Examen GL02 – A17 (Aucun document autorisé – Indiquez les réponses sur votre copie)

A/ Spécifier (10 pts)

1/ Expliquez à partir d'un exemple la relation entre expressivité et ambiguïté pour différents types de spécification vue en cours.

2/ Le FrontDoor file format (FDFF) est une proposition de format de données permettant d'exprimer des règles de contrôle d'accès (en ouverture) à des portes afin de sécuriser les accès à des lieux. Voici deux exemples :

```
#GROUP:g001
employee1
employee2
#
#ACCESS:1stdoor
+accessManager2:2015-10-31=23h59s00UTC>>
+staff01:>>2018-12-31=23h59s00UTC
+visitor1:28-12-2018=11h59s0UTC>>2018-12-31=23h59s00UTC
*g001:>>2018-07-31=11h59s00UTC
IF %USER% = g001
IF %TIME% > %TODAY%=20h30s00UTC AND %TIME% < %TODAY+1%08h30s00UTC
CLOSE
ENDIF
ENDIF
#
```

basement.fdff

```
#ACCESS:mainEntrance
+accessManager2:2015-10-31=23h59s00UTC>>
*employeeWithCar:>>2018-12-31=23h59s00UTC
IF %TIME% > %TODAY%=21h00s00UTC AND %TIME% < %TODAY+1%08h00s00UTC
CLOSE
ENDIF
#
#ACCESS:emergencyEntrance
+accessManager1:2015-10-31=23h59s00UTC>>
#
#GROUP:employeeWithCar
employee2
employee3
employee4
staff5
#
```

parking.fdff

Le FDFF permet d'exprimer des autorisations d'accès pour des personnes ou des groupes avec une date et heure de début sans limitation (%TODAY%>>), ou seulement avec une date et heure de fin pour la révocation du droit d'accès (>>%TODAY+1%). Il peut même exprimer un accès temporaire (%TODAY%>>%TODAY+1%). Par ailleurs, il est possible au travers de clauses conditionnelles de formuler des exceptions (refusant l'accès avec l'instruction CLOSE), par exemple pour empêcher l'accès aux locaux en dehors des heures de bureau. A ce titre un certain nombre de variable par défaut doivent être prise en compte : %USER% l'utilisateur sur lequel le contrôle d'accès est demandé (l'implémentation vérifiera son groupe par extension) ; %TIME% exprime la date et heure courante lors du contrôle d'accès ; %TODAY% la date du jour (heure par défaut 23h59s00UTC) ; %TODAY+1% le lendemain du jour courant.

Pour simplifier le cas, on ne considère que le fuseau horaire UTC. Le FDFF fonctionnant en liste blanche, il est obligatoire qu'il y ait toujours l'expression d'un bloc "#ACCESS" dans un fichier. Par commodité on ne tiendra pas compte de la casse.

A partir de la description du format et des exemples, proposez une grammaire au format ABNF pour le format FDFF.

3/ La Sealand Republic Aeronautic and Space Agency (SRASA) a mis au point un nouveau module radar permettant de surveiller l'espace aérien proche (nul besoin de tenir compte de la courbure de la Terre) et de déterminer la distance approximative (ie, sqrt([x1-x2]² + [y1-y2]²)) des objets détectés par rapport au lieu où le radar est installé (latitude et longitude en coordonnées décimales, resp. [-90.0,90.0] et [-180.0,180.0]). Voici les spécifications algébriques de Detected Object et de Module Radar.

Titre : Detected Object **Sorte : ...**
Références : ...
Description : Un objet détecté est construit avec l'opération CreateDO à partir de ses coordonnées en latitude et longitude. Deux opérations, Lat et Lng, permettent d'obtenir les coordonnées. La distance entre un objet détecté et un point défini en latitude et longitude est calculée avec l'opération DistanceApprox.

---------- SIGNATURE ----------

CreateDO : ...
Lat : ...
Lng : ...
DistanceApprox : ... X ... X FLOAT --> FLOAT

---------- AXIOMES ----------

Latitude(CreateDO(lat, lng)) = lat
Longitude(CreateDO(lat, lng)) = lng
…

Titre : Module Radar **Sorte : …**
Références : ...
Description : Un radar est un type complexe comprenant un point géoréférencé ainsi qu'un ensemble d'objets détectés. Il est créé vide. Add ajoute un objet détecté. LatMR et LngMR permettent d'obtenir les coordonnées du radar. Size compte les objets détectés. L'opération "Object(s) Nearer Than n" (ONT) permet d'obtenir un radar avec l'ensemble des objets détectés qui sont à une distance strictement inférieure à celle transmise en argument.

---------- SIGNATURE ----------

Create : ... | LatMR : MR --> FLOAT
Add : ... | LngMR : MR --> FLOAT
Size : ... | **ONT : ... --> MR**

---------- AXIOMES ----------

LatMR(Create(lt, lg)) = lt | LngMR(Create(lt, lg)) = lg
LatMR(Add(m,do)) = LatMR(m) | LngMR(Add(m,do)) = LngMR(m)
ONT(Create(lt, lg), n) = Create(lt, lg) **…**

*Complétez (sur votre copie, pas sur le sujet !) les éléments en **gras/…** selon les points suivants :*

a) Complétez les champs Sorte, Références ainsi que les signatures d'opérations manquantes ou incomplètes.
b) Complétez les axiomes pour chaque spécification.
c) Montrer sur le cas d'un module radar comprenant 3 objets détectés (ex : rdr = { obj300, obj200, obj800 }, avec obj300 se situant à une distance de 300.0 m et ainsi de suite), la construction d'un module radar comme sous-ensemble d'objets détectés de distance inférieure à 400.0 m via l'opération ONT.

B/ Implémenter (10 pts)

```
var data = "note: 5\nnote: 3\nnote: 1";
function tokenize(input){
        var separator = /(\n|: )/;
        input = input.split(separator);
        input = input.filter(val => !
val.match(separator));
        return input;
}
```

1/ D'après le programme ci-contre, quelle est la valeur affichée par `console.log(tokenize(data));` ? (1 réponse, w|x|y|z) **Expliquez votre réponse**.

w. "note5note3note1" | x. "note: 5\nnote: 3\nnote: 1" |
y. Rien. | z. ['note', '5', 'note', '3', 'note', '1']

2/
```
var products = [{name: "Chocobons", price: 120, quantity: 12, id: "A1"}];
module.exports = {
        getProduct: function(productId) {
                var product = products.find(p => p.id === productId);
                return product;
        },
};
```

a) Le module `productInventory.js` (ci-dessus) est importé avec succès à l'aide de l'instruction "`var inventory = require("./productInventory.js");`" dans `program.js`. Indiquez la manière appropriée de récupérer le prix des `Chocobons` depuis `program.js`. (1 réponse, w|x|y|z):

w. `var p = products[0].price` | x. `var p = inventory.getProduct("A1").price` |
y. `var p = inventory.products[0].price` | z. Aucune, il y a une erreur de syntaxe

b) Justifiez votre réponse en expliquant ce qui sera effectivement disponible pour program.js lors de l'appel à `require`.

3/ En vous basant sur la structure de données des produits décrites dans productInventory.js (supposons qu'il y en a bien plus).

a) Ajoutez une fonction `unavailableProducts` au module `productInventory.js` permettant de retourner l'ensemble des produits non disponibles, c'est-à-dire de quantité 0.

b) Ajoutez une fonction `averagePrice` au module `productInventory.js` pour calculer la moyenne des prix de l'ensemble des produits à l'inventaire (vous pouvez notamment utiliser la méthode `reduce(callback, valeurInitiale)` de `Array`).

4/ Sur la copie locale de travail d'un projet, la commande `svn status` indique :

```
C:\MyWorkingCopy\trunk>svn status
D       hello.txt
A       index.html
?       password.txt
```

Sur la base des informations ci-contre (faisant l'hypothèse qu'il n'y a pas de conflit), précisez pour ces 3 fichiers comment SVN va les traiter lors du prochain commit.

C/ Maintenance/Evolution (10 pts)

1/ Le module de distributeur automatique `VendingMachine.js` dispose des méthodes suivantes :

- `getBalance()`, retourne la quantité courante d'argent dans la machine (balance).
- `insertCoin(coin)`, retourne true si la pièce est de type conforme et incrémente la quantité d'argent, false sinon.
- vendProduct(refProduct), retourne le nom du produit en cas de succès, décompte l'argent que coûte le produit et conserve l'argent disponible pour une éventuelle transaction suivante. En cas d'échec, retourne une chaîne de caractère indiquant la quantité d'argent manquante et conserve l'argent disponible en attente d'un complément.
- `releaseChange()`, retourne un `Array` de pièces (ie, { `"1Euro" : 1, "50cents" : 1`}) correspondant au change de la quantité d'argent dans la machine pour la transaction à ce moment (éventuellement vide donc).

Le module n'accepte que les types de pièces conformes suivants (avec des valeurs en cents) :

```
var coinType = {"2Euro": 200, "1Euro": 100, "50cents": 50, "20cents": 20, "10cents": 10 };
```

On vous demande de compléter le groupe de test "`Vending Machine Test suite`" avec les 3 cas de tests qui suivent : a) Achat d'un produit avec le juste compte de monnaie, b) Tentative d'achat d'un produit avec une quantité d'argent insuffisante, c) Insertion de pièces sans achat de produit avec retour du change.

```
module.exports = {
        setUp: function(){
                var VendingMachine = require('../../VendingMachine.js');
                this.vm = new VendingMachine([{ name: "Chocobons", price: 120, quantity: 12, id: "A1" }], 0);
        },
        "Vending Machine Test suite": {
                "Initialization test" : function(assert){
                        assert.strictEqual(this.vm.getBalance(), 0, "Empty balance at start");
                        assert.ok(!vm.insertCoin("1Dolar"), "Unknown coin is refused - ie 1Dolar");
                        done();
                },      // Complétez la suite sur votre copie
```

2/ Dans un cahier de recette, pourquoi est-il important de préciser les environnements d'exécution utilisés pour les tests ?

3/ Si vous réutilisez tout ou partie d'un programme sous licence MIT : a) Pouvez-vous commercialiser votre nouvelle solution ?

b) Indiquez et expliquez les deux principales affirmations sur lesquels porte la licence MIT.

4/ Indiquez au moins 3 bonnes pratiques à adopter lors de la rédaction d'un ticket demandant une amélioration en justifiant de leur intérêt.