TP9: Révisions

L'objectif de ce TP est de vous aider à réviser. Les exercices proposés reviennent donc sur les principaux points abordés au cours des TPs précédents. Les exercices sont indépendants et peuvent être faits dans n'importe quel ordre.

Exercice allocation dynamique de mémoire (malloc/free)

Soit la variable `int ***table` servant à stocker un tableau d'entiers à 3 dimensions.

- Initialiser le tableau à 3 dimensions à l'aide de la fonction `malloc`. Toutes les cases du tableau devront être initialisées à 1.
- Vérifier l'allocation mémoire en affichant toutes les cases du tableau avec la fonction `printf`.
- Libérer la mémoire qu'occupe le tableau à l'aide de la fonction `free`.

Exercice signaux

Les processus s'exécutant dans le système d'exploitation peuvent recevoir des signaux. Lorsqu'un signal est reçu, le processus exécute le *traitant d'interruption* associé au signal. Le signal SIGSEGV est envoyé au processus par le système d'exploitation lors d'un accès mémoire non autorisé (par exemple en utilisant un pointeur non initialisé). Le traitant par défaut du signal SIGSEGV affiche "Erreur de segmentation" dans la console puis termine le processus.

- Écrire un programme qui exécute une boucle infinie dans laquelle il tente d'utiliser un pointeur non initialisé. Que se passe-t-il ?
- Redéfinir le traitant d'interruption associé à SIGSEGV pour que sur réception du signal, le processus affiche seulement "Réception de SIGSEGV, l'exécution continue".

Exercice utilisation de `argc/argv` et passage de valeurs par adresse

On souhaite disposer d'une fonction `operations` qui réalise des calculs de base sur des entiers passés en paramètre. La fonction prend donc en paramètre les deux opérandes, et renvoie la somme, le produit et la difference de ces deux opérandes. Il n'est pas possible de renvoyer plusieurs valeurs en C (et on ne souhaite pas utiliser de tableaux). On se propose donc de passer le résultat des opérations *par adresse*. C'est pourquoi la signature de la fonction `operations` est `void operations(int op1, int op2, int *som, int *prod, int *diff)`.

- Implémenter la fonction `operations`.
- Écrire un programme qui prend *en argument* deux entiers (`i.e.`, en utilisant `argc/argv`) et utilise la fonction `operations` pour afficher à la fois la somme, le produit et la difference de ces deux entiers.

Exercice manipulation de fichiers

Écrire un programme qui se comporte de façon similaire à la commande `cat` du terminal linux. Plus précisement, le programme devra :

– Récupérer un nom de fichier donné en argument
– Afficher une erreur si le fichier n'existe pas
– Afficher le contenu du fichier si le fichier existe

On utilisera les fonctions de la librairie C standard de manipulation de fichiers : `fopen`, `fread`, `fgets`, `fclose`, etc...(voir `man`).

Exercice traitement de chaînes de caractères

Écrire un programme qui demande à l'utilisateur de saisir une phrase au clavier [1] et inverse l'ordre des mots de la phrase.
Exemple : *"Ceci est une phrase"* $->$ *"phrase une est Ceci"*.

Exercice pointeurs

On se propose d'implémenter une liste doublement chaînée circulaire d'entiers. Dans ce type de structure, chaque maillon de la chaîne possède un pointeur sur l'élément suivant ainsi que sur l'élément précédent. De plus, le dernier élément de la chaîne pointe sur le premier élément.

– Écrire la fonction permettant d'ajouter un élément à la liste.
– Écrire la fonction permettant d'afficher la liste. La liste étant circulaire, comment faire pour ne pas boucler indéfiniment ?
– Pour tester vos fonctions, insérer 10 entiers et afficher la liste.

1. on utilisera la fonction `gets` plutôt que `scanf`