Série 7 :
Programmation *C* - Pointeurs (3/5) - chaînes de caractères, pointeurs sur fonctions

Buts

Le but de cette série d'exercices est de vous permettre de continuer à pratiquer les aspects de la programmation en C utilisant les pointeurs, en particulier les « chaînes de caractères » et les pointeurs sur fonctions.

Exercice 1 : retour sur l'explorateur de mémoire

Reprenez **l'exercice 3 de la série 4**, mais en l'écrivant en utilisant l'arithmétique des pointeurs, en affichant par exemple directement toutes les adresses (sans passer par un index) :

```
0x7fffb7ed88ac : 01010000  80 ('P')
0x7fffb7ed88ad : 00000000   0
0x7fffb7ed88ae : 00000000   0
0x7fffb7ed88af : 00000000   0
```

Exercice 2 : piles et parenthèses (pointeurs, niveau 3)

Buts

Utiliser des tableaux dynamiques (ou des listes chaînées si vous faites cet exercice la semaine prochaine), ou votre propre structure (à inventer), pour implémenter une structure de pile (empiler, dépiler, valeur du sommet, test de pile vide) et résoudre le problème du parenthésage à deux parenthèses (« **langage de Dyck** ») : est-ce qu'une expressions utilisant 2 sortes de parenthèses, comme par exemple « `(a + [ b * c * ( d + e ) - 3 ] ) * ( x - [ 7 * b ] + 3)` » est correctement parenthèsée ou non ?

Indications

Les indications sont progressives, aussi, dès que vous pensez en savoir assez, essayez de faire l'exercice par vous-même.

1. On veut implémenter une pile. Qu'est-ce qui définit une pile ? Son sommet (le seul élément auquel on peut accéder) et les 4 fonctions : empile, dépile, lecture du sommet et test si la pile est vide.

   On vous demande d'implémenter la pile à l'aide de `vector`. Il vous suffit donc de définir où mettre le sommet : en tête ou en queue du `vector` ?

2. Que se passe-t-il quand on empile un nouvelle élément ?
   Il devient le nouveau sommet.

   Si on choisit de représenter le sommet de la pile par la premier élément du vecteur (élément 0) alors quand on empile un nouvel élément il faudra le mettre en 0 et déplacer tout le reste.
   Ce qui n'est pas une bonne solution !

   Par contre si on choisit de représenter le sommet par le dernier élément du vecteur, empiler consiste alors juste à ajouter un élément au vecteur.

3. Une pile sera donc représentée par un `vector`, le sommet de la pile étant le dernier élément du `vector`.

   Avec cette implémentation, empiler un élément revient à l'ajouter au `vector`, dépiler revient à enlever le dernier élément du `vector`, lire le sommet revient à lire le dernier élément du `vector` et tester si la pile est vide à tester si le `vector` est vide.

4. Concernant l'application « test de parenthésage », l'algorithme est le suivant :
   - **Tant qu'**il y a un caractère à lire
     - **Si** le caractère lu est `(` ou `[`, l'empiler
     - **Sinon, Si** c'est `)` ou `]`
       - **Si** la pile est vide, **Retourner :** *faux*
       - **Sinon** lire le sommet de la pile
         - **Si** le caractère lu et le sommet de la pile « correspondent » (se ferment l'un

l'autre, de la même famille)
        - dépiler
    - **Sinon**
        - **Retourner :** *faux*
- **Retourner :** la pile est vide ?

Exemple d'application

```
Entrez une expresssion parenthésée : ([])
  -> Erreur
Entrez une expression parenthésée : ([([()[]])][[()]]())
  -> OK
Entrez une expression parenthésée :
```

Note : Le programme se termine lorsque la chaîne saisie est vide.

Exercice 3 : piles et notation polonaise inverse (niveau 3)

Description

On veut faire un programme qui interprète les opérations arithmétiques en notation polonaise inverse (c'est-à-dire notation « postfixée »).

Cette notation consiste à donner **tous** les arguments avant l'opération.

Par exemple : `4+3` s'écrit `4 3 +`. De même : `(4 * 5) + 3` s'écrit `4 5 * 3 +`

Notez qu'avec ce système de notation il n'y a pas besoin de parenthèse. Il suffit d'écrire les opérations dans le bon sens.

Par exemple : `(4+3)*(3+2)` s'écrit `4 3 + 3 2 + *` alors que 4 + (3*3) + 2 s'écrit `4 3 3 * + 2 +`

Méthode

L'algorithme pour effectuer des opérations données en notation postfixée est le suivant, où `P` est une pile :

```
Tant que lire caractère c
  Si c est un opérande
     empiler c sur P
  Sinon
     Si c est un opérateur
        y <- sommet(P)
        dépiler P
        x <- sommet(P)
        dépiler P
        empiler le resultat de "x c y" sur  P
```

À la fin de cet algorithme, le résultat est au sommet de `P`.

À faire

Dans cette série nous allons simplement nous intéresser aux opérations sur les chiffres : les seuls opérandes seront les chiffres de `0` à `9`.
(Par exemple : `14+` veut dire `1 + 4` [on pourra bien entendu aussi noter avec des espaces : `1 4 +`])

1. Reprenez vos piles de l'exercice 3 ci-dessus et changez le type des éléments de `char` à `int`.
2. Prototypez puis définissez la fonction `eval` qui prend en entrée une chaîne de caractères et renvoie un entier, résultat de l'expression postfixée contenue dans la chaîne.

   Cette fonction devra implémenter l'algorithme ci-dessus, et utilisera donc une pile d'entiers.

3. Dans la fonction `main`, lisez une chaîne de caractères au clavier (correspondant à une opération arithmétique en notation postfixée) et évaluez-là à l'aide de la fonction précédente, puis affichez le résultat.

   La fonction `main` bouclera tant que la chaîne entrée n'est pas vide (voir l'exemple ci-dessous).

Indications

a. Pour tester sur un caractère (`char`) `c` est un chiffre, faire :

```
if ((c >= '0') && (c <= '9'))
```

Note : on peut aussi utiliser la fonction standard `isspace((int) c)` définie dans `ctype.h`.

b. Pour convertir un caractère `c` représentant un chiffre en sa valeur entière (par exemple convertir `'3'` en `3`), faire :

```
(int)(c - '0')
```

Exemple de déroulement

```
Entrez une expression à évaluer : 8 6 2 - / 3 +
 -> résultat : 5
Entrez une expression à évaluer : 4 3 + 3 2 + *
 -> résultat : 35
Entrez une expression à évaluer : 4 3 3 * + 2 +
 -> résultat : 15
Entrez une expression à évaluer :
```

Exercice 4 : algorithmes de Needleman-Wunsch (algorithme de Viterbi sur le graphe d'édition) et LCS (niveau 2)

[adapté par T. Coppey (2010) sur la base d'un ancien sujet d'examen.]

La comparaison et l'alignement de séquences de caractères est une tâche fondamentale dans plusieurs domaines d'applications, notamment en reconnaissance vocale, traitement d'images et bio-informatique. Le but de cet exercice est d'implémenter l'algorithme de Needleman-Wunsch (qui est un algorithme de Viterbi), utilisé par exemple dans le BioWall que vous avez pu observer toutes les semaines en entrant dans les salles d'exercices IN.

La première partie de l'algorithme consiste à évaluer la comparaison entre deux chaînes de caractères de tailles différentes, en minimisant le nombre d'insertions de « trous » entre deux caractères d'une même chaîne. En pratique, une matrice `M` de taille `(m + 1) × (n + 1)` est construite, où m et n sont les nombres de caractères des deux chaînes à comparer. Un « trou » est caractérisé par un déplacement horizontal ou vertical dans la matrice. On assigne alors à chaque cellule $M_{i,j}$ de la matrice un score défini par les équations suivantes :

$$M_{i,0} = M_{0,j} = 0 \quad \text{pour } 0 \leq i \leq m \text{ et } 0 \leq j \leq n$$

$$M_{i,j} = \max\{ M_{i-1,j-1} + S_{i,j},\ M_{i,j-1} - 2,\ M_{i-1,j} - 2 \} \quad \text{pour } 1 \leq i \leq m \text{ et } 1 \leq j \leq n$$

avec

$$S_{i,j} = \begin{cases} 2 & \text{si } M_i = M_j \\ -1 & \text{si } M_i \neq M_j \end{cases}$$

où M_i désigne le $i^{ème}$ caractère de la première chaîne et M_j le $j^{ème}$ caractère de la 2è chaîne.

Le but est d'implémenter en C une fonction permettant de construire une telle matrice à partir de deux chaînes de caractères introduites par l'utilisateur. Chaque cellule devra, en plus de son score, conserver la/une des2 cellule(s) parente(s), pour laquelle le maximum de l'équation (1) a été atteint (voir figure 1).

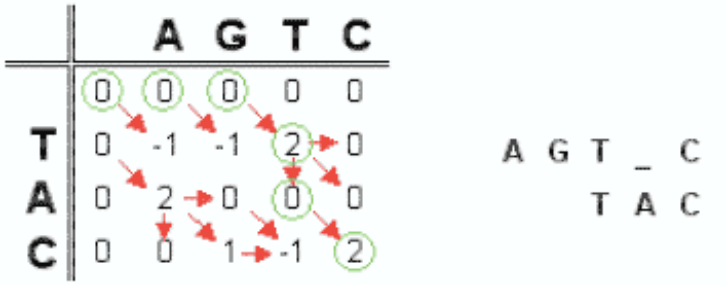

Fig. 1 – Exemple de matrice. Les flèches représentent les cellules parentes.

La seconde partie de l'algorithme permet de retrouver un alignement correct. Pour cela, il faut rechercher un chemin en partant de la cellule $M_{m,n}$ et arrivant à la cellule $M_{0,0}$, en utilisant à chaque fois la cellule parente stockée.

Pour implémenter cet algorithme, vous procèderez par étapes:

1. Dans le fichier `needle.h` décrivez les structures de données:
   - Pour comparer 2 chaînes `s1` et `s2` de longueurs respectives `l1` et `l2`, vous aurez besoin de créer un tableau de `(l2 + 1) x (l1 + 1)` cellules (en plaçant s1 horizontalement et s2 verticalement).
   - Chaque cellule contient une valeur (numérique) et une référence vers la cellule prédécesseur (il n'y a que 3 possibilités: immédiatement à gauche, en haut ou en diagonale).

2. Ecrivez ensuite dans `needle.c` une fonction `computeTable` qui prend en entrée les deux chaînes de caractères et qui construit la table en affectant à chaque cellule un prédécesseur et une valeur selon les équations précédentes :
   - Pour la première ligne, le prédécesseur est à gauche et la valeur 0.
   - Pour la première colonne, le prédécesseur est en haut et la valeur 0.
   - Remplir ensuite le tableau selon les équations précédentes, en mémorisant le prédécesseur

3. Ecrivez une fonction `extractSolution` qui prend en entrée la table et calcule le chemin de $M_{0,0}$ à $M_{m,n}$ en utilisant le prédécesseur de chaque cellule (Indication: vous devrez commencer à calculer le chemin par la fin puis l'inverser).

4. Vous pourrez ensuite tester votre programme à l'aide de la fonction

```
void afficheSolution(Solution* sol, const char* s1, const char* s2) {
 int i,j;
 for (i=0,j=0; i<sol->size; ++i) {
  switch (sol->dirs[i]) {
   case DirDiag: /*continued*/
   case DirHorz: printf("%c",s1[j]); ++j; break;
   case DirVert: printf("_");
  }
 }
 printf("\n");
 for (i=0,j=0; i<sol->size; ++i) {
  switch (sol->dirs[i]) {
   case DirDiag: /*continued*/
   case DirVert: printf("%c",s2[j]); ++j; break;
   case DirHorz: printf("_");
  }
 }
 printf("\n");
}
```

   Qui vous donnera pour les valeurs:

```
s1 = "Bonjour monsieur, quelle heure est-il à votre montre ?"
s2 = "Bonne journée madame, que l'heureuse fillette vous montre le chemin"
```

   le résultat suivant:

```
Bon___jour____ monsieur, quelle heure est-_il à__ votre montre ?________
Bonne journée madame__, que l'_heureu_se fillette vous_ montre le chemin
```

5. **Optionnel**: vous pouvez à présent aisément remplacer l'algorithme de Needleman-Wunsch par l'algorithme LCS (Longest Common Subsequence) si vous le souhaitez, remplacez simplement la formule par:

$$M_{i,j} = \begin{cases} M_{i-1,j-1} + 1 & \text{si } x_i = y_j \\ \max(M_{i-1,j}, M_{i,j-1}) & \text{si } x_i \neq y_j \end{cases}$$

Dernière mise à jour : $Date: 2013/01/11 13:26:44 $ ($Revision: 1.18 $)