Cahier de TP n°3

TP 6 & TP 7 : Les pointeurs

Quelques manipulations et vérifications

On cherche par cet exercice à vérifier que la mémoire est bien organisée conformément à ce qui a été traité en cours. Le plus simple pour cela est de définir des variables de types différents (plusieurs de chaque type) et d'afficher leurs adresses (on peut utiliser, pour afficher une adresse, le format %ld utilisé pour les entiers long en temps normal.

Ecrivez un tel programme et vérifiez que les adresses des variables se comportent comme indiqué dans le cours.

Tableaux et pointeurs

Deuxième vérification : un tableau est un pointeur.

Ecrivez un programme qui : définit un tableau de 50 entiers nommé tab, une taille utile et un pointeur vers des entiers nommé ptr.

Ce programme doit afficher les valeurs de ptr et de tab (ptr n'est pas initialisé, vous aurez ainsi l'occasion de voir la valeur contenue dans une variable non initialisée).

Ce programme doit ensuite initialiser ptr à NULL, ,et afficher ptr.

Enfin, on cherche à vérifier que les affectations entre tableaux et pointeurs ne se font pas n'importe comment :

Essayez d'affecter le tableau tab avec le pointeur ptr. Quel message le compilateur vous affiche-t-il ?

Selon ce message, devrait-il être possible d'affecter le pointeur avec le tableau ?

Essayez enfin d'affecter le pointeur ptr avec le tableau tab, puis affichez les valeurs de ptr et de tab. Que constatez-vous ?

Ultimes vérifications :

a) Reprenez le programme suivant vu en TD, et traduisez le en C de manière à ce qu'il affiche les expressions demandées. Les valeurs obtenus correspondent-elles à ce que vous attendez ? Pour que les résultats soient plus simples à interpréter, il devra afficher, avant les valeurs des expressions, les valeurs de : a, p et &p.

```
caractere a[9] ← {12, 23, 34, 45, 56, 67, 78, 89, 90};
caractere *p;
p ← a;
```

Quelles valeurs ou adresses fournissent ces expressions ? On supposera que le tableau `a` est stocké à l'adresse 3000 en mémoire de l'ordinateur. Un caractère occupe un octet.

```
a)    *p+2
b)    *(p+2)
c)    &p+1
d)    &a[4]-3
e)    a+3
f)    &a[7]-p
g)    p+(*p-10)
h)    *(p+*(p+8)-a[7])
```

b) Reprenez en C l'exercice suivant pour vérifier les résultats obtenus en TD :

```
programme arith_pointeurs


reel val_a, val_b, val_c;
reel *pdoub;
reel *qdoub;

val_a ← 5.0;    // attention il y a une modification par rapport au
                // TD
val_b ← 3.1415;
val_c ← 1.E-50;

qdoub ← &val_b;
pdoub ← qdoub;

pdoub ← p_doub-1;

afficher(qdoub-pdoub,"\n");

qdoub ← qdoub+1;

afficher(*qdoub,"\n");

*qdoub ←val_a;

*pdoub ← (*pdoub)+1;

afficher(qdoub-pdoub," ", *pdoub,"\n");


*qdoub ← *pdoub;
pdoub ← pdoub+1;

afficher(qdoub-pdoub," ", *pdoub," ", *qdoub,"\n");
```

Allocation dynamique et applications

Taquinons le système d'exploitation

Ecrivez un programme qui détermine la taille maximum d'une zone de mémoire demandée grâce à la commande `new`. Calculez et affichez le nombre de Mo ou Go correspondant à cette taille. Pour réaliser ce programme, vous aurez besoin d'utiliser une variable dont le type est `unsigned long` au lieu de `long`.

Ces résultats vous semblent-ils corrects ? Que pouvez-vous en conclure ?

Les blagues les plus courtes…

…ne sont pas forcément les meilleures, nous allons continuer à voir ce qu'un ordinateur est capable de supporter. Combien d'allocations de 10 Mo peut-on faire à la suite sans saturer le système ? Comparez le résultat avec celui de l'exercice précédent.

Pour cet exercice, vous pouvez ouvrir le gestionnaire de tâches de windows "task manager" et regarder dans l'onglet "performance" l'usage qui est fait de la mémoire. Lorsque vous fermez la fenêtre DOS de votre programme, que se passe-t-il ? Qu'en concluez-vous ?

Vérification sur les allocations dynamiques

Allocation 1D

Ecrivez un programme qui effectue une allocation dynamique d'un tableau à une dimension et le remplit avec des valeurs saisies par l'utilisateur (vous choisirez le type des éléments stockés, cela n'a pas d'importance).

Améliorez ce programme pour qu'il trie le tableau (uniquement avec la notation *) dans l'ordre que vous voulez (croissant ou décroissant), et qu'il y insère une valeur directement à sa bonne place.

Allocation 2D

Ecrivez un programme qui réalise une allocation à deux dimensions pour un tableau 2D de dimensions respectives n et p (n lignes et p colonnes), où n et p seront saisis par vos soins. Il faudra bien vérifier que n et p sont strictement positifs.

La variable que vous utiliserez sera donc du type : `long **point_point;`

En utilisant les notations avec les crochets, remplissez le tableau ainsi obtenu alternativement avec des 0 et des 1, puis affichez son contenu.

Pour terminer ce cahier de TP, nous traitons quelques thèmes abordés en TD, de manière à ce que vous ayez de manière concrète le résultat de ces programmes.

Encore des caractères….

Toujours avec des caractères, on souhaite gérer un dictionnaire contenant un nombre variable de mots, chaque mot ayant une longueur variable. Le dictionnaire stockera au maximum 1000 mots.

Au départ, le dictionnaire stocke uniquement le mot suivant : "*Abécédaire*".

La contrainte principale de ce dictionnaire est qu'il doit utiliser au mieux la mémoire : aucun octet ne doit être gâché !

Ecrivez la définition de la variable, nommé dico par exemple, qui stockera le dictionnaire.

Ecrivez les différentes initialisations qui ont été nécessaires à ce que le dictionnaire se trouve dans son état initial (contenir le mot "*Abécédaire*" uniquement).

Ecrire un programme qui : saisit un mot et indique si ce mot est contenu dans le dictionnaire.

Améliorer ce programme pour y ajouter successivement les fonctionnalités suivantes :

Saisir et ajouter un nouveau mot

Supprimer un mot

Afficher tous les mots

Trier les mots par ordre alphabétique

Recherche si le dico est trié par ordre alphabétique

Afficher tous les mots commençant par une lettre donnée

Thème de synthèse : le triangle de Pascal

Le triangle de Pascal est un tableau particulier qui stocke un ensemble de coefficients entiers que l'on appelle également : coefficient du binôme.

Dans sa ligne *i*, ce tableau stocke tous les coefficients de la forme développée de la formule : $(a+b)^i$

Exemples :

$(a+b)^0 = 1$: coefficient : 1

$(a+b)^1 = 1.a + 1.b$: coefficients 1, 1

$(a+b)^2 = 1.a^2+2.ab+1.b^2$: coefficients 1, 2, 1

En prenant les exemple des puissances 3 et 4, écrivez le triangle de Pascal et établissez une relation simple entre les coefficients permettant de passer d'une ligne à l'autre.

Avec des tableaux statiques

Ecrire un programme qui calcule et affiche les coefficients du triangle de Pascal jusqu'à un certain rang *n* en utilisant un tableau statique.

Avec des tableaux dynamiques

Ecrire un programme qui calcule et affiche les coefficients du triangle de Pascal jusqu'à un certain rang *n* en utilisant un tableau dynamique qui n'utilise que la mémoire strictement nécessaire à son stockage.

En fait, le coefficient situé à la colonne *j* de la ligne *i* correspond au nombre de tirages possibles de *j* éléments (sans remise) parmi *i*. Jusqu'à quel rang devriez-vous calculer le triangle de pascal pour obtenir le nombre de combinaisons possibles de tirage au loto ?

Ce nombre est également donnée par : i!/ [(i-j)! × j!] : écrivez un programme qui calcule cette quantité en essayant de simplifier la formule. De toute manière, un ordinateur ne peut pas calculer une factorielle supérieure à 25 !. Le programme que vous écrivez devra être capable de calculer 49 ! / [(49-6)! × 6!]

Thèmes de synthèse : la suite de Fibonacci

Le mathématicien italien Leonardo Fibonacci (1175 – 1240) a mis au point une formule mathématique pour calculer, de manière sommaire, l'évolution d'une population (à l'origine de lapins, pour pouvoir en contrôler le développement) en indiquant que :

Un couple de lapins 'adulte' donne naissance à un couple de lapins 'jeune'

A la génération suivante, le couple de lapin 'jeune' devient un couple de lapin 'adulte', capable de se reproduire. La formulation mathématique est une suite dont voici la définition :

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ \text{Pour } n \geq 2,\ F_{n+1} = F_n + F_{n-1} \end{cases}$$

Ce type de définition se programme en général grâce à la récursivité, que nous aborderons en fin d'année, ou encore par un simple calcul de suite, comme nous en avons déjà rencontré.

Ecrire un programme calculant n termes de la suite de Fibonacci en utilisant un tableau à 1 dimension n'utilisant que la place nécessaire à son stockage en mémoire.