
Le test

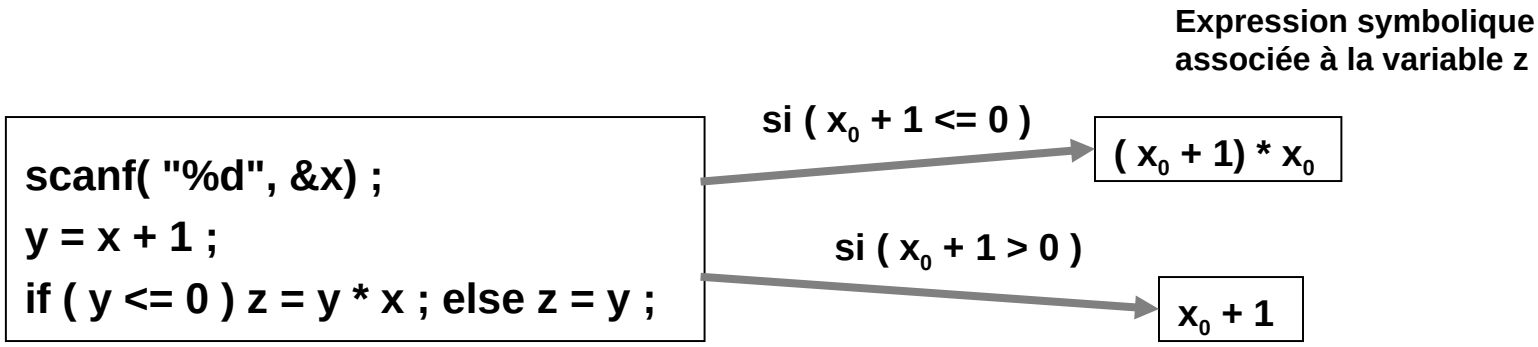
Test de logiciel

- Vérification (interne : conformité aux spécifications, ...)
  ≠ validation (externe : adéquation aux besoins du client)
- L'objectif du test est de détecter des fautes ou des inadéquations d'un logiciel par rapport à des références établies (spécifications, normes ou règles portant sur le code ou les documents le concernant).
- Les tests ne pouvant être exhaustifs, ils ne pourront jamais prouver que le programme est juste : un test ne peut que démontrer la présence d'erreurs, pas leur absence.
- Il existe deux grandes classes de méthodes de test :
  - le test statique traite le texte du logiciel sans exécuter ce logiciel sur des données réelles, il est applicable à des textes de spécification ou de programmes
  - le test dynamique repose sur l'exécution effective du logiciel sur un sous-ensemble bien choisi du domaine de ses entrées possibles

Test statique

Les principales méthodes de test statique sont :

- les lectures croisées (cycles auteur/lecteur) ou l'inspection (relecture en groupe) réalisent la vérification "collégiale" d'un document.
- l'analyse d'anomalies (telles que typage impropre, incohérence des interfaces de modules, portions de code isolées, mauvais usage de variables, etc.) assistée par des outils
- l'évaluation symbolique qui simule l'exécution du programme sur des données symboliques, exemple :



Test dynamique

- La qualité d'une démarche de test dynamique dépend de la représentativité des jeux d'essai choisis.
- Il existe deux types de test dynamique :
  - le test fonctionnel ou test de la boîte noire

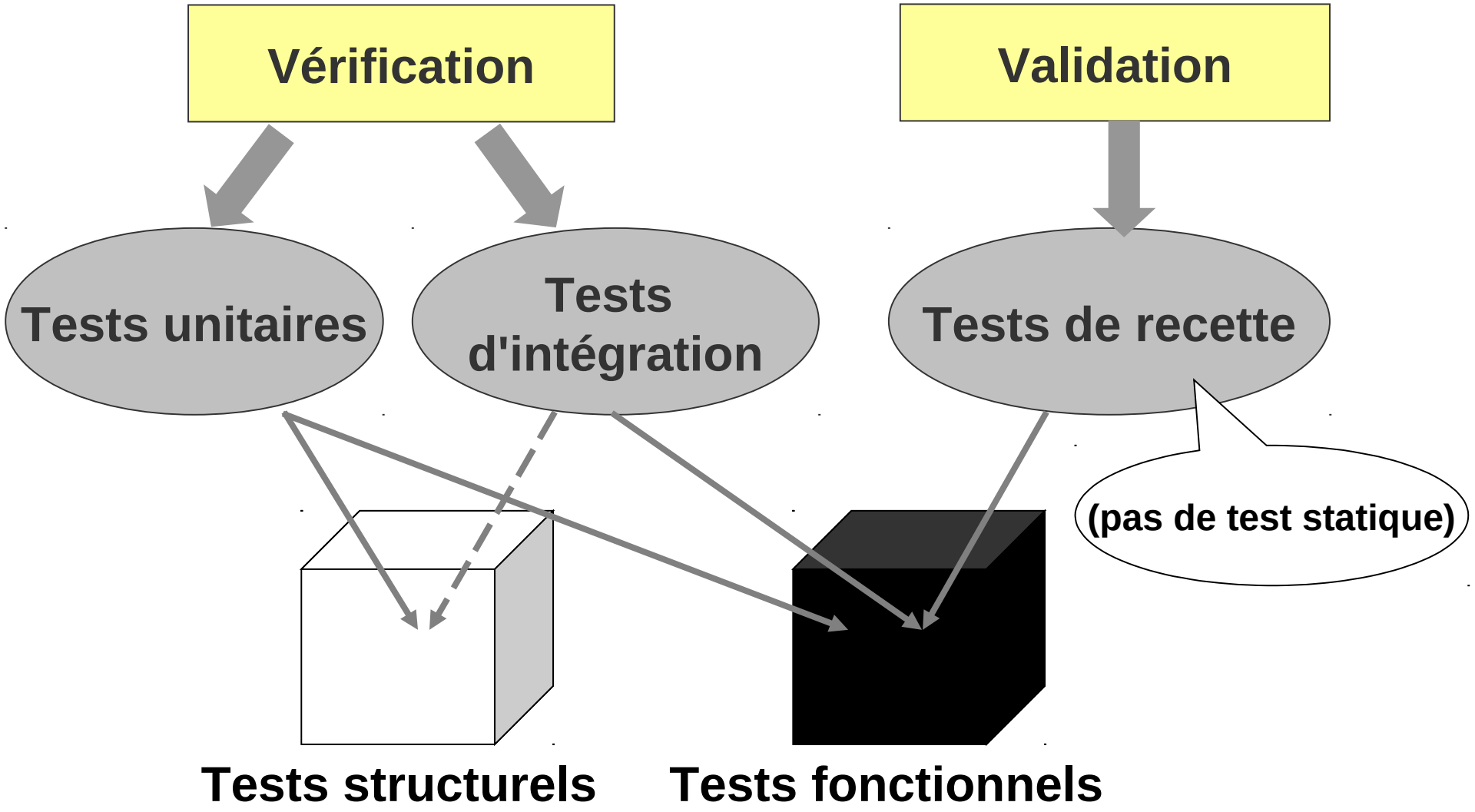
    La réalisation du programme est inconnue ; ces tests doivent être définis avant la réalisation du programme
  - le test structurel ou test de la boîte blanche

    Les tests structurels nécessitent la connaissance de la réalisation du programme.

  ➔ Une fois les tests fonctionnels réalisés, il se peut que certains branchements, voire certaines instructions n'aient jamais été exécutées, et les tests fonctionnels n'ont pu détecter aucune erreur sur ces parties de programme. Les tests fonctionnels doivent absolument être complétés par des tests structurels pour expérimenter la totalité du programme.

  ➔ Et les tests structurels doivent être complétés par des tests fonctionnels au cas où le traitement d'une partie des spécifications a été omis.

Test dynamique



Test fonctionnel

- Cas d'une fonction

  fonction trouver_minimum(A, B, C : réel) : réel

  1°/ identifier les données : les paramètres A, B et C

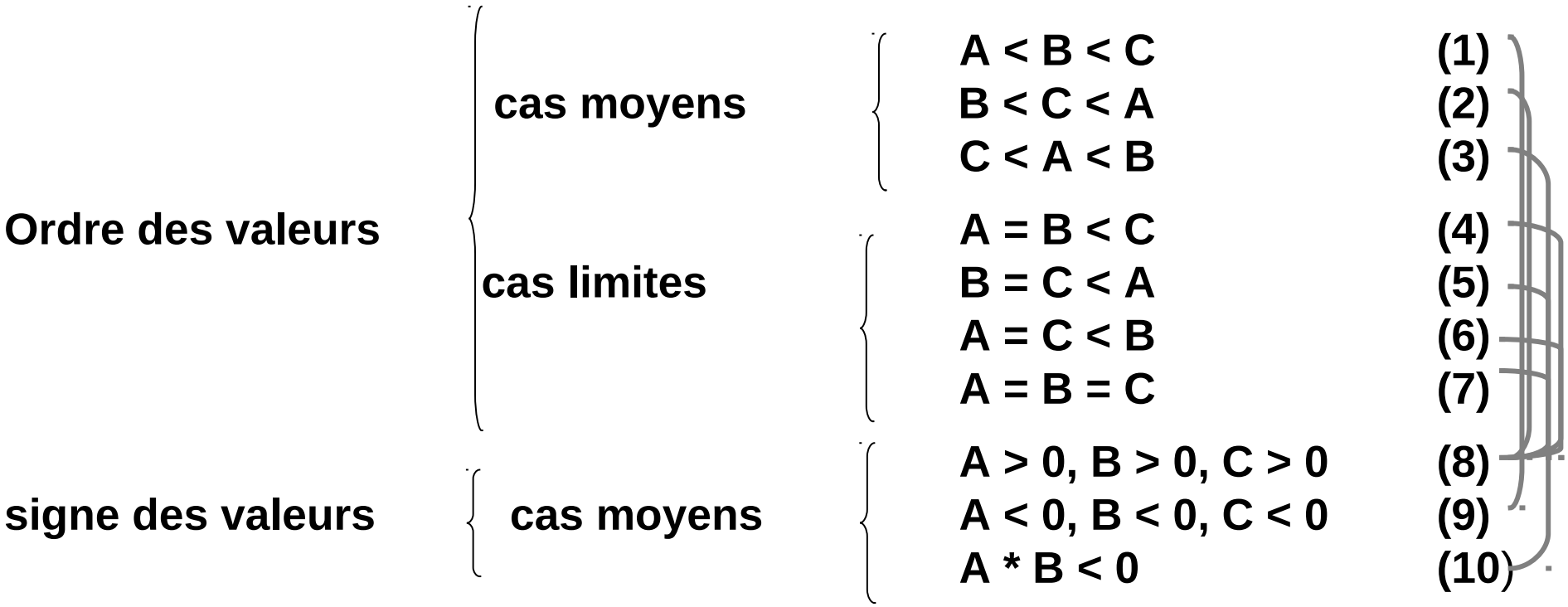
  2°/ pour chaque donnée, identifier les types d'informations caractéristiques significatifs pour le déroulement de la fonction :
  - ordre des valeurs, exemple : $A < B \leq C$
  - signe des valeurs, exemple : $A < 0$

  3°/ Pour chaque type, partitionner en domaines correspondant à des modes de fonctionnement ou de résultats distincts (domaines moyens et domaines limites) :

  4°/ choisir parmi les produits cartésiens de domaines, les classes d'équivalence définissant les tests

  5°/ identifier pour chaque classe d'équivalence, un jeu de données

Test fonctionnel



Test fonctionnel

- Cas d'un module

  module de gestion d'une pile avec les opérations dépiler et vide

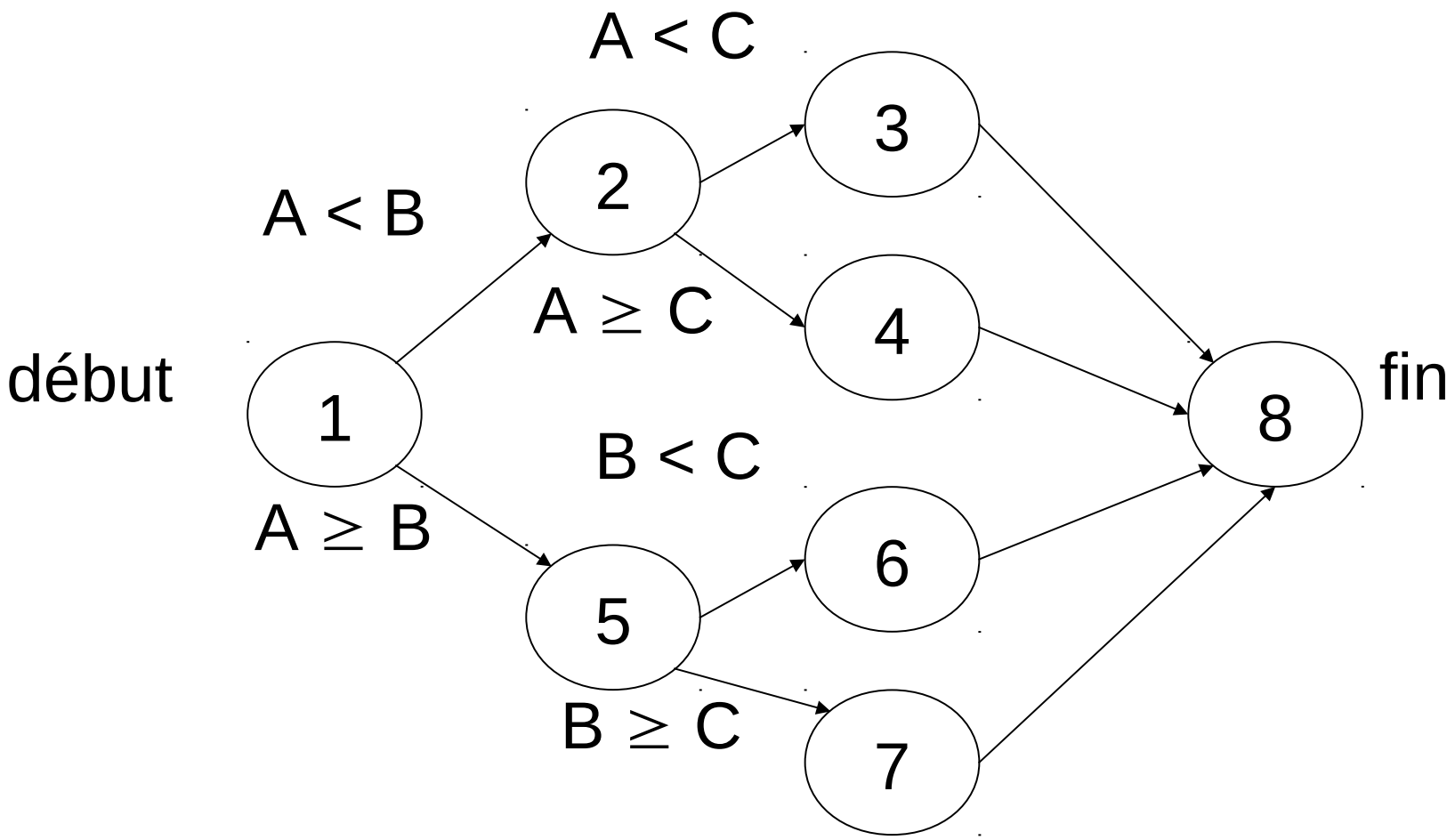
  - L'enchaînement des 2 primitives dépiler, puis vide, permet de tester à la fois le dépilement dans une pile comportant un seul élément, et le test d'une pile dont le dernier élément vient d'être dépilé.
  - Les tests des constructeurs et des transformateurs sont spécifiés en premier car ils modifient l'état du module, tous les attributs (modifiés ou non) doivent être observés en utilisant les accesseurs.

Test structurel

- Le graphe de contrôle d'un programme permet de représenter l'ensemble des cheminements possibles correspondant à son déroulement.

```
Fonction trouver_minimum( A, B, C : réel) : réel
début
        si A < B alors
                si A < C alors
                        trouver_minimum ← A
                sinon
                        trouver_minimum ← C
                finsi
        sinon
                si B < C alors
                        trouver_minimum ← B
                sinon
                        trouver_minimum ← C
                finsi
        finsi
fin
```

Test structurel



Graphe de contrôle

Test structurel

- Exigences de couverture
  - exigence minimale : couverture de toutes les branches
  - exigence maximale : impossible quand il existe une boucle

➔ Règle de couverture de l'alternative

Les jeux de test doivent permettre au minimum, si possible, d'excéuter chacune des branches définies par l'alternative

➔ Règle de couverture de la répétition

Les jeux de test doivent permettre d'exécuter le corps de boucle 0 fois, 1 fois, le nombre maximal de fois si possible ou 2 fois sinon

➔ Règle de couverture du prédicat

Couverture totale

Il est possible d'automatiser l'analyse de couverture :

- en utilisant un outil d'analyse qui vérifie a posteriori que les jeux de tests assurent bien le principe de couverture
- ou en instrumentant le code exécutable

Test dynamique aléatoire

- Il existe également des méthodes de test aléatoire pour lesquelles les jeux de test sont obtenus par tirage selon une loi de probabilité qui peut être :
  - uniforme sur le domaine d'entrée ➔ test aléatoire uniforme
  - similaire à la distribution des données d'entrée estimée lors de l'exploitation future ➔ test aléatoire opérationnel
- Les résultats expérimentaux du test aléatoire uniforme sont inégaux : le test aléatoire permet d'assurer une bonne couverture sur des programmes où les domaines d'entrée des divers chemins du test structurel sont de probabilités comparables ; il doit alors être complété par des tests des valeurs aux/hors limites ou exceptionnelles
- Le test aléatoire opérationnel permet d'utiliser des modèles de croissance de fiabilité fondés sur le relevé du nombre de défaillances observées du logiciel sous test en fonction du temps d'exécution : il n'est pas forcément utilisé pour la détection de fautes, il peut permettre de décider de l'arrêt du test quand le niveau de fiabilité estimé est suffisant

Tests de non régression

- Une modification d'un programme au cours de la mise au point ou de la maintenance peut faire apparaître une erreur dans une autre unité de programme utilisant le module modifié. La modification, au lieu d'améliorer le programme a provoqué une régression.
- Pour éviter les régressions, il faut tester chaque module sur lequel le module modifié a une incidence directe (appel) ou indirecte (modification d'attribut statique accessible)

Conception des tests

- Les tests fonctionnels sont conçus dès la spécification ou la conception préliminaire (ils seront réalisés après le codage)
  - ➔ analyse des spécifications
  - ➔ aide pour les concepteurs
  - ➔ modification des spécifications pour faciliter la réutilisation ultérieure des tests
  - ➔ outil de validation pour le client
- Les tests structurels sont conçus et réalisés au moment du codage

Réalisation des tests fonctionnels

- Il est indispensable de compléter ou remplacer la vérification manuelle des tests par une vérification par programme :
  - Réaliser un ou plusieurs programmes qui exécutent les séquences de test et vérifient les résultats, les tests sont ainsi re-exécutables. Les données peuvent être insérées dans le programme, ou mieux lues dans un fichier : elles sont alors modifiables. ➔ comparateurs de fichiers
  - Insérer la vérification dans le programme, par exemple la vérification des pré- et post-conditions ➔outils d'activation/désactivation des instruments

Réalisation des tests fonctionnels

Le débogueur réalise une instrumentation automatique minimale qui aide à la mise au point (vérifications en mode interactif), mais ne peut remplacer l'instrumentation volontaire.

L'instrumentation est souvent spécifique à l'application et l'expertise de l'équipe de développement pour un certain type d'application est importante

- La soumission des jeux de tests est une activité coûteuse (programmation de modules fictifs pour activer des composants réalisés ou simuler des composants manquants) ➔ l'instrumentation doit avoir son propre processus de développement.
- L'instrumentation peut perturber l'exécution des tests en consommant des ressources du système

Exercice

Soit la fonction suivante en langage C :

```
void mystere ( int *tab, int nb_elements, int taille_max_tab, int e)
                    {           int i, j;
                                if ( nb_elements < taille_max_tab ){
                                            i = 0 ;
                                            while ( tab[ i ] > e )
                                                        i++ ;
                                            for ( j = nb_elements ; j > i ; j-- )
                                                        tab[ j ] = tab[ j - 1 ] ;
                                            tab[ i] = e ;
                                }
                    }
```

a) Donnez un jeu d'essai qui satisfasse le critère de couverture des instructions. Justifiez votre réponse.

b) Le jeu défini précédemment satsifait-il le critère de couverture des arcs ? Justifiez votre réponse.

c) On donne la pré-condition suivante pour cette fonction :

pour tout i compris entre 0 et nb_elements-2 (tab [i] >= tab [i + 1])

Donnez la post-condition pour cette fonction.

d) Ce jeu est-il suffisant pour le test fonctionnel de la fonction ? Justifiez votre réponse. En particulier, si le jeu est insuffisant, expliquez comment il doit être complété.