DM INFORMATIQUE THEORIQUE

Machine : Somme \* Sucre \* Boisson 🡪 (Monnaie \* Boisson sucrée) U (Erreur Machine)

Def tarifs :

* Signature : Ingrédients \* Prix 🡪 Prix2
* Axiome : Prix 2 est un dictionnaire ayant pour clés les différents types d’ingrédients, et en valeurs les prix correspondant.

nombre de clés du dictionnaire = nombre d’ingrédients

Def max\_stock :

* Signature : Ingrédient \* Max\_Stock 🡪 Max\_Stock2
* Axiome : Max\_Stock2 est un dictionnaire ayant pour clés les différents types d’ingrédients, et en valeurs les stocks maximums correspondant.

nombre de clés du dictionnaire = nombre d’ingrédients

Def stock :

* Signature : Ingrédient \* Stock 🡪 Stock2
* Axiome : Stock2 est un dictionnaire ayant pour clés les différents types d’ingrédients, et en valeurs les stocks actuels correspondant.

nombre de clés du dictionnaire = nombre d’ingrédients

MODE FONCTIONNEMENT :

Def commander :

* Signature : Monnaie \* Ingrédients 🡪 Monnaie U (Commande \* Monnaie1 \* Monnaie2)
* Axiome : Soit (a,b,c,d,e,f)6 , (g,h,i,j,k,l) 6 , (m,n,o,p,q,r) 6 et (s,t,u,v,w,x) 6 quatre tuples de binaire de 6 entiers.

Monnaie = (a,b,c,d,e,f), Ingrédients = (g,h,i,j,k,l), Monnaie1 = (m,n,o,p,q,r)

Monnaie2 = (s,t,u,v,w,x)

Si commande impossible retourner Monnaie,

Sinon retourner Commande \* Monnaie1 \* Monnaie2

Monnaie 1 = Monnaie – 2€

Monnaie1 + Monnaie2 = Monnaie – Prix(Ingrédients)

Def préparer\_commande :

* Signature : Ingrédients \* Commande \* Boisson 🡪 BoissonCommandée
* Axiome : Soit (a,b,c,d,e,f)6 , (g,h,i,j,k,l) 6 et (m,n,o,p,q,r) 6 trois tuples de binaire de 6 entiers.

Ingrédients = (a,b,c,d,e,f), Commande = (g,h,i,j,k,l), Boisson = (m,n,o,p,q,r)

BoissonCommandée = Boisson

Def vérifier\_commande :

* Signature : Commande 🡪 {V,F}
* Axiome : Soit (a,b,c,d,e,f) 6 un tuple de binaire de 6 entiers.

Commande = (a,b,c,d,e,f)

a,b,c,d,e,f {0,1}

longueur(Commande) = nombre d’ingrédients + 1

Def vérifier\_stock\_suffisant :

* Signature : Commande \* Stock 🡪 {V,F}
* Axiome : Soit (a,b,c,d,e,f) 6 un tuple de binaire de 6 entiers.

Commande = (a,b,c,d,e,f)

Soit Stock un dictionnaire contenant en clé le nom de l’ingrédient et en valeur le stock restant correspondant.

Commande[i] ≤ Stock[i]

Def vérifier\_monnaie :

* Signature : Monnaie 🡪 {V,F}
* Axiome : Soit (a,b,c,d,e,f) 6 un tuple de binaire de 6 entiers.

Monnaie = (a,b,c,d,e,f)

Longueur(Monnaie) = nombre de pièces différentes

Def ramener\_deux\_euros :

* Signature : Monnaie 🡪 Monnaie1 \* Monnaie2
* Axiome : Soit (a,b,c,d,e,f)6 , (g,h,i,j,k,l) 6 et (m,n,o,p,q,r) 6 trois tuples de binaire de 6 entiers.

Monnaie = (a,b,c,d,e,f), Monnaie1 = (g,h,i,j,k,l), Monnaie2 = (m,n,o,p,q,r)

g ≤ a, h ≤ b, I ≤ c, j ≤ d, k ≤ e, l ≤ f,

somme(m,n,o,p,q,r)=2

somme(a,b,c,d,e,f)=somme(g,h,I,j,k,l)+2

Monnaie[i]=Monnaie1[i]+Monnaie2[i]

Def rendre\_monnaie :

* Signature : Monnaie \* Stock(monnaie) 🡪 Monnaie2
* Axiome : Soit (a,b,c,d,e,f)6 et (g,h,i,j,k,l) 6 deux tuples de binaire de 6 entiers.

Monnaie = (a,b,c,d,e,f), Monnaie2 = (g,h,i,j,k,l)

Monnaie[i] ≤ Sotck[i]

somme(a,b,c,d,e,f) = somme(g,h,I,j,k,l)-prix(commande)

Def vérifier\_rendu\_monnaie\_possible :

* Signature : Monnaie \* PrixBoisson 🡪{V,F}
* Axiome : Soit (a,b,c,d,e,f) 6 un tuple de binaire de 6 entiers.

Monnaie = (a,b,c,d,e,f)

Si ∃ (g,h,i,j,k,l) 6 un tuple de binaire de 6 entiers tel que

(g,h,i,j,k,l) = Monnaie – PrixBoisson, alors retourner Vrai

Sinon retourner Faux

Def formater\_commande :

* Signature : Commande 🡪 CommandeAjustée
* Axiome : Soit (a,b,c,d,e,f) 6 un tuple de binaire de 6 entiers. Soit (g,h,i,j,k) 5 un tuple de 5 entiers appartenant à [0 , 3].

Commande = (a,b,c,d,e,f) et CommandeAjustée = (g,h,i,j,k)

g = transformation du tuple (a,b) binaire en entier, h=c, i=d, j=e, k=f.

Def get\_prix\_boisson :

* Signature : Commande 🡪 Somme
* Axiome : Soit (a,b,c,d,e) 5 un tuple de binaire de 6 entiers.

Commande = (a,b,c,d,e)

Somme = prix[ingrédient]\*Commande[ingrédient]

Def get\_ingredients :

* Signature : Commande 🡪 Commande
* Axiome : Soit (a,b,c,d,e,f) 6 un tuple de binaire de 6 entiers.

Commande = (a,b,c,d,e,f)

Pour i dans [0 ;5] : Si Commande[i] ≠ 0 afficher Ingrédients[i]

Def correspondance\_boisson :

* Signature : Ingrédients 🡪{V,F}
* Axiome :

Pour i dans [0 ;5] : Si Ingrédients[i] composition d’une boisson α et α[Ingrédients] Ingrédients alors retourner Vrai

Retourner Faux

MODE MAINTENANCE :

Def changer\_prix\_unitaire :

* Signature : Ingrédient \* Nouveau\_Prix 🡪 Prix\_Unitaire
* Axiome : Prix\_Unitaire = Nouveau\_Prix

Def prix\_unitaire :

* Signature : Ingrédient 🡪 Prix
* Axiome : Prix = Prix[Ingrédient]

Def set\_max\_stock :

* Signature : Ingrédient \* Nouveau\_Max\_Stock 🡪 Max\_Stock
* Axiome : Max\_Stock[Ingrédient] = Nouveau\_Max\_Stock

Def reset :

* Signature : Distributeur 🡺 Distributeur
* Axiome : ancien historique = nouvel historique

Def vider\_caisse :

* Signature : Caisse 🡪Caisse\_vide
* Axiome : Soit (a,b,c,d,e,f) 6 un tuple de binaire de 6 entiers.

Caisse = (a,b,c,d,e,f)

Caisse\_vide = (0,0,0,0,0,0)

Def get\_stock :

* Signature : Ingrédient 🡪 Stock
* Axiome : Ingrédient = Sucre, ou Ingrédient = Lait, ou Ingrédient = Thé, ou Ingrédient = Chocolat, ou Ingrédient = Café.

Stock ≤ Max\_Stock

Def get\_all\_stock :

* Signature : Ingrédients \* Stock 🡪 Stock2
* Axiome : Soit Stock2 un dictionnaire ayant pour clé le nom de chaque ingrédients et pour valeur le stock correspondant.

nombre de clés du dictionnaire = nombre d’ingrédients

Def remplir\_stock :

* Signature : Ingrédient \* Max\_Stock 🡪 Stock
* Axiome : Stock(Ingrédient) = Max\_stock

Def remplir\_tout\_stock :

* Signature : Ingrédients \* Max\_Stock 🡪 Stocks
* Axiome : Stocks[i] = Max\_Stock[i]

Def ajouter\_stock :

* Signature : Stock \*Max\_Stock \* Quantité 🡪Nouveau\_Stock
* Axiome : Si Stock + quantité ≤ Max\_Stock, alors Nouveau\_Stock = Stock + Quantité,

Sinon Nouveau\_Stock = Stock

Def historique :

* Signature :
* Axiome :

COMPLEXITES/ORDRE DE GRANDEUR :

Def tarifs :

* Complexité : 12
* Ordre de grandeur : O(1)

Def stocks :

* Complexité : 12
* Ordre de grandeur : O(1)

Def changer\_prix\_unitaire :

* Complexité : 3n
* Ordre de grandeur : O(n)

Def prix\_unitaire :

* Complexité : 2
* Ordre de grandeur : O(1)

Def set\_max\_stock :

* Complexité : n + 3
* Ordre de grandeur : O(n)

Def reset :

* Complexité : 2n + 25
* Ordre de grandeur : O(n)

Def vider\_caisse :

* Complexité : 5n
* Ordre de grandeur : O(n)

Def get\_stock :

* Complexité : 3
* Ordre de grandeur : O(1)

Def get\_stock\_size :

* Complexité : 3
* Ordre de grandeur : O(1)

Def get\_stock\_max :

* Complexité : 5
* Ordre de grandeur : O(1)

Def get\_all\_stock :

* Complexité : 5n
* Ordre de grandeur : O(n)

Def remplir\_stock :

* Complexité : 3n + 10
* Ordre de grandeur : O(n)

Def remplir\_tout\_stock :

* Complexité : 3n + 17
* Ordre de grandeur : O(n)

Def ajouter\_stock :

* Complexité : 19
* Ordre de grandeur : O(1)

Def get\_historique :

* Complexité : 3
* Ordre de grandeur : O(1)

Def display\_stats :

* Complexité : 5 + 6n
* Ordre de grandeur : O(n)

Def verifier\_commande :

* Complexité : 11 + 4n
* Ordre de grandeur : O(n)

Def trad :

* Complexité : 23
* Ordre de grandeur : O(1)

Def \_\_get\_boites :

* Complexité : 4n + 5
* Ordre de grandeur : O(n)

Def \_\_verifier\_monnaie :

* Complexité : 15 + 20n +12n²
* Ordre de grandeur : O(n²)

Def \_\_verifier\_rendu\_monnaie\_possible :

* Complexité : 6 + 20n + 13n²
* Ordre de grandeur : O(n²)

Def match :

* Complexité : 8n² + 5n + 2
* Ordre de grandeur : O(n²)

Def \_calculer\_prix\_boisson :

* Complexité : 3n3 - 3n²
* Ordre de grandeur : O(n3)

Def calculer\_prix\_boisson :

* Complexité : 24 + 3n3 - n²
* Ordre de grandeur : O(n3)

Def \_\_verifier\_stock-suffisant :

* Complexité : 5n + 2
* Ordre de grandeur : O(n)

Def \_\_preparer\_commande :

* Complexité : 22 + 17n + 8n²
* Ordre de grandeur : O(n²)

Def commander :

* Complexité : 91 + 80n + 41n² + n3
* Ordre de grandeur : O(n3)