

## Travaux Pratiques n°3

Remarque : Ce TP se déroulera en quatre parties. La première partie concerne la réalisation de traitements sur des tableaux de nombres réels non triés. La deuxième partie concerne la réalisation des traitements sur des tableaux triés. La troisième partie concerne l'évaluation des performances des traitements réalisés en partie I et II sur des tableaux de très grande taille. La partie IV est un compte-rendu à faire en dehors de la séance.

### Objectifs

- Manipulation des tableaux
- Ajouts de données dans un tableau trié ou pas
- Connaissance des méthodes de recherche dans un tableau trié ou non
- Suppression d'éléments dans un tableau.

### Compétences attendues

- Comprendre la différence en termes de performance entre l'utilisation de tableaux triés ou non triés ;
- Savoir utiliser des fonctions en C en sachant définir les paramètres formels ;
- Savoir ne pas faire d'entrées-sorties dans les traitements ;
- Savoir tester tous les cas (cas limites et cas généraux) d'un traitement ;
- Définir des cartouches pour spécifier les traitements.

### Etapes de développement d'une application en programmation procédurale

L'approche classique de développement repose sur un cycle en V. Elle passe par 3 phases : analyse, codage et tests.

#### Phase 1 : L'analyse

L'analyse consiste dans un premier temps à comprendre le besoin afin d'envisager une solution pouvant y répondre. La compréhension nécessite souvent de faire des schémas qui permettent de mieux comprendre le problème posé. La solution se traduit ensuite dans le choix de structures de données et la définition d'un algorithme par exemple sous forme de pseudocode.

**Dès que l'analyse a été faite, il faut immédiatement définir les tests à effectués pour vérifier le code qui sera obtenu.** C'est la compréhension de la solution qui permet d'envisager les différents cas à tester : le cas général, mais également des cas limites. Dans tous les cas les tests devront répondre à trois objectifs : valider le besoin (ou le cahier de charges), vérifier la robustesse de programme (pas de plantage intempestif), vérifier l'ergonomie du programme.

## Phase 2 : Codage

On peut ensuite passer à la phase de codage. C'est dans cette phase que l'on choisit la plateforme de développement et le langage de programmation.

- La plateforme de développement est basée sur une distribution **linux**. **Ce choix est justifié par des raisons pédagogiques et professionnelles** : vous habituer à utiliser un système libre qui favorise l'insertion de l'outil informatique dans les PME. Il est également à la base des applications embarquées que l'on retrouve dans de nombreux systèmes multimédias grand-public mais également de plus en plus dans des applications industrielles.
- Le langage de programmation et le **langage C** pour son utilisation dans tous les secteurs de l'entreprise, notamment pour le développement des applications industrielles.

La phase de codage est d'autant plus rapide que l'analyse a bien été réalisée. Le code doit être écrit de manière à faciliter la maintenance de l'application. La relecture par une personne tierce doit être facilitée. Cet objectif est atteint par :

- un nommage judicieux des variables correspondant à leur rôle,
- l'indentation du programme,
- l'insertion de commentaires dans le code.

La phase de codage comprend 3 étapes :

**Etape 1** : Vous éditez votre fichier source à l'aide d'un éditeur pleine page de type « kate » ou « gedit ».

**#kate <source>.c &**

**Etape 2** : Compilation et édition de liens du fichier source en code exécutable avec gcc

**#gcc <source>.c -o <executable.exe> -lm**

La compilation sert à transformer le code source en code objet → permet de vérifier la syntaxe

L'édition de lien sert à inclure le code des fonctions des bibliothèques et à transformer le code objet en code exécutable

Avec gcc, l'option « -o » permet de faire la compilation suivi de l'édition de lien

**Etape 3** : Lancer l'exécution de votre programme

**./<executable.exe>**

## Phase 3 : Les tests

Une fois le code réalisé, il faut effectuer les tests définis dans la phase d'analyse. Ces tests doivent donner des jeux d'essais qui doivent être commentés pour indiquer en quoi il vérifie la conformité du code ou besoins ou ils prouvent sa robustesse.

### Comment faire des jeux d'essais

Tous les cas doivent être testés. Les tests doivent être commentés si possibles en renvoyant à des parties de code.

Capture du test :

```
#script <fichier_resultats>
```

Script démarré

=> que tout ce qui sort à l'écran est redigéré vers ce fichier. A la fin arrêter le script

```
#exit
```

## **Partie I – Traitements d'un tableau de données non triés**

Etant donné un tableau de nombre réels, nous désirons réaliser les traitements suivants sous forme de sous-programmes (fonction C).

1. Saisir un nombre réel.
2. Ajouter un nombre réel le plus rapidement dans un tableau. On suppose que l'on connaît la dernière position occupée du tableau.
3. Rechercher si un nombre réel est dans le tableau. Renvoyer l'indice de sa position dans le tableau ou 0 s'il n'y est pas.
4. Supprimer une composante d'un tableau à partir de la donnée de la position de cette composante dans le tableau.
5. Afficher une composante du tableau en fonction de sa position
6. Modifier la valeur d'une composante du tableau
7. Lister tous les éléments du tableau.

**Utiliser le squelette `tp3_squelette_partie1.c` pour implémenter ce programme. Vous renommerez ce fichier en `tp3_partie1.c`. Le fichier `generation_tp3.h` doit se trouver dans le même répertoire que votre application. Il vous permettra de générer de manière aléatoire des tableaux aléatoires de grande taille pour le test de performances.**

## **Partie II - Traitements d'un tableau de données triés**

Etant donné un tableau de nombre réels, nous désirons réaliser les traitements suivants sous forme de sous-programmes (fonction C).

1. Saisir un nombre réel.
2. Ajouter (insérer) un nombre réel dans un tableau trié de manière à ce que qu'il reste toujours trié. On suppose que l'on connaît la dernière position occupée de ce tableau.
3. Rechercher par dichotomie si un nombre réel est dans le tableau. Renvoyer l'indice de sa position dans le tableau ou 0 s'il n'y est pas.
4. Supprimer une composante d'un tableau à partir de la donnée de la position de cette composante dans le tableau.
5. Afficher une composante du tableau en fonction de sa position
6. Modifier la valeur d'une composante du tableau
7. Lister tous les éléments du tableau.

**Inspirez-vous du squelette de la partie 1 pour créer le fichier `tp3_partie2.c`**

## **Partie III - Evaluation des performances**

**Reprendre les fonctions de générations de tableaux de grandes tailles du TP2. Adapter les parties I et II afin de calculer le temps des opérations d'ajouts sur des tableaux de grande taille.**

**Partie IV – Réaliser un compte-rendu du TP en montrant les tests résultats obtenus. Conclure sur l'intérêt des méthodes d'ajouts, de recherche et de suppression effectuées dans les parties I et II.**