine, lettre : caractère) → booléen
    i : entier
    i ← 1
    tant que i≤long(mot) ET mot[i]≠lettre faire
      i ← i+1
    fin tant que
    retourner i≤long(mot)
fin algorithme
```

Introduisons quelques fonctions (ou primitives) agissant sur les caractères. Leur écriture s'apparente à celle d'algorithmes que vous pourrez utiliser tels quels dans les exercices.

```
estLettre(car : caractère) → booléen
```

Cette fonction indique si un caractère est une lettre. Par exemple elle retourne vrai pour 'a', 'e', 'G', 'K', mais faux pour '4', '$', '@'. . .

Si on veut savoir si une lettre est une minuscule ou majuscule, on utilisera les fonctions analogues

```
estMinuscule(car : caractère) → booléen
```

et

```
estMajuscule(car : caractère) → booléen
```

Il va de soi que si `car` n'est pas une lettre, ces deux fonctions retournent faux.

```
estChiffre(car : caractère) → booléen
```

Cette fonction permet de savoir si un caractère est un chiffre. Elle retourne vrai uniquement pour les dix caractères "0", "1", "2", "3", "4", "5", "6", "7", "8" et "9" et faux dans tous les autres cas.

toutes les lettres d'une chaine en majuscules, grâce à la fonction :

```
majuscule(texte : chaine) → chaine
```

L'opération inverse se fait avec :

```
minuscule(texte : chaine) → chaine
```

Il peut aussi être pratique de connaitre la position d'une lettre dans l'alphabet. Ceci se fera à l'aide de la fonction :

```
numLettre(car : caractère) → entier
```

qui retourne toujours un entier entre 1 et 26. Par exemple numLettre("E") donnera 5, ainsi que numLettre("e"), cette fonction traite donc de la même manière les majuscules et les minuscules. En vertu de ce qui a été écrit plus haut, numLettre retournera aussi 5 pour les caractères "é", "è", "ê", "ë". . .). N.B. : attention, il est interdit d'utiliser cette fonction si le caractère n'est pas une lettre !

Il peut être utile d'avoir un outil qui fait l'opération inverse, à savoir associer la lettre de l'alphabet correspondant à une position donnée. Pour cela, nous aurons :

```
lettreMaj(n : entier) → caractère
```

et

```
lettreMin(n : entier) → caractère
```

qui retournent respectivement la forme majuscule ou minuscule de la n-ème lettre de l'alphabet (où n sera obligatoirement compris entre 1 et 26). Par exemple, `lettreMaj(13)` retourne `"M"` tandis que `lettreMin(26)` retourne `"z"`.

1.2 En Java

java.lang.Character

Allez voir la classe `java.lang.Character` dans l'API et trouvez-y les méthodes Java correspondant aux algorithmes définis ci-dessus.

1.3 Chaine et nombre

Si une chaine contient un nombre, on doit pouvoir le convertir, et inversement.

```
chaine(n : réel) → chaine
```

transforme un nombre en chaine. Ex : `chaine(42)` retourne la chaine "42" et `chaine(3,14)` donnera "3,14".

```
nombre(ch : chaine) → réel
```

transforme une chaine contenant des caractères numériques en nombre. Ainsi, `nombre("3,14")` retournera 3,14. C'est une erreur de l'utiliser avec une chaine qui ne représente pas un nombre.

1.4 Concaténation

Il est fréquent de devoir rassembler plusieurs chaines pour former une seule chaine plus grande, il s'agit de l'opération de concaténation. Introduisons également une notation compacte, le +.

Par exemple :

```
texte ← "al" + "go" + "rithmique"  // assigne la chaine "algortihmique" à l
```

Écrivons un algorithme qui inverse toutes les lettres d'un mot. Ainsi, "algo" deviendra "ogla".

```
algorithme miroir (mot : chaine) → chaine
    miroir : chaine
    miroir ← ""
```

```
    pour i de 1 à long(mot) faire
      miroir ← mot[i] + miroir
    fin pour
    retourner miroir
fin algorithme
```

1.5 Conversion de chaine en Java

Pour convertir un type primitif en `String`, il suffit d'utiliser l'opérateur `+` avec un opérande de type `String`.

Par exemple :

```
System.out.println ("1+1␣=␣"+2);
String s = "Pi␣=␣" + 3.1415;
System.out.println ("1"+2+3); // 123
System.out.println (1+2+"3"); //33
```

Pour convertir un `String` en `char` , il faut penser à la méthode `charAt()` :

```
String nom = "Toto";
char initiale = nom.charAt(0);
```

Pour convertir un `String` en `int` , il faut aller voir l'API de la classe `Integer` :

```
int nb = Integer.parseInt ("12");
```

1.6 Manipuler les chaines

Extraction de sous-chaine

Cette fonction importante permet d'extraire une portion d'une certaine longueur d'une chaine donnée, et ceci à partir d'une position donnée. L'en-tête de cette fonction (qui reçoit donc 3 paramètres entrants) est la suivante :

```
sousChaine(ch : chaine, pos : entier, long : entier) → chaine
```

Il faudra aussi être très vigilant pour une utilisation correcte : pos doit être compris entre 1 et la taille de la chaine, et la valeur de long doit être telle qu'on ne déborde pas de la chaine !

Par exemple, `sousChaine("algorithmique", 5, 3)` donne `"rit"`.

Cette fonction est très utile pour sélectionner des portions d'une chaine contenant des in- formations codées sous un certain format. Prenons par exemple une date stockée dans une chaine ch de format "JJ/MM/AAAA" (de longueur 10). Pour extraire le jour de cette date, on écrira `sousChaine(ch, 1, 2)` ; le mois s'obtiendra avec `sousChaine(ch, 4, 2)` et l'année avec `sousChaine(ch, 7, 4)`. Attention, les 3 résultats obtenus sont des chaines de chiffres et non des nombres mais il est possible de les convertir via la fonction `nombre` !

Recherche de sous-chaine

Cette fonction permet de savoir si une sous-chaine donnée (ou un caractère) est présent dans une chaine donnée. Elle permet d'éviter d'écrire le code correspondant à une recherche :

`position(ch : chaine, sous-chaine : chaine)` $\rightarrow$ `entier`

La valeur de l'entier renvoyé est la position où commence la sous-chaine recherchée.

Par exemple, `position("algorithmique", "mi")` retournera `9`. Si la sous-chaine ne s'y trouve pas, la fonction retourne `0`. On peut admettre ici d'écrire un caractère à la place de la sous-chaine.

Par exemple, `position("algorithmique", "m")` retournera également `9`.

1.7 En Java

java.lang.String

Allez voir la classe `java.lang.String` dans l'API et trouvez-y les méthodes Java correspondant aux algorithmes définis ci-dessus.

1.8 Égalités de types références

Pour les types primitifs

Les valeurs sont comparées :

On a : a==c mais a !=b et b !=c

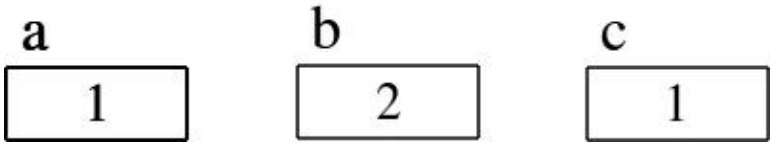

FIGURE 1 – java-logi-egal1.jpg

Pour les types références

Les références sont comparées :

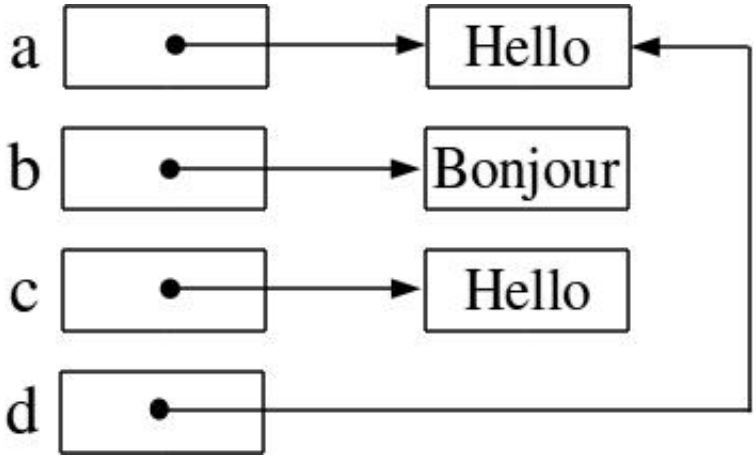

FIGURE 2 – java-logi-egal2.jpg

On a : a !=b, a !=c mais a==d

Cas particulier du type String

Le compilateur réutilise l'espace pour les littéraux de type String

```
String s1 = "Hello";
String s2 = "Hello";
String s3 = "Hel";
s3 = s3 + "lo";
System.out.println(s1==s2); // Vrai
System.out.println(s1==s3); // Faux
s2 = "Bye";
System.out.println(s1==s2); // Faux
```

Egalité de valeur : equals()

```
String s1 = "Hello";
String s2 = "Hello";
```

```
String s3 = "Hel";
s3 = s3 + "lo";
System.out. println (s1. equals(s2 )); // Vrai
System.out. println (s1. equals(s3 )); // Vrai
```

Ne teste pas que les références sont identiques mais bien que les valeurs référencées sont égales.

2 Exercices

Maintenant, mettons tout ça en pratique.

2.1 À vous de jouer...

N'oubliez pas nos quelques conseils pour vous guider dans la résolution de tels problèmes :

- il convient d'abord de bien comprendre le problème posé ; assurez-vous qu'il est parfaitement spécifié ;
- résolvez le problème via quelques exemples précis ;
- mettez en évidence les variables **«données »**, les variables **«résultats »** et les variables de travail ;
- n'hésitez pas à faire une ébauche de résolution en français avant d'élaborer l'algorithme définitif pseudo-codé ;
- déclarez ensuite les variables (et leur type) qui interviennent dans chaque algorithme ; les noms des variables risquant de ne pas être suffisamment explicites.
- Écrivez la partie algorithmique **AVANT** de vous lancer dans la programmation en Java.
- Demandez-vous si vous avez besoin de parcourir tout le tableau ou de sortir prématurément (si on a trouvé ce qu'on cherche par exemple).
- Pour la partie Java, dessinez l'arborescence des fichiers.
- **Écrivez le plan de tests en écrivant l'algorithme. Codez les tests après avoir écrit le code Java.**

Écrivez les algorithmes et codez les programmes Java correspondant qui

1. reçoit une fraction sous forme de chaine, et retourne la valeur numérique de celle-ci. Par exemple, si la fraction donnée est "5/8", l'algorithme renverra 0,625. On peut considérer que la fraction donnée est correcte, elle est composée de 2 entiers séparés par le caractère de division '/'.
2. reçoit le nom complet d'une personne dans une chaine sous la forme "nom, prénom" et la renvoie au format "prénom nom" (sans virgule séparatrice). Exemple : "De Groote, Jan" deviendra "Jan De Groote".

3. met un mot en «ou »au pluriel. Pour rappel, un mot en «ou »prend un «s »à l'exception des 7 mots bijou, caillou, chou, genou, hibou, joujou et pou qui prennent un «x »au pluriel. Exemple : un clou, des clous, un hibou, des hiboux. Si le mot soumis à l'algorithme n'est pas un mot en «ou », un message adéquat sera affiché.
4. vérifie si un mot donné sous forme de chaine constitue un palindrome (comme par exemple "kayak", "radar" ou "saippuakivikauppias" (marchand de savon en finnois))
5. vérifie si une phrase donnée sous forme de chaine constitue un palindrome (comme par exemple "Esope reste ici et se repose" ou "Tu l'as trop écrasé, César, ce Port-Salut !"). Dans cette seconde version, on fait abstraction des majuscules/minuscules et on néglige les espaces et tout signe de ponctuation.
6. reçoit en paramètre le tableau avion de n chaines et qui retourne un booléen indiquant s'il contient au moins un élément de valeur «pilote».

En java, n'oubliez pas d'écrire la javadoc et les tests de vos méthodes.