Programmation Impérative en Langage C – Examen (durée 2 heures)

Première Session du 11 Janvier 2012

Notes : Seul le document "**Correction TP1/TP2**" (5 pages) fourni ce jour est autorisé. Ne pas le dégrafer pour éviter que des feuilles volantes ne soient répandues sur les tables. Sont interdits tout autre document, les calculatrices, les téléphones, ainsi que tout autre ustensile de calcul et/ou de communication.

Remarque 1 : Une indication sur le niveau atteint est précisé après chaque exercice. Les exercices pourront cependant être traités indépendamment de leur ordre.

Remarque 2 : Dans la suite, les indications concernant les nombres de lignes sont données en comptant toute ligne de code non-vide (#include, prototypes, accolades...).

Remarque 3 : Les petits oublis de point-virgule, parenthèse, accolades, virgules, doubles quotes, *etc.* ne seront pas pénalisant tant qu'ils restent **ponctuels**.

Exercice 1 : Questions de cours

Consigne : *Quelques mots d'explication, ou un exemple, montrant que vous avez compris le concept suffiront. Ne répondez que brièvement, pas besoin de disserter, afin de conserver du temps pour les exercices à suivre. Si vous ne savez pas répondre dans l'instant, le mieux sera certainement de passer à la suite.*

1) À quoi sert un compilateur ?
2) Comment est-il possible de récupérer les arguments de la ligne de commande ?
3) Dans l'usage de la fonction `printf()`, que signifient les formatages `%s` et `%c` ?
4) Combien de fonctions `main()` faut-il dans un programme ? Pourquoi ?
5) Pour accéder à une case d'un tableau, quel opérateur utilise-t'on ?
6) Dîtes ce que fait l'instruction suivante `i += 3 ;`
7) À quoi sert la fonction `fprintf()` ?
8) À quoi sert la fonction `feof()` ?
9) Lorsqu'on a utilisé la fonction `malloc()` quelque part, que faut-il faire avant la fin du programme ?
10) Qu'est-il possible d'observer lors de l'exécution d'un programme avec un *debugger* ?
11) On utilise le logiciel *Glade* et la bibliothèque *GTK* pour construire quoi ?
12) À quoi servent les logiciels *Visual Studio* et *Eclipse* ?

E **Les connaissances générales du domaine sont validées**

Exercice 2 : Compteur d'occurrences (au clavier)

Écrire **un programme** dans lequel l'utilisateur entre 20 nombres entiers au clavier, et qui affiche ensuite le nombre total de nombres strictement positifs lus, strictement négatifs et nuls.

Remarque : 18 lignes (environ)

Exercice 3 : Moyenne d'un tableau

Écrire **une fonction** qui calcule la moyenne des valeurs du tableau d'entiers, de taille `N`, donné en paramètre. La fonction retournera le résultat du calcul. Le prototype de la fonction sera le suivant :

```
double moyenne (int *a, int N) ;
```

Remarque : 6 lignes (environ)

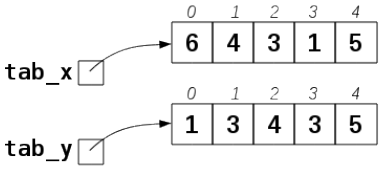
D **Les exercices donnés à l'avance sont résolus**

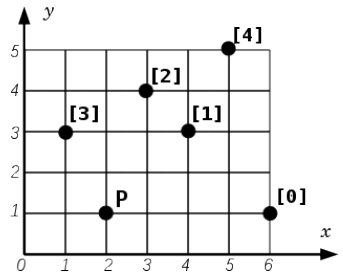
Exercice 4 : Le point le plus proche

Plaçons nous par exemple dans le contexte du développement d'une couche logicielle pour un écran tactile multipoint (smartphone, table interactive, touchsmart, ...). Pour implémenter le double clic, nous avons besoin de déterminer le point le plus proche par rapport à un ensemble de points (*i.e.* l'ensemble des points détectés dans la frame précédente). Remarquons que ce type de calcul pourrait aussi bien aider à déterminer la direction à prendre pour un robot devant passer par ensemble de cibles, ou encore pour un livreur de colis.
Écrire **une fonction** qui détermine quel point est le plus proche d'un point `P` parmi un ensemble de `N` points. Le point `P` sera représenté par ses coordonnées entières `xP` et `yP`. L'ensemble de points sera représenté par deux tableaux d'entiers `x` et `y`, chacun de taille `N`, stockant respectivement les abscisses et les ordonnées des points. La fonction retournera l'indice du point le plus proche. Le prototype de la fonction sera le suivant :

```
int le_plus_proche (int xP, int yP, int *x, int *y, int N) ;
```

Par exemple, soient les deux tableaux `tab_x` et `tab_y` de taille 5 représentant les coordonnées d'un ensemble de points. Soit le point `P` de coordonnées (2, 1).





L'appel de la fonction `le_plus_proche(1,2,tab_x,tab_y,5)` devra alors retourner 3, *c.à.d.* l'indice du point de coordonnées (1,3). Pour réaliser cet exercice, vous utiliserez et écrirez aussi **une fonction** qui calcule la distance euclidienne entre deux points. Son prototype sera le suivant :

```
double distance (int xA, int yA, int xB, int yB) ;
```

Remarque : respectivement 16 et 3 lignes (environ)

C **Les briques de base sont maîtrisées dans un exercice nouveau**

Exercice 5 : Matrices

1) Écrire **une fonction** qui vérifie si la matrice carré d'entiers, de taille `N`, passée en paramètre, est symétrique. Une matrice carré est symétrique lorsqu'elle est égale à sa propre transposée. Une façon simple pour déterminer si une matrice `A` est symétrique est de vérifier que $\forall i, j$ tq $i < j$ on a : $A_{i,j} = A_{j,i}$. Ainsi, on ne se préoccupe pas des valeurs de la diagonale. Par exemple, ces quatre matrices sont des matrices symétriques :

$$\begin{pmatrix} 1 & 4 & 6 & 8 \\ 4 & 1 & 9 & 7 \\ 6 & 9 & 1 & 5 \\ 8 & 7 & 5 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 6 & 4 \\ 2 & 6 & 0 & 5 \\ 3 & 4 & 5 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix} \quad \begin{pmatrix} * & a & b & c \\ a & * & f & d \\ b & f & * & e \\ c & d & e & * \end{pmatrix}$$

La fonction retournera 1 si la matrice est symétrique, et 0 sinon. Le prototype de la fonction sera le suivant :

```
char est_symetrique (int **A, int N) ;
```

2) Écrire **une fonction** qui réalise l'addition de deux matrices A et B dans une troisième C, et toutes trois de dimension $N \times M$. On veut donc une fonction qui réalise l'opération $\forall i, j : C_{i,j} = A_{i,j} + B_{i,j}$. Le prototype de la fonction sera le suivant :

```
void addition (int **C, int **A, int **B, int N, int M) ;
```

Remarque : 5.1 – 11 lignes (environ) / 5.2 – 6 lignes (environ)

B **Manipulation de structures de données complexes**

Exercice 6 : Chiffres, lettres et caractères de ponctuation

1) Écrire **une fonction** qui compte le nombre de chiffres, de lettres et de caractères de ponctuation, dans la chaîne de caractères qui lui est passée en paramètre. Après appel de la fonction, les 3 valeurs comptées seront récupérables dans les variables entières dont l'adresse aura été donnée en argument de la fonction. Le prototype de la fonction sera donc le suivant :

```
void compter (const char *s, int *cpt_chiffre, int *cpt_lettre, int *cpt_ponctuation) ;
```

Aide : voici quelques fonctions de `ctype.h` et `string.h`
- La fonction `int isalpha(char c)` retourne vrai si `c` est un caractère alphabétique, faux sinon.
- La fonction `int isdigit(char c)` retourne vrai si `c` est un chiffre, faux sinon.
- La fonction `int ispunct(char c)` retourne vrai si `c` est un caractère de ponctuation, faux sinon.
- La fonction `int strlen(const char *s)` retourne la longueur de la chaîne passée en paramètre.

2) Écrire **une fonction main()** qui appelle la fonction `compter()` de la question précédente sur la chaîne de caractère suivante : "Vive la programmation !!! Et 1, et 2, et 3... zero !". Ensuite le programme affichera le nombre de chiffres, de lettres et de caractères de ponctuation de cette chaîne.

Remarque : 6.1 – 11 lignes (environ) / 6.2 – 8 lignes (environ)

A	**La notion de fonction est réellement acquise et l'utilisation des pointeurs est maîtrisée**