TP7-Programmation C-IGI 2101

ATTENTION toujours INDENTER VOS PROGRAMMES et COMPILER AVEC -Wall

Exercice 1 : fonction et pointeur en argument

Lors du TP précédent vous avez écrit une fonction « Puissance » qui retourne la puissance d'un nombre passé en 1er argument, et l'exposant en 2nd argument. Les valeurs des arguments restaient inchangées après exécution de la fonction, pour mémoire :

```
float a=10.0, res ;
int b=3 ;
res=Puissance(a,b) ;
printf(''%f puissance %d = %f'', a, b, res) ;
```

Cette fois vous allez écrire une fonction « puissanceV2 » qui **ne vas rien retourner** (donc retourne « void » dans sa signature) mais en revanche **va modifier directement son premier argument**. Pour cela Il faut passer en argument l'adresse de la variable « a » (donc « &a » et non pas « a »). Le programme principal devient ainsi :

```
float a=10.0;
int b=3 ;
printf(''%f puissance %d = '', a, b)
PuissanceV2(&a,b) ;
printf(''%f '', a) ;
```

La signature de la fonction « Puissance2 » est donc « void Puissance2(float *, int) ; ».

Il vous est demandé d'écrire la fonction « Puissance2 » et de la tester.

Exercice 2 : Lecture Fichiers

1) Avec un éditeur simple (notepad++, gedit etc), créez un fichier texte contenant au moins une dizaine de mots de votre choix (sans utiliser d'accents).
2) Ecrire un programme qui ouvre ce fichier texte et affiche les mots qu'il contient un à un, pour cela il faut utiliser 3 fonctions en vous aidant du poly. de cours (p. 199) :
   - La fonction **fopent**
   - La fonction **fscanf** (autant de fois que nécessaire puisqu'à chaque appel elle retourne le mot suivant dans le fichier)
   - La fonction **fclose**
3) Modifier légèrement votre programme pour qu'il puisse compter et afficher le nombre total de mots du fichier texte.

L'algorithme du programme en pseudo-langage est donc :

- Ouvrir en lecture le fichier donné en argument
- Si l'ouverture du fichier n'a pas réussi
    - Quitter le programme en affichant un message d'erreur
- Lire un mot du fichier tout en vérifiant que l'on a pas atteint la fin du fichier (while (fscanf(f,mot) != EOF)
    - Affiche le mot lu
- Fermer le fichier

Note 1 : la fonction fopen retourne un pointeur si le fichier à bien été ouvert. Si pour une raison ou une autre le fichier n'a pas été trouvé, donc pas ouvert, la valeur du pointeur retournée par fopen vos « NULL ». Dans ce cas il ne faut surtout pas essayer de lire et afficher chaque mot du fichier avec la fonction fscan, il faut quitter le programme (avec « return 0 »).

Note 2 : Attention, la déclaration suivante de mot est fausse bien que cela puisse compiler :

```
char *mot ;

../..

fscanf(''%s'',mot) ;
```

En effet : à chaque appel la fonction fscanf va lire un mot du fichier et le stocker à l'adresse spécifiée (ici dans le tableau « mot »). Cette adresse doit correspondre à une zone **réservée** pour stocker le nom qui a été lu. **Or « char *mot ; » ne réserve pas de mémoire** pour stocker une chaine de caractère, cela déclare juste une variable de type pointeur. ➔ Il faut donc bien déclarer « char mot[50] ; » si on veut par exemple réserver 50 octets consécutifs.

Exercice 3 – Fonction CountWord

En reprenant et transformant le code écrit dans l'exercice précédent, écrire un nouveau programme de façon à créer et utiliser une fonction « CountWord » qui va prendre en argument le nom de fichier à ouvrir sous la forme d'une chaine de caractère passée en argument, et retourner le nombre de mots contenus dans ce fichier. La signature de cette fonction est donc « int CountWord(char [ ]) ; »

Et pour utiliser cette fonction dans le main :

```
int NombreMots ;
char NomFichier[]= ''texte.txt''
nombreMots=CountWord(NomFichier) ;
printf(''Le nombre de mot du fichier %s est %d,NomFichier,NombreMots) ;
```

Remarque : si le fichier n'a pas pu être ouvert (i.e. la fonction fopen retourne NULL) alors la fonction CountWord doit retourner -1.

Exercice 4 - Fonction : int SearchWord (char Fichier[ ], int N, char Mot[ ])

A partir du programme précédent, ajouter une fonction nouvelle fonction SearchWord qui prend en argument :

- une chaine de caractère qui sera le nom d'un fichier à ouvrir
- un entier N
- une chaine de caractère qui sera remplit par cette fonction

Cette fonction doit :

- Ouvrir le Fichier dont le nom est donné en argument,
- Lire le Nième mot de ce fichier (faire N fois fscanf)
- Copier le mot lu dans la chaine de caractères passée en argument.
- Retourner la taille du mot si tout s'est bien passé, ou -1 si N étant trop grand par rapport au nombre de mots du fichier ou si le fichier n'a pas pu être ouvert.

Testez la fonction avec un programme que vous appellerez test-SearchWord

Exercice 5 - Fonction int RandomN(int)

A partir du programme précédent, ajouter une fonction " **int RandomN (int N) ; "** qui retournera un entier choisi aléatoirement entre 1 et l'entier N passé en argument.

Pour cela il faut utiliser 2 fonctions **rand** et **srand** dont les signatures se trouvent dans le fichier stdlib.h

`int rand(void); retourne un nombre pseudo aléatoire entre 0 et RAND_MAX (vous pouvez afficher RAND_MAX avec un simple printf, il dépend de votre système). Si vous exécutez plusieurs fois votre programme, la même suite de nombres est générée (voilà pourquoi on parle de pseudo aléatoire). Pour éviter ça il faut exécuter la fonction srand (s pour seed = graine) avant le premier appel à rand et lui donner une valeur idéalement aléatoire. On peut par exemple lui donner l'heure système grâce à la fonction time (dont la signature est dans time.h), bien que ça ne soit pas vraiment aléatoire non plus…`

`Ainsi après un appel à « srand(time(NULL)); » les appels successifs à rand() retourneront une séquence de nombres pseudo aléatoires différents à chaque fois que la graine change, c'est-à-dire à chaque fois que l'heure est différente.`

`Il suffit ensuite de faire un modulo sur le résultat pour borner les valeurs retournées.`

Dans le fichier test-Rand.c testez cette fonction en l'appelant par exemple 10 fois de suite en affichant à chaque fois la valeur retournée. Attention, si RandomN contient l'appel à srand(time) et

comme time n'aura pas le temps de changer entre chacun des 10 appels à RandomN nous risquons d'obtenir toujours le même nombre**. Il faut donc générer une seule fois la graine** au début du main (=ne pas mettre srand dans la fonction RandomN).

Exercice 6 – Test d'intégration

Vous avez effectué le test unitaire de chacune de vos fonctions. Il s'agit maintenant de faire un test d'intégration : écrire un programme test-ex6 qui utilise les 3 fonctions précédentes (CountWord, RandomN, SearchWord) pour tirer aléatoirement un mot dans un fichier et afficher ce mot.

Exercice 7 : Jeu du pendu

Dans un fichier pendu.c, à partir des fonctions précédentes, implémenter le jeu du pendu. Votre programme doit :

- Tirer un mot au hasard dans un fichier texte,
- Indiquer le nombre de lettres à deviner par autant de points,
- Demander d'entrer un mot et montrer les lettres bien placées. Cette dernière étape est répétée un nombre de fois limité (par exemple 10).

Remarque : un pendu peut être dessiné à chaque itération.

Premier essai :

```
/-----
```

Second essai :

```
/-----
|     |
```

Troisième essai :

```
/-----
|     |
|     Q
```

etc. jusqu'à par exemple :

```
/-----
|     |
|     Q
|    \|/
|     |
|    / \
|_______
```