Outils de manipulation de valeurs

Luis D. Pedrosa & Jean-Cédric Chappelier 2017

Cette semaine, nous vous demandons de créer une série d'outils permettant de manipuler les contenus associés aux clés tels que :

- concaténer les valeurs ;
- extraire une sous-chaîne d'une valeur ;
- rechercher une sous-chaîne dans une valeur.

Ces outils fonctionnent suivant le même schéma général :

1. demander, suivant les besoins, une ou plusieurs valeurs via sa/leur clé ;
2. effectuer les opérations nécessaires ;
3. stocker le résultat comme valeur d'une clé donnée.

Des exemples sont fournis ci-dessous.

NOTE : une fois n'est pas coutume, mais, le fait que nous vous demandons de coder ces outils dans trois fichiers sources séparés va nécessairement engendrer un peu de « copier-coller » (entre ces fichiers). Une meilleure alternative eût été d'augmenter la généricité du code demandé et de passer par des pointeurs sur fonctions, mais nous avons finalement volontairement abandonné cette (meilleure) solution car elle aurait finalement été trop lourde pour ce projet (vous demandant trop de travail au total).

I. `pps-client-cat`

Le but de l'outil `pps-client-cat` (à écrire dans `pps-client-cat.c`) est de concaténer les valeurs de plusieurs clés et stocker le résultat sous une nouvelle clé (ou d'écraser la valeur d'une clé existante). Par exemple :

```
$ ./pps-client-put
a
alpha
OK
b
beta
OK
^D
```

```
$ ./pps-client-cat
a
b
c
^D
OK

$ ./pps-client-get
c
OK alphabeta

$ ./pps-client-cat
a
d
e
^D
FAIL
```

Dans cet exemple, la concaténation « `alphabeta` » des valeurs « `alpha` » de la clé « `a` » et « `beta` » de la clé « `b` » a été stockée sous la clé « `c` ». La dernière concaténation échoue car la clé « `d` » n'existe pas.

L'outil `pps-client-cat` prend donc une ou plusieurs clés existantes en entrée (une par ligne), effectue autant de « *get* » pour obtenir les valeurs correspondantes et les concatène, sans caractère séparateur, pour enfin stocker ce résultat comme valeur associée à la clé reçue comme paramètre sur la dernière ligne de son entrée standard (voir exemple ci-dessus).

Cet outil répondra simplement `"OK\\n"` ou `"FAIL\\n"` en fonction que l'ensemble des opérations effectuées ont été un succès ou que l'une d'entre elle a échoué (voir exemple ci-dessus).

NOTE : cet outil permet donc aussi simplement d'effectuer la copie d'une valeur vers une autre clé ; comme par exemple :

```
$ ./pps-client-put
a
alpha
OK
^D

$ ./pps-client-cat
a
b
^D
OK
```

```
$ ./pps-client-get
b
OK alpha
```

II. pps-client-substr

Le but de l'outil `pps-client-substr` (à écrire dans `pps-client-substr.c`) est d'extraire une sous-chaîne d'une valeur et de stocker le résultat sous une nouvelle clé (ou d'écraser la valeur d'une clé existante). Par exemple :

```
$ ./pps-client-put
a
123456
OK
^D

$ ./pps-client-substr
a
3
2
b
OK
a
-4
3
c
OK
^D

$ ./pps-client-get
b
OK 45
c
OK 345
```

Dans cet exemple, la sous-chaîne de longueur 2 commençant à la position 3 (mesurée à partir de 0) de la valeur associée à la clé « `a` » est stockée sous la clé « `b` » puis la sous-chaîne de longueur 3 commençant à la position 4 mesurée à partir de la fin (le dernier élément a l'indice -1) de la valeur associée à la clé « `a` » est stockée sous la clé « `c` ».

L'outil `pps-client-substr` prend donc successivement (jusqu'à ce que son entrée standard soit fermée ou qu'une erreur survienne) :

- une clé ;
- une position (voir détails ci-dessous) ;

- une longueur ;
- une clé ;

et :

1. récupère la valeur associée à la première clé fournie ;
2. extrait la sous-chaîne correspondante comme expliqué ci-dessous ;
3. stocke cette sous-chaîne comme (nouvelle) valeur associée à la seconde clé fournie.

L'indication de la sous-chaîne recherchée se fait au moyen de deux entiers : position et longueur. La position est mesurée positivement à partir de 0 dans le sens de lecture ou négativement à partir de -1 dans le sens inverse à partir de la fin de chaîne ; la longueur est simplement le nombre de caractères désirés (toujours plus grand ou égal à 0), 0 indiquant la chaîne vide.

Exemple illustrant toutes les positions positives et négatives (sans leur signe) pour le mot « exemple », puis quelques exemples d'interprétation des deux paramètres position et longueur :

```
chaîne            : exemple
position positive : 0123456
position négative : 7654321
longueur          : 1234567
******************************

position longueur --> résultat

   0        2     --> ex
   1        1     --> x
   4        3     --> ple

  -1        1     --> e
  -2        2     --> le

   2        0     --> "" (chaîne vide)
  -2        0     --> ""

   5        3     --> FAIL
  -1        2     --> FAIL
```

L'outil `pps-client-substr` répondra simplement `"OK\\n"` ou `"FAIL\\n"` en fonction que l'ensemble des opérations effectuées ont été un succès ou que l'une d'entre elle a échoué. En cas d'échec, l'outil s'arrêtera (ne lira plus de nouvelle entrée). De plus, une longueur trop longue correspond aussi à un échec comme indiqué dans l'exemple ci-dessus. Autre exemple :

```
$ ./pps-client-put
a
```

```
123456
OK
^D

$ ./pps-client-substr
a
0
7
b
FAIL

$
```

NOTE : l'outil `pps-client-substr` permet donc aussi simplement d'effectuer la copie d'une valeur vers une autre clé ; comme par exemple :

```
$ ./pps-client-put
a
123456
OK
^D

$ ./pps-client-substr
a
0
6
b
OK
^D

$ ./pps-client-get
b
123456
OK
^D
```

III. `pps-client-find`

Le but de l'outil `pps-client-find` (à écrire dans `pps-client-find.c`) est de retourner la position d'une sous-chaîne (valeur associée à une clé) recherchée dans une valeur associée à une (autre) clé. Par exemple :

```
$ pps-client-put
a
123456123
```

```
OK
b
34
OK
c
67
OK
d
123
OK
^D

$ client-find
a
b
OK 2
a
c
OK -1
a
d
OK 0
```

L'outil `pps-client-find` prend successivement (jusqu'à ce que son entrée standard soit fermée ou qu'une erreur survienne) deux clés (une par ligne) et retourne la position de la valeur de la seconde clé dans la valeur de la première clé (voir exemple ci-dessus). La position retournée est celle la plus à gauche (sens de lecture), à partir de 0 (voir exemple ci-dessus). Si la sous-chaîne n'est pas trouvée, la position est -1 (voir exemple ci-dessus). Notez bien qu'ici (`pps-client-find`), on ne répond que des positions *positives* si la sous-chaîne est trouvée et que la valeur négative -1 (la seule possible ici) indique ici une *erreur* (contrairement à la signification des positions négatives dans `pps-client-substr`).

L'outil `pps-client-find` répondra simplement `"OK position\\n"` ou `"FAIL\\n"` en fonction que l'ensemble des opérations effectuées ont été un succès et ont conduit à la position `position` ou que l'une d'entre elle a échoué. Ne pas trouver la sous-chaîne recherchée **n'**est **pas** considéré comme un échec, mais retourne simplement la valeur -1 comme position. En cas d'échec, l'outil s'arrêtera (ne lira plus de nouvelle entrée).

IV. Conseil et rendu

Le travail de cette semaine ne sera pas évalué séparément, mais devra faire partie du second rendu intermédiaire du projet (délai : le dimanche 13 mai 23:59). Néanmoins, nous vous conseillons comme toujours de continuer à travailler

régulièrement et faire systématiquement des tags hebdomadaires (si votre travail correspondant est opérationnel) afin de profiter de nos comptes-rendus de tests automatiques. Le tag correspondant à cette semaine est `week09`.

Pour cette semaine, il faudrait mettre à jour tous les fichiers nécessaires (sans oublier le `Makefile`, mis à jour pour que les cibles `pps-client-cat`, `pps-client-substr` et `pps-client-find` soient aussi compilées par défaut) et ajouter les fichiers `pps-client-cat.c`, `pps-client-substr.c` et `pps-client-find.c` au répertoire `done/` de votre dépôt GitHub (de groupe).