

## R1.01 : Initiation au développement (partie 2) Feuille TP n° 2

version 1

### Algorithmes classiques pour tableaux : Recherches dichotomiques

#### OBJECTIFS PEDAGOGIQUES :

- 1.- Codage d'algorithmes sous forme modulaire : création et utilisation de sous-programmes et de modules, séparation du code dans fichiers de spécification et d'implémentation
- 2.- S'exercer à l'écriture progressive de programmes.
- 3.- Réaliser et consigner le test fonctionnel d'un programme.

#### OBJECTIF PRATIQUE :

- Regrouper dans une bibliothèque (module) nommée **bibTableaux** un ensemble de sous-programmes réalisant des opérations sur des tableaux, d'entiers et d'enregistrements de type **Personne**,
- Coder ces sous-programmes à partir des algorithmes conçus en TD,
- Programmer le test individuel de chaque sous-programme dans le fichier **main.cpp**,
- Passer le test fonctionnel de chaque sous-programme et consigner les résultats des tests dans un fichier de tests.

#### RESSOURCES A VOTRE DISPOSITION POUR REALISER CE TP :

**ressourcesTP2.zip** : une archive composée de

- De deux fichiers **.txt** contenant du code à intégrer dans vos modules.
- **main.cpp** à compléter
- **feuilleTests\_tp2.xls** : une feuille de tests qui vous permettra de consigner les résultats des tests fonctionnels réalisés sur les sous-programmes développés

#### EXERCICES A CODER

3 sous-programmes de la feuille de TD n°3 :

- sous-programme **recherchePremiereOccDichoEntier** : recherche dichotomique d'un entier dans un tableau d'entiers trié
- sous-programme **recherchePremiereOccDichoPersonne** : recherche dichotomique d'une personne dont on fournit le nom dans un tableau de personnes trié par le nom
- sous-programme **determinerPremierDernier** : détermine la première et dernière position d'une valeur entière donnée dans un tableau d'entiers trié, ou bien indique que la valeur n'est pas dans le tableau

A terminer **avant** la prochaine séance de TP (semaine prochaine).

Une fois les 3 sous-programmes développés et testés, le fichier excel de test contiendra autant d'onglets que de sous-programmes développés et testés, chaque onglet contiendra le jeu d'essai d'un sous-programme à tester.

#### PREPARATION AU TRAVAIL

1. Dans votre espace de travail de programmation vsCode (dossier **r101\_partie2**), créer un dossier **tp2** pour accueillir les fichiers de cette feuille de TP.
2. Sur eLearn, récupérer l'ensemble des ressources associées à ce TP.
3. Décompresser l'archive **ressourcesTP2.zip**, puis déplacer le fichier **main.cpp** dans le dossier **tp2** qui vient d'être créé.
4. Compiler.

## MISE EN PLACE

### Créer les fichiers du module `bibTableaux`

- Utiliser le contenu du fichier `ressourcesBibTableaux.txt` pour **constituer** et **lier entre eux** les fichiers Interface et Corps du module `bibTableaux`.
- Compiler.

### Lier le fichier `main.cpp` avec le module `bibTableaux`

- Compléter le fichier `main.cpp` fourni avec la directive `#include` appropriée et compiler.
- Supprimer le fichier `ressourcesBibTableaux.txt`

### Conclusion :

Cette première étape est fondamentale. Elle garantit que le ‘cadre’ de travail (développement) est stable. On peut donc s’appuyer dessus pour continuer le développement.

## DOCUMENTER LE MODULE BIBTABLEAUX

### 9. Documenter l’interface

La liste des sous-programmes pouvant intégrer ce module peut être longue. Vous ajouterez pour l’instant dans l’interface les sous-programmes encadrés ci-dessous.

- Primitives d’Entrée-Sortie : sous-programme `afficherTableauEntiers()` fourni
- Observateurs :
- Par exemple, sous-programmes `estTrieCroissant()` et `estTrieDecroissant()` (cf. feuille de TD n°1)
- Recherches :
  - Recherche séquentielle d’une valeur entière dans un tableau d’entiers trié décroissant (cf. feuille de TD n°1)
  - Recherche dichotomique d’une valeur entière dans un tableau d’entiers trié décroissant (cf. feuille de TD n°3) : `recherchePremiereOccDichoEntier`
  - Recherche dichotomique d’une personne par son nom, dans un tableau de personnes trié par ordre décroissant sur le nom (cf. feuille TD n°3) : `recherchePremiereOccDichoPersonne`
  - Recherche première et dernière occurrence d’une valeur entière dans un tableau d’entiers trié décroissant (cf. feuille TD n°3) : `determinerPremierDernier`
- Autres primitives :

Par exemple, enlever les doublons (cf. feuille de TD n°1) d’un tableau trié (cf. feuille n°1).

### 10. Documenter le corps

Selon les directives indiquées sur la feuille de TP n°1.

## **SOUS-PROGRAMME `recherchePremiereOccDichoEntier()`**

### **Ajouter le sous-programme au module `bibTableaux`**

11. Ajouter la **déclaration** du sous-programme **dans l'Interface** du module et le **corps minimal** du sous-programme **dans le corps** du module. Penser aux gardes et aux directives d'inclusion.
12. Compiler. Corriger la/les erreur(s) avant de passer à l'étape suivante.
13. Coder le corps du sous-programme

### **Écrire le sous-programme de test : `testRechercheDichoEntiers()`**

14. Préparer le fichier excel contenant les valeurs à fournir à la recherche dichotomique pour vérifier son bon fonctionnement.
15. Dans le fichier `main.cpp`, créer un sous-programme de test, intitulé par exemple `testRechercheDichoEntiers()`. Les instructions de ce sous-programme contiennent les appels au sous-programme de recherche dichotomique avec les valeurs contenues dans le fichier de test.
16. Compiler. Corriger la/les erreur(s) avant de passer à l'étape suivante.
17. Dans la fonction `main()`, ajouter l'instruction appelant le sous-programme de test `testRechercheDichoEntiers()`.
18. Compiler. Corriger la/les erreur(s) avant de passer à l'étape suivante.

### **Passer le test**

19. Cela correspond à :
  - exécuter le programme `main()`,
  - noter les valeurs obtenues sur le fichier de test excel fourni,
  - et comparer les **valeurs obtenues** (fournies par l'exécution du programme) avec les **valeurs attendues** (celles figurant dans la colonne 'valeurs attendues' du fichier excel de tests).

### **Compter la nombre d'accès au tableau**

20. Reprendre la feuille de TD n°3 – question n°7 :
  - Répondre à 7.a)
  - Modifier en conséquence le code de votre sous-programme de recherche dichotomique `recherchePremiereOccDichoEntier()` ;  
Le nombre d'accès au tableau peut être ajouté comme paramètre résultat supplémentaire du sous-programme et affiché par le programme de test `testRechercheDichoEntiers()`
  - Modifier le sous-programme de test `testRechercheDichoEntiers()` pour qu'il affiche ce résultat supplémentaire
  - Répondre à 7.b)

## **SOUS-PROGRAMME `recherchePremiereOccDichoPersonne()`**

L'ajout de ce sous-programme suppose tout d'abord l'ajout d'un module `Personne` permettant la gestion d'éléments de type `Personne`.

### **Créer les fichiers du module `Personne`**

21. Utiliser le contenu du fichier `ressourcesPersonne.txt` pour **constituer** et **lier entre eux** les fichiers `Interface` et `Corps` du module `Personne`. Penser aux gardes et aux directives d'inclusion.
22. Compiler.
23. Supprimer le fichier `ressourcesPersonne.txt`

### **Ajouter le sous-programme `recherchePremiereOccDichoPersonne` au module `bibTableaux`**

24. Ajouter la **déclaration** du sous-programme `recherchePremiereOccDichoPersonne()` **dans l'Interface** du module `bibTableaux` et le **corps minimal** du sous-programme **dans le corps** du module. Penser aux gardes et aux directives d'inclusion.
25. Faire le lien entre le module `bibTableaux` et le module `ressourcesPersonne`
26. Compiler. Corriger la/les erreur(s) avant de passer à l'étape suivante.
27. Coder le corps du sous-programme `recherchePremiereOccDichoPersonne()`

### **Écrire le sous-programme de test : `testRechercheDichoPersonne()`**

28. Préparer le fichier excel contenant les valeurs à fournir à la recherche dichotomique pour vérifier son bon fonctionnement.
29. Dans le fichier `main.cpp`, créer un sous-programme de test, intitulé par exemple `testRechercheDichoPersonne()`. Les instructions de ce sous-programme contiennent les appels au sous-programme de recherche dichotomique avec les valeurs contenues dans le fichier de test.
30. Compiler. Corriger la/les erreur(s) avant de passer à l'étape suivante.
31. Dans la fonction `main()`, ajouter l'instruction appelant le sous-programme de test `testRechercheDichoPersonne()`.
32. Compiler. Corriger la/les erreur(s) avant de passer à l'étape suivante.

### **Passer le test**

33. Cela correspond à :
  - exécuter le programme `main()`,
  - noter les valeurs obtenues sur le fichier de test excel fourni,
  - et comparer les **valeurs obtenues** (fournies par l'exécution du programme) avec les **valeurs attendues** (celles figurant dans la colonne 'valeurs attendues' du fichier excel de tests).

## SOUS-PROGRAMME `determinerPremierDernier()`

### Ajouter le sous-programme `determinerPremierDernier()` au module `bibTableaux`

34. Ajouter la **déclaration** du sous-programme `determinerPremierDernier()` **dans l'Interface** du module `bibTableaux` et le **corps minimal** du sous-programme **dans le corps** du module. Penser aux gardes et aux directives d'inclusion.
35. Compiler. Corriger la/les erreur(s) avant de passer à l'étape suivante.
36. Coder le corps du sous-programme `determinerPremierDernier()`

### Écrire le sous-programme de test : `testDeterminerPremierDernier()`

37. Préparer le fichier excel contenant les valeurs à fournir à la recherche dichotomique pour vérifier son bon fonctionnement.
38. Dans le fichier `main.cpp`, créer un sous-programme de test, intitulé par exemple `testDeterminerPremierDernier()`. Les instructions de ce sous-programme contiennent les appels au sous-programme de recherche dichotomique avec les valeurs contenues dans le fichier de test.
39. Compiler. Corriger la/les erreur(s) avant de passer à l'étape suivante.
40. Dans la fonction `main()`, ajouter l'instruction appelant le sous-programme de test `testDeterminerPremierDernier()`.
41. Compiler. Corriger la/les erreur(s) avant de passer à l'étape suivante.

### Passer le test

42. Cela correspond à :
  - exécuter le programme `main()`,
  - noter les valeurs obtenues sur le fichier de test excel fourni,
  - et comparer les **valeurs obtenues** (fournies par l'exécution du programme) avec les **valeurs attendues** (celles figurant dans la colonne 'valeurs attendues' du fichier excel de tests).

## SUPPRIMER L'ARCHIVE `ressourcesTP2.zip`

### RAPPELS

Avant de coder :

- Faire un algorithme
- Préparer le jeu d'essai servant à écrire le sous-programme de test
  - Ajouter, dans le fichier de tests excel fourni, un onglet spécifique au sous-programme
  - Créer le tableau avec les valeurs qui seront fournies au sous-programme testé, et les valeurs attendues

Lors du codage : appliquer toutes les recommandations vues dans la première partie de R1.01-Initiation au développement (partie 1).

## Savez-vous bien utiliser le fichier de test ?

Il contient plusieurs **onglets (ou feuilles)**, 1 par sous-programme testé. Cf. Figure 1

données		résultats attendus	
TAILLE	monTab	valCherchée	etatRecherche
10	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	30	VRAI
10	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	-45	VRAI
idem	60, 30, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 2)	30	VRAI
	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	1	FAUX
	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	80	FAUX
	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	15	VRAI
0	(tableau 3)	15	FAUX

Figure 1 : Une feuille de test – vue n°1

Chaque feuille est composée de plusieurs rubriques, pas toujours visibles.

**Vue 1 :** sont visibles

- le jeu d'essai
- et les valeurs attendues

En dépliant la 1ere zone (cf. digne '+'), on obtient la Vue n°2 :

données		résultats attendus	
TAILLE	monTab	valCherchée	etatRecherche
10	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	30	VRAI
10	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	-45	VRAI
idem	60, 30, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 2)	30	VRAI
	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	1	FAUX
	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	80	FAUX
	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	15	VRAI
0	(tableau 3)	15	FAUX

Figure 2 : Une feuille de test - vue n°2

**Vue 2 :** sont visibles

- le jeu d'essai
- des explications sur la pertinence du choix du jeu d'essai
- et les valeurs attendues

En dépliant la 2eme zone (cf. digne '+'), on obtient la Vue n°3 :

données		résultats attendus		résultats OBTENUS		Remarques : problèmes rencontrés - Idées de solution
TAILLE	monTab	valCherchée	etatRecherche	posDansTab	etatRecherche	
10	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	30	VRAI	2		
10	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	-45	VRAI	9		
idem	60, 30, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 2)	30	VRAI	1		
	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	1	FAUX	--		
	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	80	FAUX	--		
	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	15	VRAI	4		
0	(tableau 3)	15	FAUX	--		

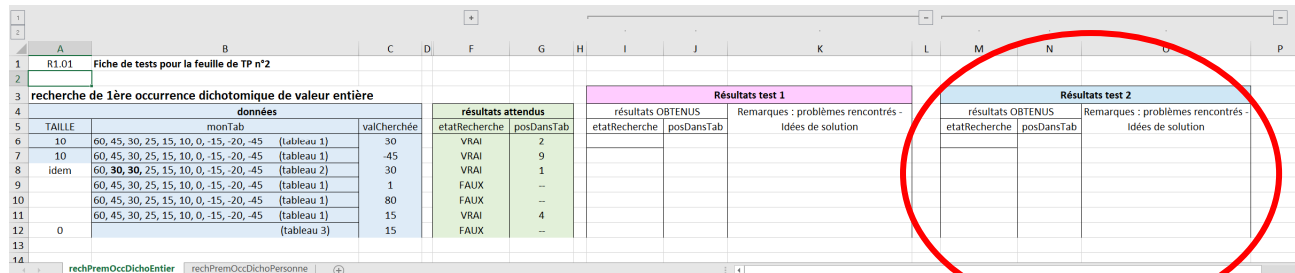
Figure 3 : Une feuille de test - vue n°3

**Vue 3 :** A utiliser pour **passer** le test. Sont visibles

- le jeu d'essai
- [des explications sur la pertinence du choix du jeu d'essai]
- les valeurs attendues
- les résultats du test, à comparer aux valeurs attendues.

Si le test se déroule mal, le programmeur doit corriger les erreurs **et repasser le test**.

Les nouveaux résultats obtenus doivent aussi être inscrits sur la feuille de test, **mais dans une nouvelle zone !**



The screenshot shows a spreadsheet titled 'Fiche de tests pour la feuille de TP n°2'. It contains several tables and sections. A red circle highlights a new test zone on the right side of the sheet, which includes columns for 'résultats test 2' and 'Remarques : problèmes rencontrés - idées de solution'.

données				résultats attendus		résultats test 1		résultats test 2	
TAILLE	monTab	valCherchée	etatRecherche	posDansTab	etatRecherche	posDansTab	Remarques : problèmes rencontrés - idées de solution	etatRecherche	posDansTab
10	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	30	VRAI	2					
10	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	-45	VRAI	9					
idem	60, 30, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 2)	30	VRAI	1					
	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	1	FAUX	—					
	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	80	FAUX	—					
	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 1)	15	VRAI	4					
0	60, 45, 30, 25, 15, 10, 0, -15, -20, -45 (tableau 3)	15	FAUX	—					

**Figure 4 : Feuille de test – vue n)4**

Si le nouveau test n'est pas satisfaisant, on ajoute une nouvelle zone de test.

Ainsi, on garde trace de tous les résultats de tests.

Le système de 'pliage' / 'escamotage' des zones permet de ne travailler qu'avec les informations nécessaires à la tâche en cours de réalisation : conception du test, première passation du test, seconde passation du test, etc....