PIM : Mini-projet 1

Auteur : Ewen Le Bihan

Temps passé sur les raffinages : 3 h 30
Temps passé sur la programmation : 3 h 0
Temps passé sur la mise au point : 0 h 20

Raffinages

On ne donnera pas d'exemple car ils vont prendre beaucoup de place. Il est cependant utile d'en prendre pour vérifier la bonne compréhension de l'énoncé.

```
R0: Réviser les tables de multiplication

R1: Comment «Réviser les tables de multiplication» ?
    Répéter
       Demander la table à réviser                              Table: out Entier
       Réviser la table de multiplication                       Table: in Entier, Erreurs: out Entier
       Afficher un message selon le nombre d'erreurs            Erreurs: in Entier
       Demander à continuer                                     Continuer: out Booléen
    JusquÀ Non Continuer

R2: Comment «Demander la table à réviser» ?
    Écrire("Table à réviser : ")
    Lire(Table)
    Si Table < 1 OuSinon Table > 10 Alors
       Écrire("Impossible. La table doit être entre 0 et 10.")
       Demander la table à réviser
    FinSi

R2: Comment «Réviser la table de multiplication» ?
    Erreurs <-- 0                                               Erreurs: out Entier
    Pour i allant de 1 à 10 Faire
       Tirer un nombre aléatoire entre 0 et 10                  Valeur: out Entier
       Réviser la valeur d'une multiplication                   Valeur, Table: in Entier, Erreurs: in out
Entier
    FinPour

R2: Comment «Afficher un message selon le nombre d'erreurs» ?
    Selon Erreurs Dans
       0      => Écrire("Aucune erreur. Excellent !")
       1      => Écrire("Une seule erreur. Très bien.")
       2..4   => Écrire("Seulement ", 10 - Erreurs, " bonne réponses. Il faut apprendre la table de ",
Table, " !")
       5..9   => Écrire(Erreurs, " erreurs. Il faut encore travailler la table de ", Table, ".")
       10     => Écrire("Tout est faux! Volontaire?")
       Autres => Rien  -- impossible
    FinSelon

R2: Comment «Demander à continuer» ?
    Écrire("On continue (o/n) ?")
    Lire(Réponse)                                               Réponse: out Caractère
    Continuer <-- Réponse = 'o' OU Réponse = 'O'                Continuer: out Booléen
```

```
R3: Comment «Réviser la valeur d'une multiplication» ?
    Écrire("(M", Valeur, ") ", Table, " * ", Valeur, " ? ")
    Lire(Réponse)                                          Réponse: out Entier
    Si Réponse /= Table * Valeur Alors
       Erreurs <-- Erreurs + 1                             Erreurs: in out Entier
    FinSi
```

Evaluation des raffinages

		Evaluation Etudiant (I/P/A/+)	**Justification / commentaire**	**Evaluation Enseignant (I/P/A/+)**
Forme (D-21)	Respect de la syntaxe Ri : Comment "... une action complexe ..." ? des actions combinées avec des structures de controle Rj : ...	+		
	Verbe à l'infinitif pour les actions complexes	+		
	Nom ou équivalent pour expressions complexes	+		
	Tous les Ri sont écrits contre la marge et espacés	+		
	Les flots de données sont définis	+		
	Une seule décision ou répétition par raffinage	A		
	Pas trop d'actions dans un raffinage (moins de 6)	A		
	Bonne présentation des structures de contrôle	A		
Fond (D21-D22)	Le vocabulaire est précis	+		
	Le raffinage d'une action décrit complètement cette action	+		
	Le raffinage d'une action ne décrit que cette action	+		
	Les flots de données sont cohérents	+		
	Pas de structure de contrôle déguisée	+		
	Qualité des actions complexes	+		

Evaluation du code

		Consigne : Mettre O (oui) ou N (non) dans la colonne Etudiant suivant que la règle a été respectée ou non. Une justification peut être ajoutée dans la colonne “commentaire”.	
Commentaire	**Etudiant (O/N)**	**Règle**	**Enseignant (O/N)**
	o	Le programme ne doit pas contenir d'erreur de compilation.	
	o	Le programme doit compiler sans messages d’avertissement.	
	o	Le code doit être bien indenté.	
Doit-on toujours mettre une branche Sinon, même vide?	O	Les règles de programmation du cours doivent être respectées : toujours un Sinon pour un Si, pas de sortie au milieu d'une répétition...	
	o	Pas de code redondant.	
	o	On doit utiliser les structures de contrôle adaptées (Si/Selon/TantQue/Répéter/Pour)	
	o	Utiliser des constantes nommées plutôt que des constantes littérales.	
	o	Les raffinages doivent être respectés dans le programme.	
	o	Les actions complexes doivent apparaître sous forme de commentaires placés AVANT les instructions correspondantes.	
		Une ligne blanche doit séparer les principales actions complexes	
	o	Le rôle des variables doit être explicité à leur déclaration (commentaire).	

Prise en compte d’évolutions possibles

Répondre de manière concise et précise aux questions posées. Ces évolutions ne doivent pas être implantées dans votre programme.

Question 1 : Au lieu de poser 10 questions, on veut en poser 15. Comment faire ?

Réponse : On change la valeur de la constante `Questions_Per_Table`.

Question 2 : On veut afficher "Bien" si l'utilisateur n'a commis que 2 ou 3 erreurs. Comment modifier le programme ?

Réponse : Rajouter une branche `when 2..3` dans le `case Errors` et changer `when 2..4` en `when 4`.

Question 3 : On veut donner la possibilité à l'utilisateur d'abandonner le programme en tapant -1 quand on lui demande le résultat d'une multiplication. Quelles modifications faut-il alors faire au programme ?

Réponse : Après avoir posé la multiplication à l'utilisateur, on compare sa réponse, si elle vaut -1, on met `Continue` à `False`. On place tout le reste de la boucle sous un `if Continue`.

Question 4 : On veut faire réviser les tables de 0 à 20. Comment modifier le programme ?

Réponse : On change la condition sur `Table` pour accepter jusqu'à la table de 20, et on instancie `Alea` avec `(0, 20)` pour pouvoir poser des multiplications au-delà de 20 × 10.

Question 5 : À la fin d'une série de questions, on veut proposer à l'utilisateur de réviser la table pour laquelle l'utilisateur a commis le plus d'erreurs. Par exemple, s'il se trompe pour 3 * 5, on compte une erreur pour 5 mais pas pour 3. Comment faire ?

Réponse : Sans utiliser des tableaux, on doit créer une variable comptant le nombre d'erreurs pour chaque nombre (`Item`) de 0 à 10. À chaque fois que l'utilisateur se trompe, on met à jour la bonne variable. On suggère ensuite la table correspondant à la variable ayant la plus grande valeur.

Question 6 : Si l’utilisateur fait 4 erreurs, on arrête de poser les multiplications en disant qu’il faut aller apprendre la table. Comment modifier le programme ?

Réponse : Similairement à la question 3, on compare le nombre d'erreurs, si il vaut 4, on affiche un message et on met `Continue` à `False`.

Question 7 : Comment l'extension 1 a été prise en compte (pas de fois de suite la même multiplication) ?

Réponse : Les tableaux étant interdits, on a considéré que l'unicité ne devait être garantie qu'avec la question précédente. On stocke donc la question précédente dans une variable et on compare le nombre tiré avec cette dernière. On tire un nouveau nombre jusqu'à ce qu'il soit différent de celui de la question précédente.

Question 8 : Comment l'extension 2 a été prise en compte (proposer de réviser la table avec la plus grande hésitation) ?

Réponse : On stocke l'hésitation (durée de réponse) en récupérant le temps actuel avant et après réponse, et on met progressivement à jour l'hésitation moyenne ainsi que l'hésitation maximum. Enfin, après avoir posé toutes les questions pour une table, on compare l'héstation moyenne et maximum, et on affiche un message si leur différence est assez grande.

Difficultés rencontrées

N/A

Informations complémentaires

N/A

Bilan

N/A