



Exercice 1 On donne la spécification suivante : La fonction $somme(n, maxint)$ retourne la somme des n premiers entiers tant que celle-ci est inférieure à $maxint$. Si n est négatif, on considère sa valeur absolue. Sinon la fonction renvoie une valeur *erreur*.

1. Déterminer les classes d'équivalences permettant de définir les cas de tests (identifier les classes valides et les classes invalides). On supposera que les types du langage utilisé sont *int*, *float* et *string*.
2. Définir des données de test et l'oracle correspondant pour les cas de test identifiés.

Exercice 2 La classe *Pile* réalise une implémentation d'une pile bornée à Max éléments. Elle contient les opérations *empiler* (ajoute un élément x sur la pile, *dépiler* (supprime le sommet de pile), *sommet* (renvoie le sommet de pile). Les fonctions *vide* (teste si la pile est vide) et *pleine* sont aussi définies.

1. Définir les profils des opérations et leurs domaines de définition. En donner une spécification en indiquant à chaque fois *precondition* et *postcondition*. (vous pouvez utiliser les fonctions de la classe pour écrire les pré et post-conditions).
2. Définir les classes d'équivalence pour tester ces opérations et donner une suite de test permettant de tester la classe en utilisant notamment les cas limites.

Exercice 3 La classe *Compte* définit un compte en banque avec trois attributs privés, *nom* une chaîne non vide de caractères alphabétiques majuscules, *num* un entier positif inférieur à 1000000 et *solde* un réel positif ou nul. Les deux opérations publiques sont *crediter* et *debiter* qui prennent chacune un argument positif ou nul et modifie le solde en conséquence. Par ailleurs on suppose que le solde ne peut être inférieur à un montant $soldemin=100$ euros. La classe a un constructeur *Compte* (*String*, *int*, *float*) qui affecte les champs privés avec les valeurs passées en paramètre.

1. Spécifier chaque opération de la classe.
2. Définir les classes d'équivalence et les cas limites pour définir une suite de test pour cette classe.

Exercice 4 Donner une spécification d'une méthode qui prend un tableau d'éléments quelconques et renvoie un nouveau tableau dans lequel l'ordre des éléments est inversé, on renvoie null si le tableau initial est null.

1. Définir une suite de tests pour cette méthode. Pour cela vous identifierez des classes d'équivalence, et les cas limites associés.
2. Dérouler votre suite de test sur la méthode suivante :

```
public static Object[] (Object[] a){
    if (a==null) return null;
    Object [] b=new Object[a.length];
    for (int i=0;i<a.length;i++){
        b[i]=a[a.length -1-i];
    }
    return b;
}
```

Exercice 5 On donne le règlement d'obtention de diplôme master : *Si un étudiant a tous ces modules avec au moins 10/20 à chaque module et au moins 12/20 de moyenne générale, il obtient son diplôme. Sinon, il peut passer des rattrapages s'il a au moins 8/20 de moyenne générale, sinon il redouble. De plus, certains modules sont prioritaires, et moins de 8/20 de moyenne à ces modules empêche de passer les rattrapages et fait redoubler. Un étudiant qui réussit ses rattrapages passe.*

1. Définir les causes et les effets contenus dans la spécification.
2. Dessiner le graphe Cause/Effet.

Exercice 6 [(examen 2010)] Un système de contrôle de passage à niveau pour des trains de banlieue est défini ainsi : Les trains ne circulent que dans une direction E-O (ligne unidirectionnelle) et on distingue trois zones autour du passage à niveau :

1. zone à l'approche entre 1000 et 500m **avant** le passage à niveau,
2. zone *critique* entre 500 et 0m **avant** le passage à niveau,
3. zone *occupation* de 0 à 200m **après** le passage à niveau.

Les trains sont des trains courts (100m) ou longs (200m) et peuvent se suivre à une distance minimale de 400m grâce à un système de contrôle de distance. Trois capteurs C_1, C_2, C_3 à 1000m et 500m avant et 200m après le passage à niveau permettent de déclencher des événements signalant le passage du train au niveau du capteur. Dans son état d'attente, le passage à niveau a sa barrière ouverte et son feu est vert. Le fonctionnement des capteurs lorsque la tête du train arrive à leur niveau est décrit ainsi :

1. En C_1 : un évènement *train à l'approche* est produit, l'action *mettre feu au rouge* est déclenchée.
2. En C_2 : un évènement *train en zone critique* est produit et l'action *baisser barrière* est déclenchée. Si le feu est *vert*, un évènement *danger* est produit, on déclenche l'envoi d'un message *arrêt immédiat*, le feu est mis au rouge et la barrière baissée, puis le système se met dans un état de blocage.
3. En C_3 : l'évènement *libre* est produit, les actions *mettre feu au vert* et *lever barrière* sont déclenchées.

Une action commandant la couleur du feu est instantanée et a comme résultat de mettre le feu sur la couleur indiquée. Une action baisser barrière a pour effet de baisser la barrière si elle n'est pas déjà dans fermée sauf si un obstacle est présent (dans ce cas la barrière reste levée). Le fonctionnement est similaire pour lever la barrière. Si le système est dans un état de blocage, il y reste quels que soient les évènements qui se produisent.

1. Définir les évènements et les actions définissant le système de contrôle du passage à niveau. Attention : les grandeurs données dans l'énoncé sont indicatives et ne sont pas utiles à la modélisation.
2. Donner un automate évènement/action permettant de modéliser le système.
3. La situation suivante est-elle possible : les barrières sont relevées, le feu est au vert et un train est en zone critique ? Si oui expliquer pourquoi et suggérer une modification de la spécification pour la corriger, si non le prouver avec votre modélisation.