IGI-2101 - TP 6 - Programmation C

T. Grandpierre

Toujours INDENTER VOS PROGRAMME et COMPILER AVEC -Wall

Exercice 1 : Première fonction C – Très facile

Sous Linux, dans votre racine (appelée aussi HomeDirectory) **créez un sous- répertoire *IGI2101***, puis un sous-**répertoire *TPC6***, puis un sous-répertoire ***Exo1***

Dans un fichier *min.c*, écrire

1) le code de la **fonction min3float** qui prendra **exactement** 3 arguments flottants en entrée. Cette fonction retournera le plus petit flottant des 3.

2) une fonction main qui :

    a. déclare et initialise 3 flottants *x1, x2 ,x3* et *res avec 0.0.*
    b. demande ensuite la saisie des 3 valeurs avec un printf suivi de 3 scanf. (rappel scanf à besoin de recevoir l'adresse de la variable pour pouvoir y stocker la valeur saisie au clavier, Cf. TPs précédents)
    c. appelle la fonction *min3float* en lui passant *x1,x2,x3,* et récupère le résultat de cette fonction en le stockant dans *res*
    d. affiche la valeur de *res*.

3) Si tout fonctionne bien, déplacer le code de la fonction après le main plutôt qu'avant. Pensez à ajouter la signature de la fonction avant sa première utilisation

Exercice 2 : fonction « Puissance » en C – Très facile

1. Dans le sous-répertoire Exo2 de TP6, créer un programme « puissance.c », puis y écrire une fonction « Puissance » :
    - Elle reçoit deux arguments :
        - un argument flottant « x »
        - un argument entier « y »
    - Elle retourne le flottant « x » élevé à la puissance « y »
2. Pour calculer le résultat, le programme effectuera une boucle « y » fois dans laquelle l'entier sera multiplié par lui-même dans une variable temporaire notée « temp ».
3. Ecrire une fonction main qui teste cette fonction puissance, par exemple :

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

Exercice 3 : fonctions et structures

Dans le sous-répertoire Exo3 de TP6, créer un programme « complexe.c », puis y déclarer une structure « Cmpx » adaptée au stockage des nombres complexes. Elle possèdera donc un champ « Re » et un champs « Im » de type flottant.

1) Ecrire une fonction Aff_Cmpx qui reçoit en argument un nombre complexe. Elle a pour rôle d'afficher la partie réelle et la partie imaginaires avec des printf. Sa signature est donc :

   void Aff_Cmpx(struct Cmpx);

2) Ecrire une fonction Add_Cmpx qui reçoit en arguments 2 nombres complexes et retourne un nombre complexe qui sera la somme des deux arguments (il suffit d'additionner les parties réelles entre elles et les parties imaginaires entre elles). La signature de cette fonction est donc :

   struct Cmpx Add_Cmpx (struct Cmpx, struct Cmpx);

3) Pour tester vos fonctions, ajouter un programme « main » qui déclare 3 nombres complexes x, y et z. Initialise les 2 premiers. Le troisième recevra la somme des 2 premiers :

```
struct Cmpx  x={1.0,2.0}, y={10.0,100.0}, z;
Aff_Cmpx(x) ;
Aff_Cmpx(y) ;
z=Add_Cmpx(x,y) ;
Aff_Cmpx(z) ;
```

4) Ajouter une ligne au début du programme de façon à permettre la saisie des 4 champs des nombres complexes avec 4 scanf : rappelez-vous, scanf à besoin de l'adresse à laquelle elle va stocker les variables…

Exercice 4 : fonction et tableaux, crible d'Eratosthène

Pour écrire les fonctions ci-dessous**, reprendre ce qui a été effectué lors du dernier TP**, nous allons transformer les différentes parties en fonctions.

Dans un **sous-répertoire *Exo3*** écrire un programme qui :

1. déclare un tableau de 100 entiers,
2. puis appel une fonction Init_tab qu'il faut créer :
   a. la fonction **Init_tab** reçoit un tableau d'entier ainsi que sa taille (sa signature est donc ***void Init_tab(int Tab[], int taille)*.**
   b. Cette fonction initialise le tableau avec les valeurs des 100 premiers entiers à l'aide d'une boucle : ainsi à tab[0] on devra trouver 0, à tab[1] on devra trouver 1 etc.,
3. affiche ensuite le contenu de chaque case du tableau à l'aide d'une fonction **Affiche_tab** qui reçoit le tableau (elle reçoit donc aussi la taille en argument). Cette fonction ne retourne rien. Testez la avec le programme suivant :

```
int Nombres[100] ;
Init_tab(Nombres, 100) ;
Affiche_tab(Nombres,100) ;
```

4. Ecrire une fonction **ElimineMultiples** capable de mettre à zéro toutes les cases dont les indices sont multiples d'un entier M passé en argument : il suffit de

faire une boucle qui parcourt les indices de M en M et met à zéro cette case (voir exemple en 6)

La signature de cette fonction est donc **void ElimineMultiples(int Tab[], int taille, int M) ;**

<u>Attention</u>, il faut veiller à ne pas dépasser la taille du tableau lors de la mise à zéro d'un élément du tableau : il suffit de faire un test avant l'affectation.

5. Testez votre fonction **ElimineMultiples** avec le programme suivant :

```
int Nombres[100] ;
Init_tab(Nombres, 100) ;
ElimineMultiples (Nombres,100, 2) ;
Affiche_tab(Nombres,100) ;
```

6. Avec cet exemple, le résultat affiché doit être :

```
Nombres [0] = 0
Nombres [1] = 1
Nombres [2] = 2
Nombres [3] = 3
Nombres [4] = 0   ← mise à 0
Nombres [5] = 5
Nombres [6] = 0   ←mise à 0
etc.
```

7. Vous avez maintenant les fonctions nécessaires pour ne conserver que les nombres premiers dans le tableau. Pour cela il suffit d'éliminer successivement tous les multiples des éléments qui n'ont pas été éliminés précédemment, voici l'algorithme à implanter :

   - Soit une variable entière **m** initialisée à 2
   - Tant que m<taille du tableau :
     - Si tab[**m**] ≠ 0 alors éliminer tous les multiples de m (avec la fonction ElimineMultiple)
     - **m**=**m**+1
   - Afficher les éléments non vides du tableau : ils sont premiers

   ⇨ Dans le tableau ne reste que les nombres premiers. Ce n'est pas l'algorithme le plus rapide, mais il a l'avantage de ne reposer sur aucun calcul.

Exercice 5 : intérêt des pointeurs – Exemple « Permutation »

Dans le dernier TP vous avez écrit un code qui permet de permuter le contenu de 2 variables x et y. Transformer ce code pour le mettre sous la forme d'une fonction : « **void permute(int x, int y) ; »**
Vérifiez qu'elle fonctionne bien avec le programme suivant :

```
int a=10, b=20 ;
printf(''a=%d  b=%d", a, b);
permute (a, b);
printf(''a=%d  b=%d", a, b);
```

Est-ce que cela fonctionne ? Comment pourrait-on faire ?