IUT de BAYONNE PAYS BASQUE - DÉPARTEMENT INFORMATIQUE
Semestre 1- 2021-2022

R1.01 : Initiation au développement (partie 2)
Feuille TD n° 6

Utilisation de Files

version 1

Objectifs :

1.- Apprendre les manipulations de base d'un Type abstrait de Données File

1. Afficher le contenu d'une file

On souhaite écrire un sous-programme, nommé **`afficher`**, qui affiche à l'écran le contenu d'une file **`file`** passée en paramètre (par exemple, une file d'entiers).

TRAVAIL A FAIRE

1. Écrire en C++ la *déclaration* de ce sous-programme.
2. Justifier les choix suivants : procédure/fonction et type de passage de paramètre choisi.
3. Définir la stratégie de cet algorithme.
   (a) Modèle(s) d'algorithme(s) vu(s) en cours applicable(s) pour atteindre le but visé. *Justifier.*
   (b) Action(s) à répéter
   (c) La/les conditions de fin de répétition
4. Écrire l'algorithme correspondant, accompagné d'un *dictionnaire succinct* des variables/constantes utilisées : nom, type, signification.

2. Manipulation de files

On souhaite écrire le sous-programme **`defilerJusquA`** suivant :

Étant donnés ses paramètres :

- **`file`**, une file d'entiers, éventuellement vide, dont les valeurs ne sont pas triées, et avec éventuellement des valeurs doublons,
- **`valEnt`**, une valeur entière,

ce sous-programme supprime les éléments de **`file`** jusqu'à trouver une occurrence de **`valEnt`**. La première valeur **`valEnt`** trouvée n'est pas supprimée de **`file`**, elle se retrouve en tête de **`file`**.

Si **`valEnt`** n'est pas trouvée, **`file`** , le résultat du sous-programme, est retournée **vide**.

Exemple : nous utiliserons comme exemple d'illustration les 2 files **`file1`** et **`file2`** suivantes :

`file1`	`file2`
tête 3 4 1 2 1 5 queue sens de circulation `premier(file1) = 3`	file vide

Le tableau ci-dessous illustre différents appels du sous-programme **`defilerJusquA`** sur les éléments **`file1`** et **`file2`** :

Etat de la file **AVANT** action	Action exécutée : `defilerJusquA(file, valEnt)`	Etat de la file **APRES** action	avec
tête 3 4 1 2 1 5 queue	`defilerJusquA(file1, 3)`	tête 3 4 1 2 1 5 queue	`premier(file1) = 3`
tête 3 4 1 2 1 5 queue	`defilerJusquA(file1, 2)`	tête 2 1 5 queue	`premier(file1) = 2`
tête 3 4 1 2 1 5 queue	`defilerJusquA(file1, 1)`	tête 1 2 1 5 queue	`premier(file1) = 1`
tête 3 4 1 2 1 5 queue	`defilerJusquA(file1, 5)`	tête & queue 5	`premier(file1) = 5`
tête 3 4 1 2 1 5 queue	`defilerJusquA(file1, 8)`	File vide	…
File vide	`defilerJusquA(file2, 2)`	File vide	…
File vide	`defilerJusquA(file2, 8)`	File vide	…

TRAVAIL A FAIRE

1. En utilisant la notation vue en cours,

Si le sous-programme `action` a pour paramètre `element` :

- `element` sert à écrire la pré-condition du sous-programme `action`, à savoir les propriétés du paramètre `element` avant l'exécution du sous-programme `action`
- `element'` sert à écrire la post-condition du sous-programme `action`, à savoir les propriétés du paramètre `element` après l'exécution du sous-programme `action`

écrire les pré-conditions et les post-conditions associées à ce sous-programme.

2. Écrire en C++ la déclaration de ce sous-programme.
3. Décrire la **stratégie** utilisée dans l'algorithme (max 8 lignes). Pour cela, préciser :
   a) Le modèle d'algorithme à utiliser
   b) La/les action(s) à répéter
   c) Si cela est pertinent, la/les condition(s) de fin d'itération
4. a.- Écrire l'algorithme de ce sous-programme.
   Ne pas oublier de mentionner les Données/Résultats des principales actions.

   b.- Accompagner l'algorithme d'un dictionnaire succinct (nom, type, signification) des éléments **autres** que ceux déjà donnés dans le sujet.
5. Analyse de l'algorithme produit. Appelons :
   - `nbAppelsPremier` : le nombre d'*appels* au sous-programme `premier()` contenus dans votre algorithme. C'est un entier ≥ 0.
   - `nbAppelsDefiler` : le nombre d'*appels* au sous-programme `defiler()` contenus dans votre algorithme . C'est un entier ≥ 0.
   - `nbAppelsEnfiler` : le nombre d'*appels* au sous-programme `enfiler()` contenus dans votre algorithme. C'est un entier ≥ 0.
   - `nbAppelsTaille` : le nombre d'*appels* au sous-programme `taille()` contenus dans votre algorithme. C'est un entier ≥ 0.

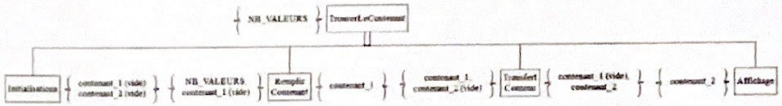
   Pour une `file` contenant `nbElements`, remplacer chaque élément par sa valeur dans les phrases ci-dessous :

   Pour une file contenant `nbElements` éléments, mon sous-programme `defilerJusquA(file)` appelle :
   a) le sous-programme `premier(file)` au **maximum** *nbAppelsPremier* fois
   b) le sous-programme `defiler(file)` au **maximum** *nbAppelsDefiler* fois
   c) le sous-programme `defiler(file)` au **maximum** *nbAppelsEnfiler* fois
   d) le sous-programme `taille(file)` au **maximum** *nbAppelsTaille* fois
6. Remarque : dans quelle situation les maximums calculés précédemment seront-ils atteints ?

Exercice récapitulatif

3. Structure de données adéquate

L'algorithme ci-dessous utilise 2 variables : ce sont des structures de données linéaires homogènes, nommées respectivement **contenant_1** et **contenant_2**. On supposera qu'elles contiennent des **entiers.**



avec :

`Initialisations` :	effectue les éventuelles nécessaires initialisations des variables `contenant_1` et `contenant_2`.
`RemplirContenant` :	stocke `NB_VALEURS` valeurs entières dans `contenant_1`. On supposera pour cet exercice que `NB_VALEURS` = 4, et que les valeurs stockées sont, dans l'ordre : **1** puis **2** puis **3** puis **4**.
`TransfertContenu` :	Les valeurs de `contenant_1` sont transférées dans `contenant_2`.
`Affichage` :	Le contenu de la variable `contenant_2` est affiché à l'écran.

L'algorithme a été exécuté 3 fois en choisissant pour `contenant_1`, soit une pile, soit une file. Les figures exécution_a.-, exécution_b.-, et exécution_c. ci-dessous montrent le résultat à l'écran de l'action **Affichage** l'action selon la nature de `contenant_1`.

`exécution_a`

contenant_1 est	l'affichage du contenu de contenant_2 est
une **pile**	`Restitution des valeurs saisies :` `1 . 2 . 3 . 4 .`

`exécution_b`

contenant_1 est	l'affichage du contenu de contenant_2 est
une **file**	`Restitution des valeurs saisies :` `4 . 3 . 2 . 1 .`

`exécution_c`

contenant_1 est	l'affichage du contenu de contenant_2 est
une **pile**	`Restitution des valeurs saisies :` `4 . 3 . 2 . 1 .`

TRAVAIL A FAIRE :

1. Pour chacun des cas (a, b et c), indiquer quelle est la structure de données ***la plus appropriée*** à choisir pour l'élément `contenant_2` de sorte à produire l'affichage indiqué.
2. Pour chacun des cas (a, b et c) :
   - Écrire la déclaration d'un sous-programme `remplirContenant` dont les paramètres sont :
     - o `nbValeurs`, le nombre de valeurs à mettre dans
     - o `destination`, l'élément recevant les `nbValeurs` entiers
   - Écrire l'algorithme de ce sous-programme
3. Pour chacun des cas (a, b et c) :
   - Écrire la déclaration d'un sous-programme `TransfertContenu` dont les paramètres sont :
     - o `origine`, le paramètre duquel on enlève le contenu
     - o `destination`, le paramètre recevant le contenu
   - Écrire l'algorithme de ce sous-programme