Séance 6 de travaux pratiques

1 Un programme complet

Affichage des arguments. (*?*) String programmeentier tableaudeString [q1]

Ecrivez un programme qui affiche les arguments qui lui ont été passés sur la ligne de commande, un par ligne.

Exemple :

```
$ java ArgsPrinting toto titi tata tutu
toto
titi
tata
tutu
```

. `ArgsPrinting.java`

Bonjour nom ! (*?*) String programmeentier [q2]

Compléter le programme `HelloName` qui prend en paramètre une chaîne de caractères `nom` et qui affiche `"Bonjour nom !"` (où `nom` est bien sûr remplacé par l'argument). Attention aux espaces !

. `HelloName.java`

Bonjour à chacun ! (*?*) String programmeentier tableaudeString fonction [q3]

1. Compléter le programme `HelloNameReturn` avec une fonction `String getHelloMessage(String name)` qui prend en paramètre une chaîne de caractères `name` et qui retourne `"Bonjour nom !"` (où `nom` est bien sûr remplacé par l'argument).

2. Utilisez la fonction précédemment définie pour que le programme affiche `"Bonjour nom !"`, pour chaque `nom` spécifié en argument du programme. Par exemple :

Par exemple, si le tableau est `{{".",".","*"}, {".", "!", "."}}`, la procédure affichera :

```
$ java HelloNameReturn Alice Bob Chloé
Bonjour Alice !
Bonjour Bob !
Bonjour Chloé !
$
```

. `HelloNameReturn.java`

Est-ce un nom de fichier ? (*??*) String programmeentier [q4]

Souvent, les noms de fichiers basiques ont la forme `nom.extension`, comme par exemple `notes.txt`, `rapport.tex`, `Prog.java`, etc. Evidemment, ils ne sont souvent pas aussi simplistes, car enrichis, de chiffres (`tp01.txt`), de plusieurs extensions (`archive.tar.gz`) ou autre.

Pour les besoins de l'exercice, on considèrera comme simple (voire simpliste) un nom de fichier composé d'une suite d'au moins une lettre, puis d'un point, puis d'une suite de 1 à 5 lettres. Notez que les lettres

peuvent être minuscules ou majuscules. Le point doit donc apparaître une et une seule fois, être précédé d'au moins un caractère, et suivi d'entre 1 et 5 caractères ; tous ces caractères doivent être des lettres.

1. Compléter le programme `IsSimpleFileName` avec une fonction `boolean isSimpleFileName(String name)` qui prend en paramètre une chaîne de caractères `name` et qui retourne `true` si `name` a une forme simple de nom de fichier, et `false` sinon.

Exemple de chaînes ayant une forme simple de nom de fichier :

```
essai.txt
virus.EXE
README.md
SuPeRPhOtO.Jpeg
IsSimpleFileName.java
HelloName.class
```

Exemple de chaînes n'ayant PAS une forme simple de nom de fichier :

```
extensionless
H4x@r.3x3
ab.cdefgh
ab12.io
ineed$.euro
.png
archive.tar.gz
Zip.
```

2. Utilisez la fonction précédemment définie pour que le programme affiche pour chaque chaîne passée en argument `oui` si c'est une chaîne qui a la forme simple d'un nom de fichier, ou `non` dans le cas contraire. Pour chaque argument, on affichera la chaîne correspondante suivie de `": "` puis de la réponse `oui` ou `non`, et d'une nouvelle ligne.

```
$ java IsSimpleFileName bonjour test.1 fichier.exe son.mp3 allO.txt PP.
bonjour: non
test.1: oui
fichier.exe: oui
son.mp3: non
allO.txt: oui
PP.: non
$
```

`. IsSimpleFileName.java`

Papier-Feuille-Ciseaux-Lézard-Spock (*???*) int String boolean [q5]

On considère un jeu où deux adversaires (ici la machine et l'utilisateur) choisissent puis présentent simultanément une figure parmi les cinq possibles, les règles de supériorité entre ces figures étant : Ciseaux coupe Feuille, qui recouvre Pierre, qui casse Ciseaux, qui décapite Lézard, qui empoisonne Spock, qui vaporise Pierre, qui écrase Lézard, qui mange Feuille, qui réfute Spock, qui tord Ciseaux.

Notes : en pratique, les figures seront codées par leurs initiales ('P' , 'F' , 'C' , 'L' , 'S'), et vous pourrez aux besoin vous aidez des fonctions annexes permettant de passer de ce codage à un codage par des entiers (0, 1, 2, 3, 4) et inversement.

1. Écrire une fonction `getFigure` qui prendra en paramêtre une chaîne de caractères contenant le code d'une figure et retourne le nom complet de cette figure.

2. Écrire une fonction `zuper` qui prend en arguments deux chaînes de caractères codant des figures et renvoie +1 si la première figure est supérieure à la seconde, -1 si elle lui est inférieure et 0 en cas d'égalité.

3. Écrire une fonction `tour` qui prend en paramêtre une figure `c1` et :

- choisit une fonction `c2` aléatoirement (à l'aide de la fonction `alea`) - affiche cette figure (sous la forme `"Figure : figure"` suivi d'un retour à la ligne) - affiche `"c1 gagne contre c2"`/`"c1 perd contre c2"`/`"égalité entre c` (en remplaçant `c1` et `c2` par leurs valeurs), suivi d'un retour à la ligne - renvoie +1 si c1 est supérieure à c2, -1 si elle lui est inférieure et 0 sinon.

4. Un match est constitué de plusieurs tours, le vainqueur étant celui des deux joueurs qui gagne le plus de tours. Écrire la fonction `main` de telle sorte qu'à chaque figure passée en paramêtre par le joueur (dans `args`) se joue un tour contre une valeur aléatoire (en utilisant `tour` avec ses affichages), et qu'à la fin soit affiché `"Vainqueur : X !"` où "X" est remplacé par "machine" ou "joueur" suivant le vainqueur (en cas d'égalité, on affichera "Match nul!").

Pour tester votre `main`, vous téléchargerez le fichier source, ajoutez-y les fonctions que vous avez définies et faîtes des essais à la main.

Pour aller plus loin : vous pourrez, dans la version à télécharger, vous servir de la fonction `getMove` fournie dans le fichier pour obtenir de façon interactive la figure choisie par le joueurs et améliorer ainsi le jeu. Vous pouvez aussi imaginer comment modifier votre code pour pouvoir tricher discrètement.

. `PFCLS.java`

2 Boucles while avancées

Fusion de tableaux triés (*???*) String [q6]

1.Écrivez une fonction `merge` qui prend deux tableaux triés `t1` et `t2` en argument, en renvoie un tableau trié contenant tous les éléments des deux tableaux (avec les doublons).

Indication : on utilisera une boucle while, et un indice par tableau.

2. Testez votre fonction à l'aide d'un main sur les tableaux `{1,3,4,7,9}` et `{2,5,6,7,8,9}` : le main affichera le tableau résultant, c'est-à-dire `{1, 2, 3, 4, 5, 6, 7, 7, 8, 9, 9}`

. `ArrayMerge.java`

Un code dont vous êtes le héros (*??*) bouclewhile tableaudeboolean [q7]

Les éditions « Aventures Livresques », spécialistes des histoires dont vous êtes le héros, vous ont embauché pour vous assurer que leur livres sont d'excellente facture. Plutôt que de vérifier s'ils contiennent du gluten, vous devez vous assurer que le lecteur imprudent ne peut pas tomber dans une boucle dans le scénario qui l'obligerait alors à continuer à vivre l'aventure jusqu'à ce que mort s'ensuive.

Quelque peu méfiants de vos connaissances en algorithmique, les éditions « Aventures Livresques » limitent pour l'instant votre vérification à la série « MiniMorveux » consacrée aux 3-5 ans. Heureusement pour vous, il n'y a en effet aucun choix à faire, et l'histoire est complètement linéaire.

Une histoire est donnée par un tableau d'entiers représentant chaque page, où la valeur de la case du tableau représente le numéro de la page suivante (indicé à partir de zéro), ou bien `-1` si l'histoire prend fin à cette page.

Compléter le programme `Browse` qui prend en paramètre une histoire `history` et renvoie `-1` si en suivant l'histoire à partir de la page `0` on finit par boucler, ou bien le numéro de la page sur laquelle s'arrête l'histoire sinon.

Vous pourrez intelligemment utiliser un tableau de la même longueur que le tableau `history` afin de vous souvenir, petit poucet digital native, si vous êtes déjà passé par une page donnée.

Complétez ensuite le `main` de votre programme pour qu'il prenne en argument l'histoire sur la ligne de commande sous forme d'une liste d'entiers (que vous convertirez donc vous-mêmes) représentant pour chaque page le numéro de la page suivante. Vous afficherez l'entier retourné par la fonction `browse` suivi d'un retour à la ligne.

. `Browse.java`

Test de primalité (*??*) int String boolean [q8]

1. Écrire une fonction `isPrime` qui prend en paramêtre un entier et teste si cet entier est un nombre premier. On rappel qu'un nombre est premier si et seulement si il a exactement deux diviseurs, à savoir 1 et lui-même. En particulier, 1 n'est pas premier.

2. Écrire alors une fonction `firstsPrimes` qui prend en paramêtre un entier `n` et renvoie la chaîne de caractères donnant les `n` premiers nombres premiers, sous la forme `"2, 3, 5, 7"` (pour n=4). On utilisera la fonction `isPrime` définie en 1.

3. Compléter le `main` afin d'afficher le résultat du test de votre fonction `firstsPrimes` pour n=20.

. `IsPrime.java`