IUT de BAYONNE PAYS BASQUE - DÉPARTEMENT INFORMATIQUE
Semestre 1- 2018-2019

M1103 : Structures de Données & Algorithmes fondamentaux
Feuille TP n° 1

Algorithmes classiques sur des tableaux

OBJECTIFS PEDAGOGIQUES :

1.- Codage d'algorithmes sous forme modulaire : utilisation de sous-programmes, séparation de la spécification et de l'implémentation d'un sous-programme
2.- S'exercer à l'écriture progressive de programmes.

DOCUMENTS A VOTRE DISPOSITION POUR REALISER CE TP :

- **ressourcesTP1.zip** : une archive composée :
  - du fichier de spécification (.h) et du fichier de définition (.cpp) d'une bibliothèque **bibliothequeTableaux** dont le but est de proposer (aux programmeurs) un certain nombre d'opérations sur les tableaux d'entiers.
  - d'un modèle de feuille de tests (`feuilleTests_tp1.xlsx` et `feuilleTests_tp1.xls`), formats Open Office ou Microsoft au choix.

EXERCICES A CODER

- Exercices 2 et 3 de la feuille de TD n°1, pour lesquels le tableau d'entiers sera *ordonné par ordre décroissant, mais avec de possibles doublons*.
  - Exercice 2 à terminer **avant** la prochaine séance de TP (semaine prochaine).
  - Exercice 3 à terminer lors de la prochaine séance de TP.

Les enseignants peuvent vous demander de leur montrer le travail réalisé.

DIRECTIVES GENERALES

1. Dans votre espace de travail, créer un répertoire **`M1103`** pour accueillir tous les TPs qui seront réalisés dans le cadre de ce module.
2. Avant de continuer, lire le contenu de la feuille de TP pour prendre connaissance du travail à faire.

DIRECTIVES PARTICULIERES A CETTE FEUILLE DE TP

3. Dans l'espace de travail **`M1103`**, créer un répertoire nommé **`tp1`**. Il contiendra tous les exercices qui vous sont demandés dans cette feuille de TP.
4. Dans le répertoire **`tp1`** :
   - Vous devrez créer un projet par programme à écrire (mais attendez la suite du TP pour le faire)
   - Vous regrouperez ensuite tous les projets dans un même workSpace nommé **`tp1`** (mais attendez la suite du TP pour le faire)
5. Dézipper l'archive de ressources fournie et placer les fichiers **`bibliothequeTableaux.h`** et **`bibliothequeTableaux.cpp`** dans le répertoire **`tp1`**.

***Attention, ces fichiers ne doivent pas être déplacés : ils doivent rester à la racine du répertoire* `tp1`.**

Vous devrez compléter cette bibliothèque avec le code des sous-programmes **`rechercheOccPremiere`** et **`determinerPremierDernier`** vus en TD (mais attendez la suite du TP pour le faire).

À titre d'illustration, la Figure 1 montre ce que contient mon Workspace et le ficher **`bibliothequeTableaux.h`** de ***ma*** bibliothèque une fois le TP terminé :

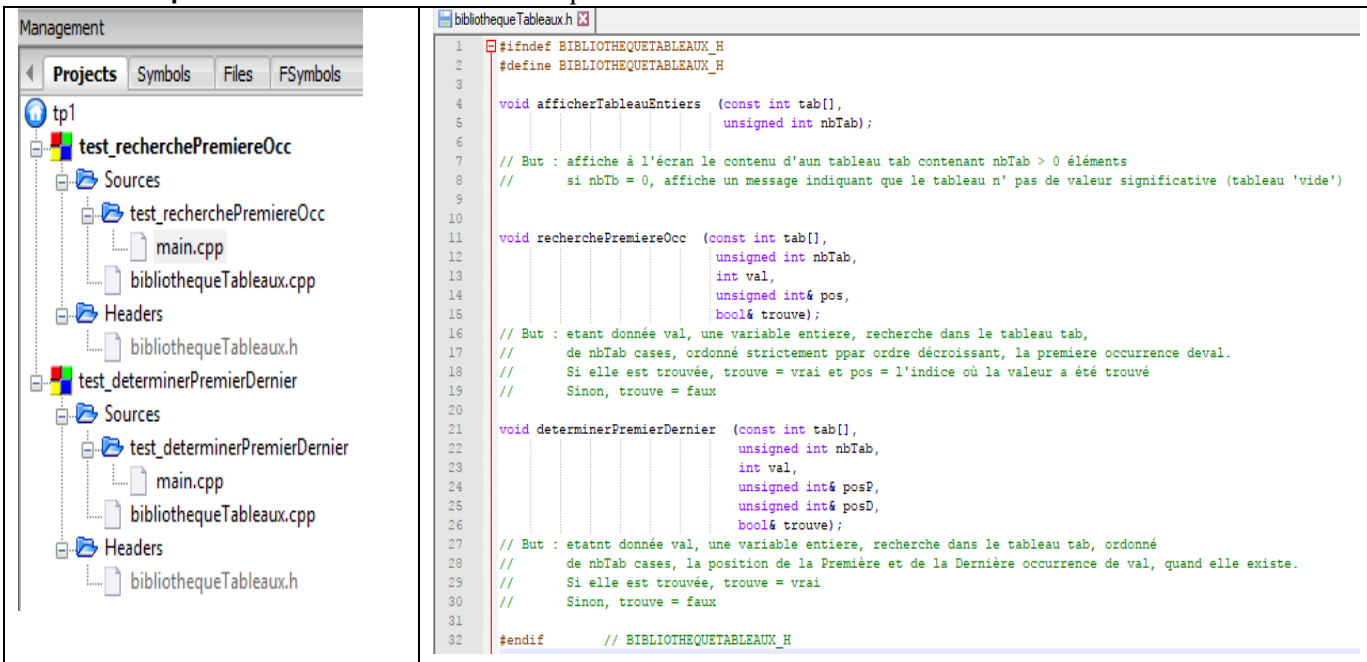
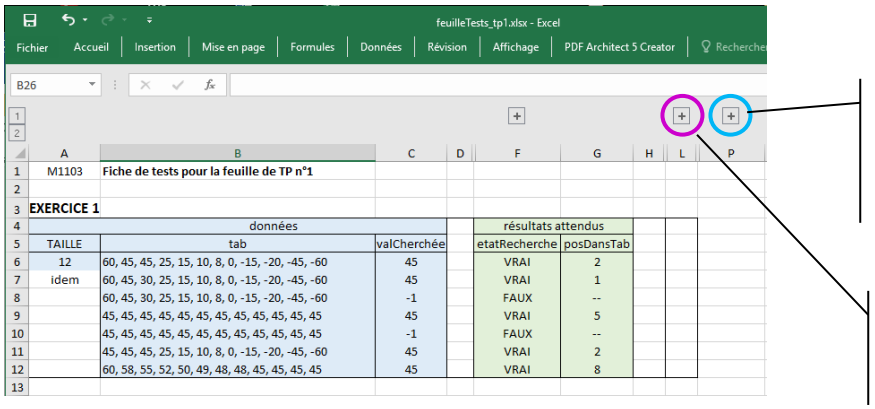


Figure 1 : mon Workspace une fois le tp1 terminé

Exercice 1.- Recherche dichotomique d'une valeur dans tableau ordonné DÉcroissant d'entiers

6. Créer un projet **`test_recherchePremiereOcc`**
7. Ajouter au projet les fichiers **`bibliothequeTableaux.h`** et **`bibliothequeTableaux.cpp`**.
8. Dans le programme **`main`**, ajouter la directive de compilation #**`include`** donnant l'accès au contenu de la bibliothèque **`bibliothequeTableaux`**
9. Compiler pour vérifier que les liens entre les 3 fichiers sont corrects avant le démarrage du codage.
10. Dans le fichier **`bibliothequeTableaux.h`**, écrire en C++ l'entête de **votre** sous-programme **`recherchePremiereOcc`**.
11. Dans le fichier **`bibliothequeTableaux.cpp`**, créer un corps vide de **votre** sous-programme **`recherchePremiereOcc`**. Compiler.
12. Compléter le corps de votre sous-programme **`recherchePremiereOcc`**. Compiler régulièrement.
13. **Avec votre éditeur de texte, créer une feuille de tests.**
    Consigner dans le document l'ensemble des valeurs (du tableau et/ou de la valeur à chercher) qui permettront de tester le bon fonctionnement du programme.
    Pensez à tous les cas possibles :
    - Cas 'classique' : plusieurs occurrences de la valeur cherchée situées 'au milieu' du tableau
    - Cas 'classique : le tableau est strictement ordonné décroissant ou ne contient qu'une seule occurrence de la valeur cherchée
    - Cas 'classique' : pas d'occurrence dans le tableau de la valeur cherchée
    - Cas 'limite' = un peu particulier : le tableau ne contient QUE des valeurs égales à celle cherchée
    - Cas 'limite' : les occurrences de la valeur cherchée sont placées en 'début' de tableau
    - Cas 'limite' : les occurrences de la valeur cherchée sont placées en 'fin' de tableau

    Pour chaque jeu de données prévu, écrire les résultats attendus. Sauvegarder le document dans le répertoire **`tp1`**. Vous pouvez vous inspirer du modèle de feuille de tests fourni (cf. Figure 2) :



	A	B	C	D	F	G	H	L	P
1	M1103	Fiche de tests pour la feuille de TP n°1							
2									
3	EXERCICE 1								
4		données			résultats attendus				
5	TAILLE	tab	valCherchée		etatRecherche	posDansTab			
6	12	60, 45, 45, 25, 15, 10, 8, 0, -15, -20, -45, -60	45		VRAI	2			
7	idem	60, 45, 30, 25, 15, 10, 8, 0, -15, -20, -45, -60	45		VRAI	1			
8		60, 45, 30, 25, 15, 10, 8, 0, -15, -20, -45, -60	-1		FAUX	--			
9		45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45	45		VRAI	5			
10		45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45	-1		FAUX	--			
11		45, 45, 45, 25, 15, 10, 8, 0, -15, -20, -45, -60	45		VRAI	2			
12		60, 58, 55, 52, 50, 49, 48, 48, 45, 45, 45, 45	45		VRAI	8			
13									

Figure 2: feuille de tests exercice 1

14. Dans le fichier **`main`**, écrire un programme ayant pour but de tester le sous-programme **`recherchePremiereOcc`**. Il contiendra les lignes de code réalisant les actions suivantes :
    - Initialiser un tableau d'entiers 'en dur'. Par exemple, dans mon TP, la déclaration du tableau et son initialisation sont les suivantes (plusieurs initialisations de monTab serviront à tester les différents cas de figure préalablement prévus sur la feuille de test) :

```
const unsigned int TAILLE = 12;
      int monTab [TAILLE]= {60, 45, 45, 25, 15, 10, 8, 0, -15, -20, -45, -60};
                    // ordonné DECROISSANT, plusieurs occ. de valeur cherchée au milieu
```

    - Afficher le contenu du tableau (appel de la procédure a**`fficherTableauEntiers`** fournie)
    - Demander à l'utilisateur de saisir une valeur à chercher dans le tableau
    - Effectuer la recherche (= appel du sous-programme **`recherchePremiereOcc`**)
    - Afficher l'issue de la recherche

15. Renommer le workspace **`(tp1)`** et enregistrer le nouveau nom.

    A l'issue de l'opération, le répertoire **`TP1`** aura une organisation semblable celui de la *Figure 3* :

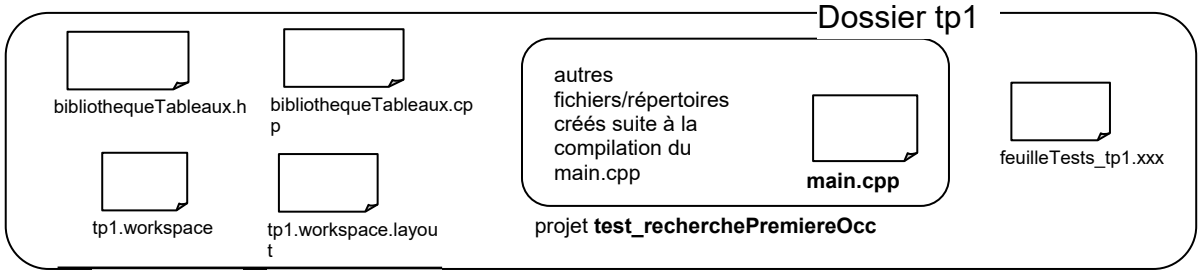


Figure 3 : Organisation provisoire du répertoire tp1

16. **Réaliser les** tests**.**

    Exécuter le programme pour chaque jeu de données prévu dans la feuille de test, en consignant les résultats obtenus (cf. Figure 4), *qu'ils soient conformes ou pas aux résultats attendus*. Sauvegarder.

	A	B	C	D	F	G	H	I	J	K	L
1	M1103	Fiche de tests pour la feuille de TP n°1									
2											
3	EXERCICE 1							Test 1			
4		données			résultats attendus			résultats OBTENUS		Remarques : problèmes rencontrés - Idées de solution	
5	TAILLE	tab	valCherchée		etatRecherche	posDansTab		etatRecherche	posDansTab		
6	12	60, 45, 45, 25, 15, 10, 8, 0, -15, -20, -45, -60	45		VRAI	2		VRAI	2	ok	
7	idem	60, 45, 30, 25, 15, 10, 8, 0, -15, -20, -45, -60	45		VRAI	1		VRAI	1	ok	
8		60, 45, 30, 25, 15, 10, 8, 0, -15, -20, -45, -60	-1		FAUX	--		FAUX	--	ok	
9		45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45	45		VRAI	5		VRAI	5	ok	
10		45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45	-1		FAUX	--		FAUX	--	ok	
11		45, 45, 45, 25, 15, 10, 8, 0, -15, -20, -45, -60	45		VRAI	2		VRAI	2	ok	
12		60, 58, 55, 52, 50, 49, 48, 48, 45, 45, 45, 45	45		VRAI	8		VRAI	8	ok	

Figure 4: Résultats première série de tests exercice 1

17. Montrer la feuille de tests à votre enseignant(e).

18. Si nécessaire, corriger le code (ou l'algorithme et le code) pour que les résultats obtenus soient conformes aux résultats attendus. Recommencer au point 16.

Exercice 2.- Déterminer la première/dernière occurrence d'une valeur dans un tableau ordonné d'entiers

19. **Avec votre éditeur de texte, compléter la feuille de tests.**
    Consigner dans le document l'ensemble des valeurs (du tableau et/ou de la valeur à chercher) qui devront être fournies au programme pour tester complètement son bon fonctionnement.
    Pensez à tous les cas possibles :
    - Cas 'classique' : plusieurs occurrences e la valeur cherchée situées 'au milieu' du tableau
    - Cas 'classique : le tableau est strictement ordonné décroissant ou ne contient qu'une seule occurrence de la valeur cherchée
    - Cas 'classique' : pas d'occurrence dans le tableau de la valeur cherchée
    - Cas 'limite' = un peu particulier : le tableau ne contient QUE des valeurs égales à celle cherchée
    - Cas 'limite' : les occurrences de la valeur cherchée sont placées en 'début' de tableau
    - Cas 'limite' : les occurrences de la valeur cherchée sont placées en 'fin' de tableau

    Pour chaque jeu de données prévu, écrire les résultats attendus. Sauvegarder le document.

20. Répéter les étapes 6. à 12. décrites précédemment :
    - Le projet s'appellera **`test_determinerPremierDernier`**
    - Vous écrirez le code du sous-programme **`determinerPremierDernier`**

21. Dans le fichier **`main`**, écrire un programme ayant pour but de tester le sous-programme **`determinerPremierDernier`**. Il contiendra les lignes de code réalisant les actions suivantes :
    - Initialiser un tableau d'entiers 'en dur'
    - Afficher le contenu du tableau (appel de la procédure **`afficherTableauEntiers`** fournie)
    - Demander à l'utilisateur de saisir une valeur à chercher dans le tableau
    - Effectuer la recherche (= appel du sous-programme **`determinerPremierDernier`**)
    - Afficher l'issue de la recherche

A l'issue de l'opération, le répertoire **`TP1`** aura une organisation semblable à celui de la *Figure 5*:

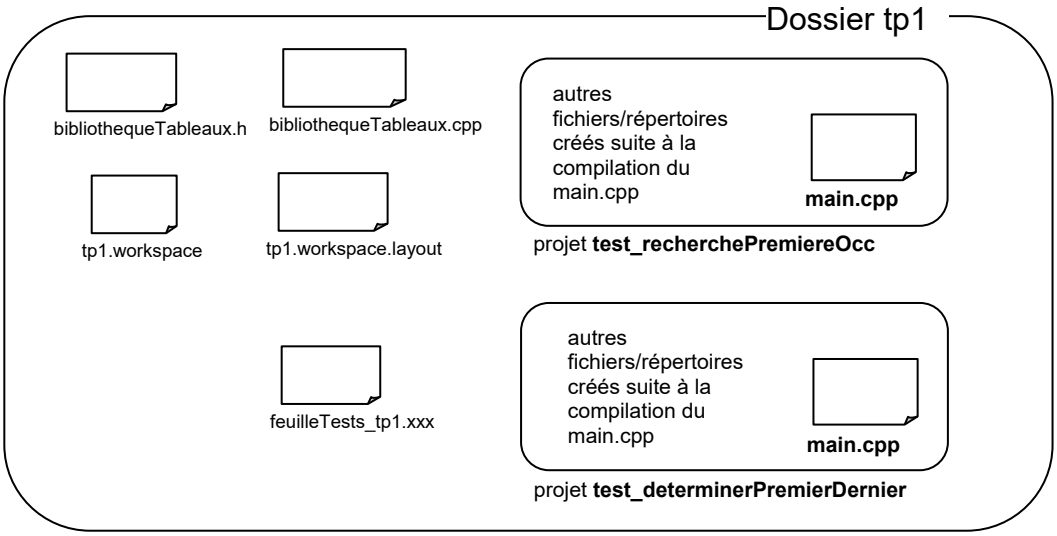


Figure 5 : Organisation finale du répertoire tp1

22. **Réaliser les** tests**.**

23. Exécuter le programme pour chaque jeu de données prévu dans la feuille de test, en consignant les résultats obtenus, *qu'ils soient conformes ou pas aux résultats attendus*. Sauvegarder.

24. Montrer la feuille de tests à votre enseignant(e).

25. Si nécessaire, corriger le code (ou l'algorithme et le code) pour que les résultats obtenus soient conformes aux résultats attendus. Recommencer au point 22.

RAPPELS DES PRINCIPALES BONNES PRATIQUES VUES JUSQU'A MAINTENANT

- Écriture progressive du code en validant chaque étape par une opération de compilation ;
- Respect des règles de nommage des variables et des constantes ;
- Chaque variable déclarée est accompagnée d'un commentaire indiquant son rôle. Ce commentaire est écrit au moment où on déclare la variable et non à la fin une fois que le programme est terminé ;
- Chaque variable est définie par le type qui la représente au mieux : unsigned short int pour un pourcentage plutôt qu'un simple int par exemple ;
- Le code doit toujours être indenté ;
- Les structures de contrôles sont toujours écrites en deux étapes : écriture du squelette de la structure (soit manuellement, si possible en privilégiant les abréviations de Code::Blocks) puis remplissage de la structure ;
- Les paramètres des sous-programmes que vous écrivez devront répondre aux bonnes pratiques vues en cours et résumées dans le document Sous-Programmes : Bonnes pratiques, disponibles sur le webcampus dans la section TP du module M1102.

Pensez à intégrer chacune de ces bonnes pratiques dès que vous codez et surtout, sollicitez votre enseignant(e) pour qu'il vous donne un avis sur les codes que vous avez terminés.

Ne codez jamais un exercice sans avoir, au préalable, réalisé l'algorithme correspondant.

Même en TP, si vous devez écrire un code dont vous n'avez pas l'algorithme, lâchez le clavier, prenez une feuille et un crayon et concevez votre solution sur papier puis faîtes la valider par votre enseignant(e).

Vous demandez à vos enseignants ce que vous ne comprenez pas.

Aucune aide ne vous sera fournie en TP si vous n'êtes pas en mesure de montrer l'algorithme sur lequel vous vous appuyez pour élaborer votre code.