Série 13: Entrée-Sortie conversationnelle

Lien avec le MOOC Initiation à la Programmation (en C++)

Exercice complémentaire (ExC)

ExC 12 : reprendre les exemples de code vu en classe inversée cette semaine (niveau 0-1)

Compilez, exécutez et explorez les programmes fournis sur moodle pour vous familiariser avec les concepts vus cette semaine

ExC13 : Vérification des entrées conversationnelles (niveau 1)

Ecrire un programme qui lit sur l'entrée standard des nombres entiers signés et les additionne dans un accumulateur tant qu'on n'a pas entré le signal EOF avec Ctrl-D. Quand ce signal est reçu on affiche la moyenne des valeurs lues.

L'opération de lecture doit détecter chaque échec de lecture pour afficher un message d'information et vider le buffer d'entrée jusqu'à la fin de ligne.

Tester le programme en redirigeant un fichier texte contenant ces données :

```
10  -20 5
11.5 150
a25
-4 1x 5
0  3
```

Quelle est la moyenne affichée ?

ExC14 : Arithmétique sur les pointeurs et tableaux (à-la-C, array, vector à un seul indice) / (niveau2)

Cet exercice illustre les opérations permises sur les pointeurs, en particulier lorsqu'ils pointent sur l'espace occupé par un tableau, que ce soit un tableau à-la-C, un array ou un vector à un indice.

Une illustration partielle a été donnée en cours dans le slide 6 du premier cours sur les pointeurs (topic7).

De même l'exercice ExC10 de la semaine dernière a illustré l'opération d'addition sur les pointeurs :

- **a_ptr** pointe sur les éléments d'un tableaux à-la-C de **char**
- **p** pointe sur les éléments d'un tableaux à-la-C de **const char***

En arithmétique sur un pointeur, l'*unité est la **taille de l'élément pointé,** quel que soit son type* :

- **a_ptr + 1** pointe sur l'élément **char** suivant dans le tableau de **char**
- **p + 1** pointe sur l'élément **const char*** suivant dans le tableau de **const char***

On peut aussi soustraire une quantité entière à un pointeur ; cela le décale vers les éléments précédents du tableau. Enfin, ce qui est valable pour les éléments d'un tableau à-la-C est aussi valable pour les éléments d'un **array** et d'un **vector** à un seul indice.

1) écrire un programme qui déclare et initialise un **array** comme suit :

```
array<int,5> tab ={1,2,3,4,5} ;
```

Initialiser un pointeur **int* p** avec l'adresse du premier élément de tab puis construire une boucle qui affiche l'ensemble des valeurs de **tab** en utilisant seulement une indirection sur le pointeur **p** et en incrémentant ce pointeur à chaque passage dans la boucle.

On se donne la contrainte supplémentaire suivante : ne pas utiliser de compteur entier pour déterminer la condition de la fin de boucle. A la place, initialiser un autre pointeur **int* pfin(&tab[5]) ;**

Piste à suivre: utiliser l'opérateur d'égalité ou de différence sur **p** et **pfin** pour exprimer la condition pour rester dans la boucle.

2) On peut aussi effectuer les opérations suivantes sur *2 pointeurs qui* ***pointent sur un même tableau:***

- **comparer 2 pointeurs avec <, >, <=, >= .**
- **calculer la différence de deux pointeurs :**
  - donne le *nombre d'éléments du tableau* entre les deux adresses.
  - Attention : cette valeur est une quantité signée.

Transposer l'algorithme de recherche dichotomique pour déterminer si une valeur lue au clavier est présente dans l'array **dicho,** en travaillant seulement avec des pointeurs :

```
array<int,15> dicho ={-2,3,5,6,9,11,15,21,24,32,36,45,49,51,66} ;
```

ExC15 : Une petite calculatrice (niveau 2)

Dans cet exercice nous voulons indiquer une opération arithmétique élémentaire sur la ligne de commande d'un programme appelé **calcul** qui affiche le résultat ou un message d'erreur.

Le programme **calcul** doit être appelé comme suit :

```
./calcul  opérande_gauche  opérateur  opérande_droit
```

Il doit recevoir 4 arguments. Les opérations acceptées sont l'addition, la soustraction, la multiplication et la division. Elles sont exprimées à l'aide de leur symbole habituel sauf la multiplication qui utilise le symbole '**x**' au lieu d'un '*****' car l'étoile est interprété comme étant « l'ensemble des fichiers du répertoire » lorsqu'il est fourni sur la ligne de commande.

Il faut aussi détecter le cas de division par zéro et afficher un message d'erreur dans ce cas.

Exemples :

./calcul 789 + 486.5
Résultat: 1275.5

./calcul 789 o 486.5
ERREUR: opération invalide.

./calcul 789.+ 486.5
Le nombre d'arguments n'est pas correct!!

Remarques :

Il ne faut pas oublier de séparer les arguments **par un espace** pour obtenir le bon nombre d'arguments.

Les arguments de programme sont de type chaine de caractères ; utilisez le tutoriel qui explique la conversion d'une chaine vers une valeur numérique.

ExC16 : convertisseur decimal / hexadécimal / octal (niveau 1)

Ecrire un programme qui contient une boucle convertissant une valeur entière positive lue au clavier d'une base **source** à une base **destination**. On utilise pour cela les manipulateurs sur la lecture avec **cin** et l'écriture avec **cout**.

Plus précisément, les bases **source** et **destination** sont codées avec un entier pouvant prendre seulement les valeurs 0, 1 ou 2 correspondant à l'un des symboles suivant (cf cours Topic2) :

```
enum Base {DEC, OCT, HEX} ;
```

Les bases sont initialisées avec DEC.

A chaque passage dans la boucle de conversion, on lit successivement la base **source**, la base **destination** et une *valeur entière exprimée dans la base source*.

Le programme affiche

… sur 4 colonnes de largeur égale et constante, les informations suivantes :

Base source : valeur source, base destination : valeur destination

… ou un message d'erreur si l'input ne correspond pas à ce qui est demandé.

La boucle se poursuit tant qu'on donne une valeur strictement positive à convertir.

Exemples :

```
1 2 20
Octal:        20          Hexadecimal : 10
2 1 10
Hexadecimal : 10          Octal :        20
0 0 0
```