Université Grenoble Alpes **Année 2025/2026**
L3 MIAGE

Introduction au Système
Planche de TP n°3

Exercice 1. *(rappels C)*

*Le but de cet exercice est de comprendre pré et post-incrémentations, et l'intérêt de la post-incrémentation de **pointeur**.*

1. Ecrire un programme dans lequel vous déclarez un entier k, vous lui affectez la valeur 8, puis vous faites afficher :
- la valeur de l'expression **++k**
- et la valeur de **k** après cette incrémentation.

Puis faire de même avec **k++** (au lieu de ++k).
Qu'observez-vous à l'exécution dans les deux cas ? (assurez-vous que les observations sont différentes !) et **qu'en déduisez-vous** sur la pré et post incrémentation ?
NB. Evidemment une observation similaire peut être faite sur la pré et post décrémentation.

2. Illustrons maintenant l'intérêt des deux opérateurs.
Ecrire un programme dans lequel vous déclarez un entier k et un entier x, vous affectez à k la valeur 8 et à x la valeur **++k**, puis vous faites afficher les valeurs de x et k.
Puis faire de même avec **k++** (au lieu de ++k).
Qu'observez-vous à l'exécution dans les deux cas ? **Qu'en déduisez-vous** sur l'utilisation (par ex. pour une affectation) d'une expression contenant une pré ou post incrémentation ?

3. Un usage fréquent concerne la **post incrémentation de pointeur**…
Faites les déclarations suivantes :

```
float tab[5] = { 0.5 , 1 , 1.8 , 2.4 , 3.1 };
float *p;
```

puis complétez et exécutez le code suivant :

```
p = tab;
while (p < (tab + ????)) {  // remplacer le ???? pour que p puisse
                            // parcourir l'intégralité du tableau tab
   printf("element dans tab : %f\n", *p++);
}
```

Qu'observez-vous à l'exécution ?

Puis faites la même chose mais en remplaçant *p++ par *++p dans le *printf* (et assurez-vous de comprendre ces expressions bien sûr !).
Qu'observez-vous à l'exécution ? Assurez-vous de bien comprendre **comment p a évolué dans les itérations** dans les deux cas.

Exercice 2. *(création de processus)*

Premières expériences de création de processus fils...

1. Ecrire un programme dans lequel le processus initial crée un **processus fils**, ce processus fils affiche son pid et meurt, le processus père affiche son numéro de pid et celui de son fils et meurt.

2. Observer le *man* en ligne de la fonction *fork*, dans la rubrique "return value" : que se passe-t-il **sur erreur** lors de l'exécution de cette fonction ? comment pouvez-vous améliorer votre code de la question 1 en conséquence ?
(pensez désormais à consulter les *mans* et à adapter ainsi si possible...)

3. Modifier le programme de la question 1 afin que le processus initial crée **2 processus frères**.

4. Modifier le programme de la question 1 afin que le processus fils créé **crée à son tour un processus fils** (c'est donc un petit-fils du processus initial). Assurez-vous de bien comprendre

la différence entre cette version et la version de la question 3.

NB. *Ne vous étonnez pas si vous obtenez un comportement étrange à l'exécution (comme un re-affichage du prompt avant la mort de tous les processus). Il sera nécessaire d'adjoindre à ces programmes l'utilisation d'une autre primitive, qui n'a pas encore été étudiée.*

Exercice 3. *(rappels C)*

L'objectif ici est de se (re)familiariser avec les structures et tableaux dynamiques.

1. Définir un type **structure `personne`** permettant de représenter une personne par son nom (chaîne de caractères) et son age (entier).

Dans la suite on souhaite écrire le programme C qui saisit au clavier une quantité quelconque B d'informations sur des personnes, enregistre ces personnes dans un **tableau** (dynamiquement dimensionné pour ne contenir que B éléments), et affiche le nom de la plus âgée de ces B personnes. Le nombre B de personnes à traiter sera **passé en paramètre** à la commande.

2. Ecrire au préalable les **fonctions** suivantes (vous déterminerez vous-même l'entête de ces fonctions) :

- fonction `saisie_personne_suivante` : saisit au clavier **un** nom et **un** entier (attendu positif), et crée et renvoie en résultat **la** structure `personne` correspondante. Elle devra demander à l'utilisateur de recommencer tant que l'entier saisi ne sera pas positif
- fonction `calcule_max` (utilise la précédente) : saisit au clavier successivement les informations pour B personnes, stocke ces personnes dans un tableau reçu en paramètre, puis renvoie en résultat *l'indice dans le tableau* de la personne la plus âgée.

3. Ecrire la fonction `main` qui :
s'assure que la commande a reçu exactement 1 argument (sinon on affichera un message d'erreur et on s'arrêtera),
interprète cet argument comme le nombre B de personnes à saisir (regarder le man de la fonction `atoi` !),
dimensionne en conséquence un tableau,
puis utilise ce qui a précédemment été défini pour afficher le *nom* de la personne la plus âgée.

On aura par exemple à l'exécution :

```
$ ./calcul_de_max 3
Saisir le nom :  Berthe
Saisir son age (entier positif) :  25
Saisir le nom :  Marcel
Saisir son age (entier positif) :  33
Saisir le nom :  Gertrude
Saisir son age (entier positif) :  -17
Attention, saisir un entier positif :  17
La personne la plus agee est : Marcel
```

Exercice 4. *(rappels C + Makefile)*

*Nous reprenons ici **l'exercice 4 du TP2**, que nous allons réorganiser modulairement...*

1. Réorganiser ici cette application en créant (correctement !) les **fichiers** suivants :

- un fichier `exo4_fcts.c` contenant la fonction `remplir` et la fonction `calcule_min`, et son fichier de déclaration `exo4_fcts.h`
- un fichier `exo4.c` contenant la fonction `main` du programme

Penser à placer les bonnes directives aux bons endroits !

2. Entrer les **commandes** permettant de produire les fichiers objets (.o) qui seront nécessaires à la création de l'exécutable, puis la commande permettant de créer cet exécutable.

Quelles commandes devrez-vous ré-exécuter pour recompiler (efficacement) le programme dans le cas où seul le fichier source `exo4.c` serait modifié ?

3. En conséquence, créer un **`Makefile`** pour la compilation de cette application.

Exercice 5. *(rappels de la mise à niveau - A refaire et compléter attentivement)*

1. Saisir dans un fichier le programme suivant, et tenter sa compilation avec l'option `-Werror` (transforme les warnings en erreurs) :

```
#include <stdio.h>

#define EXPONENT 3

int main() {
  float nb, result;
  printf("Give a real number\n");
  scanf("%f", &nb);
  result = pow(nb, EXPONENT);
  printf("%f to the power of %d is equal to %f\n", nb, EXPONENT, result);
  return 0;
}
```

Quel **message d'erreur** obtenez-vous lors de la compilation ? Est-il produit par le préprocesseur, le compilateur ou l'éditeur de liens, et **pourquoi** ?
Corriger le code pour faire disparaître ce message d'erreur spécifique.

2. Quel autre **message d'erreur** obtenez-vous maintenant (même sans l'option `-Werror`) ? Est-il produit par le préprocesseur, le compilateur ou l'éditeur de liens, et **pourquoi** ?
Adapter votre commande de compilation pour ne plus avoir ce message d'erreur (pour rappel, les fonctions mathématiques sont définies dans la bibliothèque `libm.so`).
Il vous reste à exécuter ce programme et vérifier qu'il fonctionne correctement.

3. Voir des compléments d'informations avec les liens suivants, en particulier relativement à l'intérêt de **l'option -D de gcc** (à destination du préprocesseur) :

`http://gcc.gnu.org/onlinedocs/gcc/Preprocessor-Options.html`
`http://www.rapidtables.com/code/linux/gcc/gcc-d.htm`

puis reprendre le code précédent de façon à permettre la **modification de la macro** `EXPONENT` **dynamiquement** lors de la compilation (via option).
Compiler ce programme en fixant la valeur 5 pour `EXPONENT`, et exécuter ce programme.