Algorithmique et Programmation 1
IMAC 1ere année

TP 4

Tableaux et pointeurs

Dans cette séance de travaux dirigés, les points suivants sont abordés :
— les tableaux,
— les pointeurs,
— l'utilisation des pointeurs pour qu'une fonction transmette plusieurs valeurs,
— l'utilisation de valeurs de retour comme codes d'erreurs.

Comme lors des séances précédentes, pensez à faire régulièrement des sauvegardes et à documenter votre code. Pensez également à régyulièrement tester vos code. Dans ce TP on aborde les tableaux et les pointeurs.

Exercice 1. (Manipulation des tableaux)

Cet exercice est en grande partie une reprise de l'exercice 5 du TP3. Ceux qui ont réussi à le faire peuvent passer directement à l'exercice 2.

1. Écrire une fonction `void afficheTab(int tab[], int taille)` affichant les valeurs du tableau `tab` de taille effective `taille`.
2. Écrire une fonction `void remplitTab(int tab[], int taille)` demandant à l'utilisateur de remplir le tableau `tab` de taille effective `taille`.
3. Écrire une fonction `int sommeTab(int tab[], int taille)` calculant et renvoyant la somme des valeurs de `tab`.
4. Écrire une fonction `void incrementeTab(int tab[], int taille)` incrémentant toutes les valeurs contenues dans `tab`.
5. Si vous ne l'avez pas déjà fait, testez vos fonctions.

Exercice 2. (Rencontre avec les pointeurs)

Cet exercice a pour but de vous permettre d'effecter des manipulations de base (déclaration, affectation, affichage, ...) sur les pointeurs. Vous effectuerez ces manipulations dans une fonction `main()`.

1. Définir une variable `n` de type `int` que vous initialisez à `1`.
2. Déclarer une variable `p` de type "pointeur sur int".
3. Initialiser `p` en y stockant l'adresse de `n`.
4. Afficher l'adresse de `n`. Vous effectuerez cet affichage de deux façons différentes : une en utilisant la variable `n` mais pas la variable `p` et réciproquement.

5. Changer la valeur de `n` en utilisant uniquement la variable `p`.

Exercice 3. (Pointeurs en paramètres de fonction)

Pour cet exercice, vous commencerez par définir et initialiser un entier `n` dans une fonction `main()`.

1. Écrire une fonction `incremente(int m)` incrémentant un entier `m` passé en paramètre puis affichant sa valeur et son adresse.
2. Appeler la fonction `incremente(n)` dans le `main` puis afficher la valeur et l'adresse de `n` après l'appel.
3. Écrire une foncion `incrementeParAdresse(int *p)` incrémentant la valeur pointée par `p`. Appeler cette fonction dans le `main()` en lui passant en argument l'adresse de `n`, puis afficher `n` après cet appel.
4. On souhaite définir une fonction `echange` servant à échanger deux valeurs entières. Quels doivent être les types des paramètres de cette fonction ? Écrire la fonction `echange` et la tester dans le `main()`.

Exercice 4. (Les pointeurs pour calculer plusieurs valeurs)

On a vu dans l'exercice précédent que les pointeurs permettent de modifier des variables du programme. Le but de cet exercice est d'en déduire comment cela permet à une fonction de calculer plusieurs résultats et de transmettre plusieurs fonctions au programme appelant.

1. On souhaite écrire une fonction `quotientEtReste` prenant en arguments deux entiers `diviseur` et `dividende` et calculant le quotient et le reste dans la division enclidienne de `dividende` par `diviseur`. Le quotient et le reste doivent pouvoir être stockés dans deux valeurs entières `quotient` et `reste` du `main()` dans lequel `quotientEtReste` a été appelé :

```
1   /*Au debut du programme on a 4 valeurs de type int*/
2   int dividende, diviseur, quotient, reste;
3
4   /*on initialise uniquement diviseur et dividende*/
5   scanf("%d %d", &diviseur, &dividende);
6
7   /*On appelle la fonction quotientEtReste*/
8   quotientEtReste(...);
9
10  /*Apres l'appel, quotient et reste sont initialises*/
11  printf("%d = %d*%d + %d\n", dividende, quotient, diviseur, reste);
```

   Définir la fonction `quotientEtReste`. Tester cette fonction en remplissant correctement la ligne `8` dans le `main()` précédent.

2. Écrire une fonction `minEtMax` permettant de stocker dans des variables du programme appelant la valeur minimale et la valeur maximale d'un tableau.

Exercice 5. (Codes de retour)

Dans la question 1 de l'exercice précédent, on n'a pas pris en compte le fait que le diviseur puisse être égal à 0, c'est-à-dire qu'on ne sait pas nécessairement si la division enclidienne a pu être effectuée ou pas. Afin de pouvoir quand même utiliser cette fonction dans un programme, on va lui demander de transmettre une information supplémentaire : est-ce que la division a été possible ou pas ? On utilise pour cela un *code de retour*.

1. Modifier la fonction `quotientEtReste` de façon à ce qu'elle retourne un entier : la fonction retourne 0 (code d'erreur) si la division n'est pas possible et 1 (code de succès) sinon.
2. Dans votre `main()`, appeler la fonction `quotientEtReste` pour 3 couples `(dividende, diviseur)` différents et afficher le résultat de la division euclidienne uniquement s'il a été possible. Si les couples sont respectivement `(10, 3)`, `(2, 0)` et `(0, 10)`, le programme doit afficher quelque chose ressemblant à :

```
10 = 3*3 + 1 /*affichage pour le couple (10, 3)*/
division par 0 impossible! /*affichage pour le couple (2, 0)*/
0 = 0*10 + 0 /*affochage pour le couple (0, 10)*/
```