R1.01 : Initiation au développement (partie 2)
Feuille TD n° 1

Algorithmes classiques – révisions
Utiliser les assertions dans les algorithmes

Objectifs :
1.- Consolider les apprentissages en algorithmique vus dans la première partie du module.
2.- Consolider la démarche algorithmique par l'utilisation d'assertions (pré-post conditions)

Exercice 1.
Recherche de première occurrence dans un tableau d'entiers strictement ordonné

Etant donnés :
- `tab` un tableau de `lgTab` valeurs entières, *strictement ordonné par ordre* ***décroissant***, c'est à dire tel que `tab(i)` > `tab(i+1)` pour tout `0` ≤ `i` < `lgTab-1`
- et une valeur entière `val`

On souhaite écrire un sous-programme **`recherchePremiereOcc`** qui indique si (oui ou non) `val` ∈ `tab`, et si oui, donne sa position dans `tab`.

Partie 1 –Déclaration et Appel

Travail à faire

1. La position de `val` dans `tab` est-elle unique ? ***Argumenter***.
2. Écrire la *déclaration C++* du sous-programme **`recherchePremiereOcc`** en respectant les noms qui vous ont déjà été fournis (nom du sous-programme et des éléments `tab`, `lgTab` et `val`).
3. Compléter le programme **`main()`** ci-dessous avec *l'appel* du sous-programme **`recherchePremiereOcc`**, les déclarations des éléments que vous jugez utiles, et l'action exploitant les résultats produits par cet appel.

```
1   int main()
2   {
3       // ELEMENTS DU PROGRAMME (constantes et variables)
4       const unsigned short int TAILLE = 10;
5       int monTab [TAILLE] = {60, 45, 30, 25, 15, 10, 0, -15, -20, -45};
6                             // trié strictement décroissant
7       int valCherchee;      // valeur cherchée dans monTab, saisie au clavier
8       …
9
10      // TRAITEMENTS
11
12      // (clavier) >> saisie valCherchee >> valCherchee
13      cout << "Entrer le nombre cherche : ";
14      cin >> valCherchee;
15
16      // monTab, TAILLE, valCherchee >> rech. 1ere occ. >> ??
17      …
18
19
20      // ?  >> exploiter Résultat >> (écran)
21      …
22
23
24      return 0;
25  }
```

Code n° 1 : Appel de `recherchePremiereOcc` à compléter

Partie 2 – Stratégie de l'algorithme basée sur les modèles connus

Travail à faire

4. Parmi les modèles d'algorithmes connus à ce jour, indiquer le modèle le plus adapté pour résoudre ce problème. ***Argumenter et préciser les modalités d'adaptation***.
5. Exprimer les conditions de fin d'itération à l'aide des éléments de l'algorithme.
6. Écrire l'algorithme :
   a.- accompagné des spécifications internes succinctes nécessaires.
   b.- en précisant les pré-post-conditions associées
7. Bonus (hors séance de TD). Compléter cet algorithme de sorte qu'il calcule et affiche le nombre de fois qu'une case du tableau est accédée au cours de l'exécution de l'algorithme.

Partie 3 – Tableau d'enregistrements Bonus (hors séance de TD)

Supposons maintenant que le tableau **`tab`** contient **`lgTab`** enregistrements décrivant des personnes, et que l'on souhaite lancer la recherche d'une personne par son **`nom`**.

Chaque enregistrement est du type **`UnePersonne`** suivant :

```
struct UnePersonne
{
    string nom;
    string prenom;
    UneAdresse adresse;
};
```

avec

```
struct UneAdresse
{
    string numRue;
    string nomRue;
    unsigned short int codePostal;
    string nomVille;
};
```

Travail à faire

8. Quelles sont les pré-conditions (conditions sur les données de l'algorithme) pour pouvoir appliquer la recherche de première occurrence développée à la question précédente ?
9. En supposant toutes les conditions satisfaites, adapter l'algorithme à ce cas particulier de recherche.

Exercice 2
Assertions pour préciser les conditions d'utilisation d'un algorithme

Reprise de l'exercice R1.01 – Partie 1 : TD n°7 Exercice 4

« Proposer un algorithme qui calcule la moyenne de toutes les valeurs stockées dans un tableau tab d'entier composé de lgTab cases. »

Considérons la solution proposée ci-dessous, sous la forme d'une fonction **`moyenneValeurs`** dont les paramètres données sont : le tableau d'entiers **`tab`**, et **`lgTab`**, le nombre de cases du tableau.
La valeur résultante retournée est la moyenne calculée.

Fonction : `moyenneValeurs`
But : Calcule et retourne la moyenne réelle des **`lgTab`** valeurs entières du tableau **`tab`**.

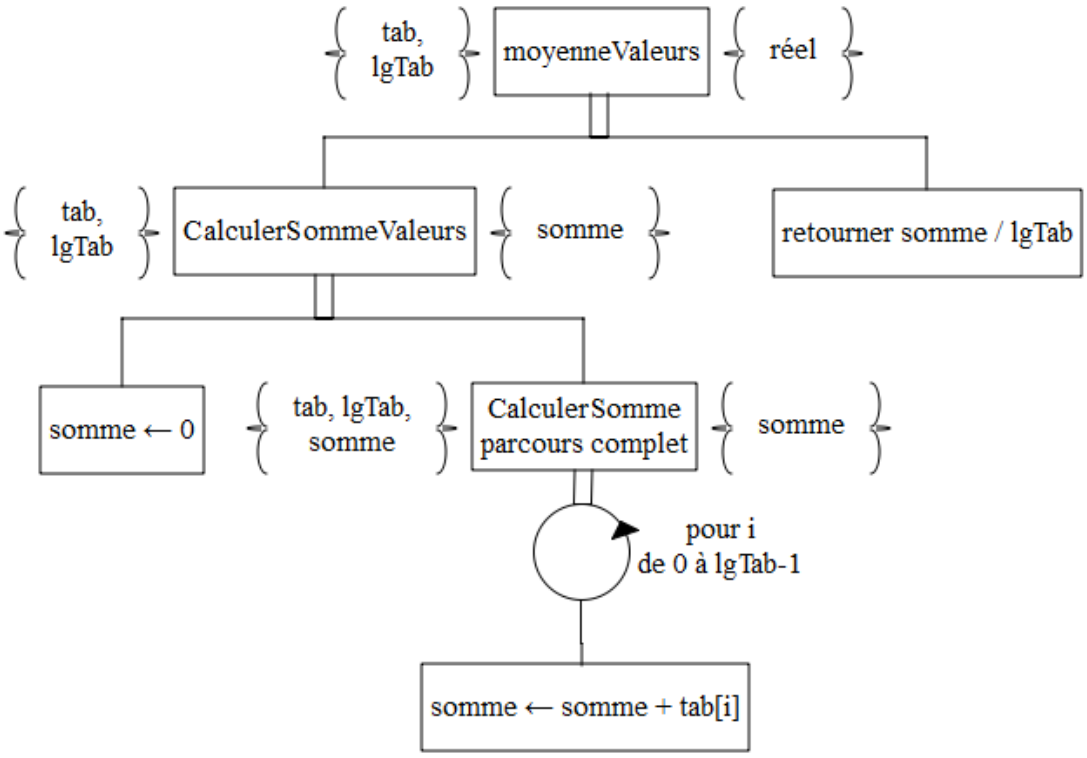


Figure 1 : Algorithme fonction moyenneValeurs calculant la moyenne des valeurs d'un tableau d'entiers

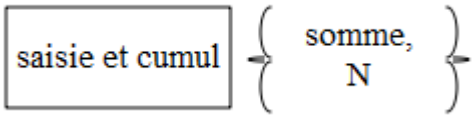
Dictionnaire des éléments :
`tab` : tableau d'entiers comprenant **`lgTab`** cases
`lgTab` : entier, nbre de cases du tableau
`somme` : accumulateur entier, somme des **`lgTab`** valeurs du tableau **`tab`**

Travail à faire

1. Discussion : quelle est l'hypothèse implicite du concepteur de cet algorithme et quelle(s) en sont les conséquences ?
2. Quelle(s) solution(s) imaginez-vous pour corriger le manque de robustesse de cet algorithme ?

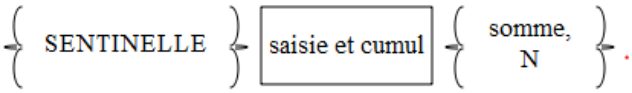
Exercice 3.
Assertions pour comprendre l'état des variables après exécution d'un algorithme

On souhaite écrire un sous-programme réalisant la somme de nombres saisis au clavier. Le nombre de valeurs à saisir n'est pas connu à l'avance.

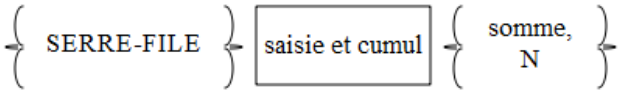


Il existe deux principales stratégies pour la mise en œuvre de cet algorithme :

- **Stratégie avec sentinelle** : L'algorithme utilise une valeur butoir prédéfinie servant à indiquer la fin de la saisie.
  La sentinelle ne fait cependant pas l'objet du traitement (sa valeur n'est pas cumulée à la somme).



- **Stratégie avec serre-file** : L'algorithme utilise une valeur butoir prédéfinie servant à indiquer a fin de la saisie.
  Le serre-file fait l'objet du traitement (sa valeur est cumulée à la somme).

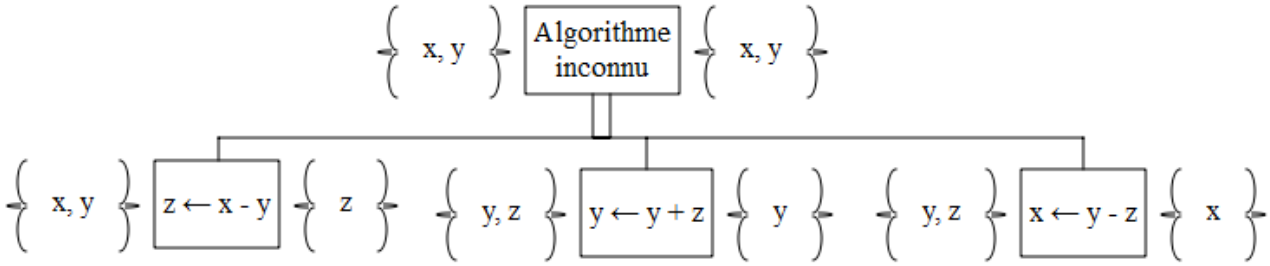


Travail à faire

1. Donner une illustration des concepts de sentinelle / serre-file dans la vie courante
2. Pour chaque algorithme, préciser les propriétés des résultats au moyen de post-conditions.
3. Écrire les algorithmes correspondant à chaque stratégie.

Exercice 4.
Assertions pour comprendre un algorithme

Soit l'algorithme suivant, où x et y sont deux variables entières.



Cet algorithme est composé de 3 actions, que nous nommerons, dans l'ordre d'exécution, action1, action2, action3.

Travail à faire

1. Exécuter cet algorithme à la main en traçant le contenu des variables, pour les valeurs de :

   x = 5 et y = 3

Après l'action, les valeurs des éléments sont :	x = 5	y = 3	z
Après action n°1			2
Après action n°2			
Après action n°3			

2. Que peut-on en déduire sur le but de cet algorithme ?
3. Utiliser les assertions pour démontrer que le but de l'algorithme est bien celui trouvé par expérimentation :
   Si (x_i) est la suite des valeurs prises par la variable x au cours de cet algorithme, et sachant que $x_0 = 5$
   Si (y_i) est la suite des valeurs prises par la variable y au cours de cet algorithme, et sachant que $y_0 = 3$
   Si (z_i) est la suite des valeurs prises par la variable z au cours de cet algorithme, et sachant que z_0 est calculée par l'action n°1

   Ecrire les post-conditions de chaque action de cet algorithme, en se focalisant sur la relation existant entre les valeurs des variables.

   Exemple :
   Post-condition de l'action n°1 : la variable z est telle que sa valeur, $z_0 = x_0 - y_0$

Exercice 5.
Tableau trié par ordre croissant ?

Soit **`tab`** un tableau d'entier de **`lgTab`** cases.
On souhaite écrire un sous-programme **`estTrieCroissant`** indiquant si un tel tableau est, ou pas, trié par ordre croissant de valeurs.
La stratégie proposée décompose le problème initial en plusieurs sous-problèmes.

Définition préalable
On dit qu'il y a **inversion** par rapport à la position **pos** de ce tableau s'il existe un indice **`indiceInversion`**, tel que

- **`0 <= indiceInversion <= lgTab-1`**
- **`indiceInversion > pos`** et **`tab[pos] > tab [indiceInversion]`**

Travail à faire

1. Dessiner deux exemples de tableau de 5 cases, l'un vérifiant cette propriété, l'autre ne la vérifiant pas. Préciser, à chaque fois, les valeurs de **pos** et de **`indiceInversion`**

2. Sous-programme **`inversionPos`** qui calcule et retourne le nombre total d'inversions trouvées dans le tableau **`tab`** par rapport à la position **`pos`**
    a) Écrire sa déclaration, en justifiant le choix de la nature de sous-programme (procédure / fonction)
    b) Si cela est le cas, préciser sur quel modèle d'algorithme connu s'appuie sa stratégie et les modalités d'adaptation
    c) Écrire l'algorithme (+ pré-post conditions pertinentes), accompagné de son dictionnaire des éléments

3. Sous-programme **`inversionTab`** qui calcule et retourne le nombre total d'inversions trouvées dans le tableau **`tab`**
    a) Écrire sa déclaration, en justifiant le choix de la nature de sous-programme (procédure / fonction)
    b) Si cela est le cas, préciser sur quel modèle d'algorithme connu s'appuie sa stratégie et les modalités d'adaptation
    c) Écrire l'algorithme (+ pré-post conditions pertinentes), accompagné de son dictionnaire des éléments

4. Sous-programme **`estTrieCroissant`** qui indique si le tableau **`tab`** est trié ou pas par ordre croissant de valeurs
    a) Écrire sa déclaration, en justifiant le choix de la nature de sous-programme (procédure / fonction)
    b) Si cela est le cas, préciser sur quel modèle d'algorithme connu s'appuie sa stratégie et les modalités d'adaptation
    c) Écrire l'algorithme (+ pré-post conditions pertinentes), accompagné de son dictionnaire des élément

Exercice 6.
Elaguer doublons

Étant donné **`tabP`** un tableau d'entiers, de taille **`lgTab`**, *ordonné* (avec possibles doublons), on souhaite écrire le sous-programme **`elaguerDoublons`** produisant un tableau **`tabR`** trié comme **`tabP`** mais sans doublons.

Travail à faire

1. Écrire la déclaration C++ de ce sous-programme.
2. Indiquer le(s) modèle(s) d'algorithme qui sera/seront utilisé(s) pour écrire cet algorithme et les modalités d'adaptation.

   Préciser si l'ordre (croissant/décroissant) dans lequel se trouvent les éléments est important ou pas.
3. Écrire l'algorithme accompagné des spécifications internes succinctes nécessaire, sans oublier les pré-post-conditions associées.