PIM : Mini-projet 1

Auteur : BAURIAUD Laura (Groupe I)

Temps passé sur les raffinages : 3 h 15
Temps passé sur la programmation : 3 h 15
Temps passé sur la mise au point : 1 h 30

TODO : Nommer votre document PIM-PR1-NOM-Prénom (en utilisant votre NOM et votre Prénom).

Raffinages (iit.neeraj@gmail.com)

TO DO : écrire ici les 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**
{La table doit appartenir à [0,10] }
Poser les 10 multiplications à l'utilisateur et compter le nombre d'erreurs commises ainsi que le temps de réponse moyen erreur, tps_moy : **out**
{tps_moy correspond au temps moyen de réponse à une multiplication}
Afficher un message en fonction du nombre d'erreurs commises erreur : **in**
Afficher un message en fonction du pire temps de réponse tps_moy, pire_tpstIm, pire_nb : **in**
{Il est possible qu'aucun message ne s'affiche}
Traiter le choix continue : **out**
{continue est un booléen qui va nous permettre de savoir si l'utilisateur continue de réviser ou non}
Jusqu'à non continue

R2 : **Comment** "demander la table à réviser" ?

Demande <- Vrai
TantQue demande
Écrire ("Table à réviser : ")
Écrire (table)
demande <- (table<0 ou table>10)
{demande est un booléen qui nous permet de savoir si la valeur de table appartient ou non à [0,10] }
Si demande **Alors**
Écrire **("Impossible. La table doit être entre 0 et 10.")**
Sinon
Rien
FinSi
FinTQ

R2 : Comment “poser les 10 multiplications et compter le nombre d’erreurs commises ainsi que le temps de réponse moyen”?

```
Erreur <- 0
Tps_moy <- 0
{tps_moy qui contient le temps moyen de réponse à une multiplication}
Pire_tps <- 0
{pire_tps est la valeur du pire temps de réponse à une multiplication proposée}
Pire_nb <- 0
{pire_nb est la valeur pour laquelle l’utilisateur à mis le plus de temps à répondre à la multiplication}
Pour i De 1 À 10 Faire
    Poser une multiplication        table : in ; erreur, tps_moy : in/out ; tps, nb : out
    {tps contient le temps pour effectuer la multiplication et nb le nombre de droite de la multiplication}
    Si tps > pire_tps Alors
        Pire_tps <- tps
        Pire_nb <- nb
    Sinon
        Rien
    FinSi
    Tps_moy <- tps_moy + tps
FinPour
Tps_moy <- tps_moy / 10
```

R2 : Comment “afficher un message en fonction du nombre d’erreurs commises”?

```
Selon erreur Dans
    0 => Écrire (“Aucune erreur. Excellent !”)
    1 => Écrire (“Une seule erreur. Très bien.”)
    10 => Écrire (“Tout faux ! Volontaire ?”)
    6..9 => Écrire (“Seulement ”)
        Écrire(10 - erreur)
        Écrire(“bonnes réponses. Il faut apprendre la table de “)
        Écrire (table)
        Écrire(“ !”)
    Autres => Écrire(erreur)
        Écrire (“erreurs. Il faut encore retravailler la table des “)
        Écrire (table)
        Écrire (“.”)
```

FinSelon

R2 : Comment “Afficher un message en fonction du pire temps de réponse” ?

Si pire_tps > (tps_moy + 1) **Alors**
Écrire (“Des hésitations sur la table de “)
Écrire (pire_nb)
Écrire (“ : “)
Écrire (pire_tps)
Écrire (“secondes contre”)
Écrire (tps_moy)
Écrire (“en moyenne. Il faut certainement la réviser.”)
SinonM
Rien
FinSi

R2 : Comment “traiter le choix” ?

Écrire (“On continue (o/n) ?”)
Écrire (choix)
Continue <- (choix = o ou choix = O)

R3 : Comment “poser une multiplication” ?

Appel Get_Random_Number pour initialiser nb nb : **out**
{nb est un nombre aléatoire compris entre 0 et 10}
Écrire (table)
Écrire (“ * “)
Écrire (nb)
Écrire (“ ?”)
Écrire (résultat)
Tps <- Appel mesure temps pour calculer le temps de réponse tps : **out**
{Nous faisons appel aux fonctions auxiliaires données pour calculer le temps que l’utilisateur met pour répondre à la multiplication proposée}
Tps_moy <- tps_moy + tps
Si résultat = (table x nb) **Alors**
Écrire (“Bravo !”)
Sinon
Écrire(“Mauvaise réponse.”)
erreur <- erreur + 1
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 : ...	A		
	Verbe à l'infinitif pour les actions complexes	A		
	Nom ou équivalent pour expressions complexes	A		
	Tous les Ri sont écrits contre la marge et espacés	A		
	Les flots de données sont définis	A		
	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	A		
	Le raffinage d'une action décrit complètement cette action	A		
	Le raffinage d'une action ne décrit que cette action	A		
	Les flots de données sont cohérents	A		
	Pas de structure de contrôle déguisée	A		
	Qualité des actions complexes	P		

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.	
	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.	
	O	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 : Il suffit de changer le nombre de répétitions dans le Pour, on le passe de 10 à 15.

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

Réponse : On rajoute une condition dans le Selon, si la valeur est 2 ou 3, nous affichons “Bien”.

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 : À la place d’une boucle for, on utilise une boucle répéter. On incrémentera un compteur et on ajoutera une condition de sortie si l’utilisateur répond -1 à une multiplication.

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

Réponse : On change la condition qui fait redemander à l’utilisateur une table si elle n’appartient pas à [0,10], on modifie en [0,20].

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 : Il faudrait créer un tableau regroupant le nombre d'erreurs par nombre de droite. Il s’agirait donc d’un tableau à 10 cases. On regarderait ensuite où le plus d’erreurs a été commis et on demande à l’utilisateur s’il souhaite la réviser ou non.

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 : En transformant le Pour par un Répéter, on ajoute un compteur et une condition de sortie dès que le nombre d'erreurs excèdent 4.

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

Réponse : J'ai ajouté une variable auxiliaire pour se souvenir de la multiplication précédente posée et ainsi éviter de la redemander.

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

Réponse : On ne redemande pas à l'utilisateur de choisir une table quelconque à réviser mais on lui propose directement celle avec la plus grande hésitation. La solution serait d'ajouter un si avant "demander la table à réviser", si c'est la première fois que le programme est lancé ou si l'utilisateur ne souhaite pas réviser la table avec la plus grande hésitation, on pose la question sinon les multiplications sont posées sur la table avec la plus grande hésitation.

Difficultés rencontrées

TODO : Indiquer ici les difficultés rencontrées dans la réalisation de ce mini-projet.
(cette partie n’est pas prise en compte dans la notation)

Au moment de calculer le temps moyen, je pensais que je ne pouvais pas diviser par 10 car la variable était de type durée et non flottant ou entier. Il suffisait en réalité de diviser par le flottant 10.0 et la division est possible.

Informations complémentaires

TODO : Indiquer ici ce qui est utile à l'enseignant pour comprendre les raffinages et/ou le programme correspondant. Cette partie peut être vide.

Ayant un mac, j'ai codé avec Mac Vim et le programme compile sans me mettre de message d'erreur mais je n'ai pas testé sur une autre machine. Je ne pense pas qu'il y ait de soucis en passant sur un autre ordinateur mais je préfère vous prévenir au cas où.

Bilan

TODO : Dire quel bilan vous tirez de ce mini-projet Cette partie n'est pas prise en compte dans la notation !

Ce mini-projet m'a permis de bien comprendre le fonctionnement des raffinages et de prendre en main le langage de programmation ADA.