Cours MOOC EPFL d'introduction à la programmation illustré en C++

Dernier devoir noté
Tableaux et chaînes de caractères

V. Lepetit, J.-C. Chappelier & J. Sam

du 1er novembre au 25 novembre 2013

1 Exercice 1 – Fermeture-éclair de tableaux

On s'intéresse ici à écrire un programme permettant d'entrelacer deux tableaux d'entiers, à une dimension.

Cela consiste à créer le tableau dont chaque élément est pris tour à tour dans un puis dans l'autre des tableaux :
- son premier élément sera le premier élément du premier tableau reçu en paramètre ;
- son deuxième élément sera le premier élément du second tableau reçu en paramètre ;
- son troisième élément sera le deuxième élément du premier tableau reçu ;
- etc.

Lorsqu'un des deux tableaux est épuisé, on continue à remplir en ne prenant alors que dans le tableau restant, jusqu'à épuisement de celui-ci.

Par exemple si le premier tableau est `{ 1, 2, 3 }` et le second `{ 4, 5, 6 }`, alors le tableau résultant sera `{ 1, 4, 2, 5, 3, 6 }`.

Autre exemple : si le premier tableau est `{ 1, 2, 3 }` et le second `{ 4, 5, 6, 7, 8 }`, alors le tableau résultant sera `{ 1, 4, 2, 5, 3, 6, 7, 8 }`.

A noter donc que l'entrelacement avec un tableau vide redonne un tableau identique. Par exemple si le premier tableau est `{ 1, 2, 3 }` et que le second est vide, alors le tableau résultant sera `{ 1, 2, 3 }`.

Ecrivez un programme C++ contenant une fonction `entrelace` (respectez strictement ce nom) qui prend deux tableaux dynamiques d'entiers en paramètre et re-

tourne un tableau dynamique d'entier, résultat de l'entrelacement des 2 tableaux reçus, tel que décrit ci-dessus.

Pour pouvoir être noté, votre programme devra contenir une fonction `main()`, quelqu'elle soit (même vide).

2 Exercice 2 – Cryptage

Jules César utilisait un système de codage très simple, qui consiste à remplacer chaque lettre d'un message par la lettre placée plusieurs rangs après dans l'ordre alphabétique. Par exemple, pour un décalage de 4, A devient E, B devient F, jusque Z qui devient D.

Il s'agit ici d'appliquer cette technique pour coder une chaîne de caractères. Vous écrirez pour cela un programme qui met en œuvre les traitements décrits ci-dessous.

2.1 Codage des caractères

Ecrivez un programme C++ contenant une fonction `code` (respectez strictement ce nom) qui prend un *caractère* et un entier en paramètres et retourne le caractère «décalé» correspondant si c'est une lettre majuscule ou minuscule, et retourne la même lettre sinon.

Par exemple, avec le décalage de `4` (second paramètre) :
- pour `'a'`, cette fonction retournera `'e'` ;
- pour `'A'`, elle retournera `'E'` ;
- pour `'Z'`, elle retournera `'D'` ;
- et pour `'!'`, elle retournera `'!'` (inchangé).

La façon de procéder est la suivante :
- créez une fonction `decale` (respectez strictement ce nom, sans accent) qui
  - prend trois paramètres : un caractère `c`, un caractère `debut` et un entier `decalage`,
  - tant que le `decalage` est strictement négatif, lui ajoute 26 ;
  - et retourne un caractère suivant la formule :
    ```
    debut + (((c - debut) + decalage) % 26)
    ```
    (Note : ceux qui compilent avec l'option `-Wconversion` auront ici un message d'alerte (*warning*) du compilateur, que l'on peut ignorer).
- soit `c` le caractère et `d` le décalage reçus par la fonction `code` :
  - si `c` est supérieur ou égal à `'a'` et qu'il est inférieur ou égal à `'z'` (on peut

comparer les caractères avec les mêmes opérateurs que les nombres), alors renvoyer le résultat de l'appel de `decale` sur `c`, à partir de `'a'` avec le décalage `d` ;
- si `c` est supérieur ou égal à `'A'` et qu'il est inférieur ou égal à `'Z'`, procédez de même mais en partant de `'A'` ;
- sinon renvoyez `c` inchangé.

2.2 Codage des chaînes

Ecrivez ensuite une fonction `code` (respectez strictement ce nom) qui prend en paramètres une *chaîne de caractères* et un entier, et qui retourne une nouvelle chaîne de caractères en appliquant la fonction `code` *précédente* à chacun des caractères de la chaîne reçue.

Cette fonction retournera par exemple :
- `Jycid qererxw` lorsqu'elle reçoit la chaîne `Fuyez manants` et un décalage de 4 ;
- `Laekf sgtgtzy` pour la même chaîne et un décalage de 6 ;
- `Bquav iwjwjpo` pour la même chaîne et un décalage de -4 ;
- `Ezid-zsyw zy qiw 3 glexw ix qiw 2 glmirw ?` pour `Avez-vous vu mes 3 chats et mes 2 chiens ?` et un décalage de 4.

2.3 Décodage des chaînes

Ecrivez une fonction `decode` (respectez strictement ce nom) qui prend en paramètres une *chaîne de caractères* et un entier, décode la chaîne reçue (pour le décalage reçu) et retourne la chaîne ainsi décodée.

Il suffit de remarquer que décoder consiste à coder avec le décalage opposé. Cela peut donc s'écrire en une seule instruction.

Pour pouvoir être noté, votre programme devra contenir une fonction `main()`, quelqu'elle soit (même vide).

Et n'oubliez pas de tester vos fonctions...

3 Exercice 3 – codes d'Elias

Cet exercice s'intéresse à construire un code binaire (suite de 0 et de 1) pour les nombres entiers, de sorte qu'on puisse les transmettre en séquence de façon non

ambiguë. Par exemple :
– si on choisit de coder l'entier 1 par la chaîne binaire `"1"` et l'entier 3 par la chaîne binaire `"11"`, ce code est ambigu : si un émetteur envoie la chaîne binaire `"11"` le récepteur ne sait pas s'il doit interpréter cette chaîne comme un seul 3 ou une séquence de deux 1 ;
– en revanche, si on code 1 par `"1"` et 3 par `"001"`, cette ambiguïté disparaît et ce code est un code non ambigu (au moins pour les valeurs 1 et 3).

L'avantage des codes non ambigus est de permettre d'envoyer des données sous une forme simple à décoder tout en étant concis. Le but de cet exercice est de vous faire écrire des fonctions pour pouvoir calculer un tel code, appelé code d'Elias, d'après le nom de son inventeur.

Nous aurons pour cela besoin de quatre étapes assez simples et vous demandons de faire cinq fonctions.

3.1 Fonction `convert`

La première fonction est simplement utilitaire nous servant à convertir un `int` en `char` suivant la convention suivante :
– si l'entier reçu en paramètre vaut 1, retourner le caractère `'1'` ;
– sinon retourner le caractère `'0'`.

Ecrivez un programme C++ contenant une fonction `convert` (respectez strictement ce nom) qui prend en paramètre un entier et retourne un caractère suivant la convention expliquée ci-dessus.

3.2 Fonction `Z0`

Le premier code dont nous avons besoin pour construire le code d'Elias d'un nombre est simplement son code binaire, appelé ici Z_0 :

n	1	2	3	4	5	6	7	8
$Z_0(n)$	1	10	11	100	101	110	111	1000

Pour construire ce code, il faut procéder de façon similaire à ce qui avait été fait en base 10 dans l'exercice 1 du devoir précédent, « sommes et produits », mais ici en base 2 :
– tant que n n'est pas nul (réfléchissez si c'est une boucle `while`, ou une boucle `do-while`) ajouter le caractère correspondant à `n % 2` (en utilisant la fonction `convert`), et diviser n par 2.

Créez pour cela une fonction `Z0` (respectez strictement ce nom) qui prend en paramètre un entier n et retourne une chaîne de caractères correspondant au code binaire de n tel que décrit ci-dessus.

Attention, l'ajout de chaque nouveau symbole (ici, un caractère retourné par la fonction `convert`) doit se faire *à l'avant* de la chaîne à retourner et non pas à sa fin.

Testez votre fonction avec les valeurs indiquées plus haut.

3.3 Fonction `Z0t`

On peut remarquer que le code Z_0 de n'importe quel entier commence toujours par un '1' (à gauche). Le code « Z_0 tronqué » correspond au code Z_0, sans ce '1' le plus à gauche :

n	1	2	3	4	5	6	7	8
$Z_{0t}(n)$		`0`	`1`	`00`	`01`	`10`	`11`	`000`

Notez qu'il est effectivement vide pour la valeur 0.

Créez une fonction `Z0t` (respectez strictement ce nom) qui prend en paramètre un entier n et retourne une chaîne de caractères correspondant au code « Z_0 tronqué » de n tel que décrit ci-dessus.

Testez votre fonction avec les valeurs indiquées plus haut.

3.4 Fonction `Z1`

Pour éviter les ambiguïtés du code Z_0, la première idée émise par Elias fut d'ajouter autant de zéros devant $Z_0(n)$ que sa longueur moins 1 :

n	1	2	3	4	5	6	7	8
$Z_1(n)$	`1`	`010`	`011`	`00100`	`00101`	`00110`	`00111`	`0001000`

Par exemple, le code Z_0 de 1 est `1` ; sa longueur est donc 1 et sa longueur moins 1, 0. Donc on n'ajoute rien devant et son code Z_1 sera également `1`.

Autre exemple : le code Z_0 de 7 est `111` ; sa longueur est donc 3 et sa longueur moins 1, 2. On ajoute donc deux zéros devant et son code Z_1 sera alors `00111`.

Créez une fonction `Z1` (respectez strictement ce nom) qui prend en paramètre un entier n et retourne une chaîne de caractères correspondant au code Z_1 de n tel que décrit ci-dessus. C'est plus simple à écrire en C++ qu'en français `;-)`.

Testez votre fonction avec les valeurs indiquées plus haut.

3.5 Fonction `Z2`

Le code Z_1 est non ambigu, cependant il n'est pas très concis. Le « truc » intelligent utilisé par Elias pour raccourcir le code fut de changer le codage de la longueur fait par une suite de zéros en le code Z_1 de cette longueur. Le code Z_2 de n est ainsi constitué de l'enchaînement du code Z_1 de la longueur de $Z_0(n)$ et du code « Z_0 tronqué » de n lui-même.

Par exemple, 7 est codé en `111` par Z_0, ayant ainsi une longueur de 3. Ceci donne le préfixe $Z_1(3) =$`011`, et le suffixe `11`, code « Z_0 tronqué » de 7. Le code Z_2 de 7 est alors `011111` (=`011`,`11`). Voici d'autres exemples :

n	1	2	3	4	5	6	7	8
$Z_2(n)$	`1`	`0100`	`0101`	`01100`	`01101`	`01110`	`01111`	`00100000`

Pour finir, créez une fonction `Z2` (respectez strictement ce nom) qui prend en paramètre un entier n et retourne une chaîne de caractères correspondant au code Z_2 de n tel que décrit ci-dessus. C'est plus simple à écrire en C++ qu'il n'y paraît !

Testez votre fonction avec les valeurs indiquées plus haut. Pour information, voici également les codes pour 255 :

```
Z0  : 11111111
Z0t : 1111111
Z1  : 000000011111111
Z2  : 00010001111111
```

Remarque : pour pouvoir être noté, votre programme devra contenir une fonction `main()`, quelqu'elle soit (même vide).

4 Exercice 4 – Tranches maximales d'un tableau à 2 dimensions

Il s'agit ici d'écrire un programme effectuant un certain nombre de traitements sur des tableaux dynamiques d'entiers *positifs* à deux dimensions. Le but final est de construire le tableau composé des lignes du tableau de départ qui comportent la plus grande somme d'éléments non nuls *consécutifs*.

Par exemple, pour le tableau

$$\begin{pmatrix} 2 & 1 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 1 & 3 & 0 & 0 \\ 0 & 2 & 2 & 0 \end{pmatrix}$$

on veut créer le tableau :

$$\begin{pmatrix} 1 & 3 & 0 & 0 \\ 0 & 2 & 2 & 0 \end{pmatrix}$$

car ces deux lignes (les deux dernières lignes du tableau de départ) ont une somme d'éléments non nuls consécutifs qui vaut 4, ce qui est la plus grande telle somme.

Notez bien que la plus grande somme d'éléments non nuls consécutifs n'est pas la plus grande somme de la ligne. Dans l'exemple ci-dessus la somme de la première ligne est 5, mais les 2,1 du début de ligne et le 2 de fin de lignes ne sont pas consécutifs (il y a un 0 entre le 1 et le 2) donc la plus grande somme d'éléments non nuls consécutifs de cette première ligne est 3, qui est plus petit que 4, somme d'éléments non nuls consécutifs des deux dernières lignes.

Notez également que si le tableau est vide ou ne contient que des 0, le résultat sera le même tableau que celui de départ (vide ou tout nul).

Pour résoudre ce problème apparemment assez compliqué, nous allons le décomposer en trois sous-tâches plus simples :

1. le calcul de la somme maximale d'éléments non nuls consécutifs d'une ligne donnée ;
2. le calcul d'un tableau contenant les numéros des lignes ayant la plus grande somme maximale d'éléments non nuls consécutifs ;
3. la création du tableau de réponse.

Notez que la première étape recherche un maximum par ligne (la plus grande somme d'éléments non nuls consécutifs) alors que l'étape 2 recherche un maximum sur toutes les lignes du tableau : le maximum des maxima précédemment calculés. Cela est plus compliqué à dire qu'à coder...

4.1 Somme maximale d'éléments consécutifs

Ecrivez un programme C++ contenant une fonction `somme_consecutifs_max` (respectez strictement ce nom) qui prend en paramètre un tableau dynamique d'entiers (à *une seule* dimension) et retourne la plus grande somme d'éléments non nuls consécutifs.

Par exemple :

– sur le tableau `{ 0, 2, 2, 0 }`, cette fonction retournera `4` puisque $2+2=4$;
– sur le tableau `{ 2, 3, 0, 0, 4 }` ainsi que sur le tableau `{ 4, 0, 2, 3 }`, cette fonction retournera `5` puisque $2+3=5$ est plus grand que 4 ;
– sur le tableau vide ou le tableau `{ 0, 0, 0, 0, 0 }`, cette fonction retournera `0`.

Pour écrire cette fonction, il sera utile d'avoir deux variables, une pour la somme en train d'être calculée et une pour la meilleure somme d'éléments non nuls consécutifs calculée jusqu'ici.

Il pourra également être utile de remarquer que dès qu'un élément de la ligne est nul la somme en train d'être calculée redevient nulle.

4.2 Lignes de somme maximale d'éléments consécutifs

Ecrivez ensuite une fonction `lignes_max` (respectez strictement ce nom) qui prend en argument un tableau dynamique d'entiers à *deux* dimensions et retourne un tableau dynamique de `size_t` à *une seule* dimension.

Le tableau retourné correspond à la liste des numéros de lignes où la somme maximale d'éléments non nuls consécutifs est atteinte.

Par exemple, pour le tableau

$$\begin{pmatrix} 2 & 1 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 1 & 3 & 0 & 0 \\ 0 & 2 & 2 & 0 \end{pmatrix}$$

cette fonction retournera le tableau `{ 2, 3 }` puisque la troisième et quatrième ligne (d'indices `2` et `3`, donc !) sont les deux lignes de ce tableau où la somme d'éléments non nuls consécutifs est maximale (elle vaut 4).

Pour un tableau vide, cette fonction retournera un tableau vide.

La méthode pour construire le tableau solution est assez simple : en partant d'un tableau vide, on ajoute le numéro de toute ligne dont la somme d'éléments non nuls consécutifs est la meilleure jusqu'ici.

Si par contre la somme d'éléments non nuls consécutifs de la ligne courante est strictement plus grande que la meilleure somme connue jusqu'ici, on vide le tableau solution (et l'on corrige la meilleure somme connue).

Il faudra faire attention à l'ordre de ces deux opérations et il sera bien sûr utile de faire appel à la fonction `somme_consecutifs_max`.

4.3 Tranches maximales

Pour finir, écrivez une fonction `tranches_max` (respectez strictement ce nom) qui prend en argument un tableau dynamique d'entiers à *deux* dimensions et retourne un tableau dynamique d'entiers à *deux* dimensions tel qu'expliqué au début de cet exercice.

Cela s'écrit assez simplement en utilisant la fonction `lignes_max` précédente.

Voilà, c'est terminé !

Pour pouvoir être noté, votre programme devra contenir une fonction `main()`, quelqu'elle soit (même vide).

N'oubliez pas de tester vos fonctions avec différents cas, comme par exemple celui-ci :

$$\begin{pmatrix} 2 & 1 & 0 & 2 & 0 & 3 & 2 & & \\ 0 & 1 & 0 & 7 & 0 & & & & \\ 1 & 0 & 1 & 3 & 2 & 0 & 3 & 0 & 4 \\ 5 & 0 & 5 & & & & & & \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & & \end{pmatrix}$$

qui donne :

$$\begin{pmatrix} 0 & 1 & 0 & 7 & 0 & & & \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & \end{pmatrix}$$

Pensez également à tester les cas particuliers comme un tableau vide, un tableau avec une ligne vide, un tableau où toutes les lignes sont identiques, un tableau sans aucun 0, ...