TP: Récursivité

1 Tours de Hanoï

Vous allez programmer la résolution du problème des tours de Hanoï avec l'algorithme décrit en cours. Puis vous allez exprimer en fonction du nombre n de disques initialement sur la tour A le nombre de déplacements de disques pour résoudre ce problème avec cet algorithme.

1.1 L'unité hanoiUtils.pas

L'unité hanoiUtils.pas fournit quelques outils facilitant la programmation des tours de Hanoï. En voici la partie inteface :

interface

```
type
   TOUR = 'A'..'C';

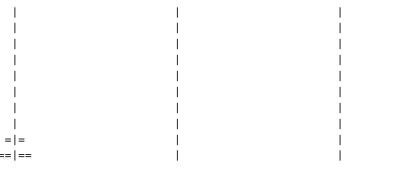
var
   visualisation_tours : BOOLEAN;

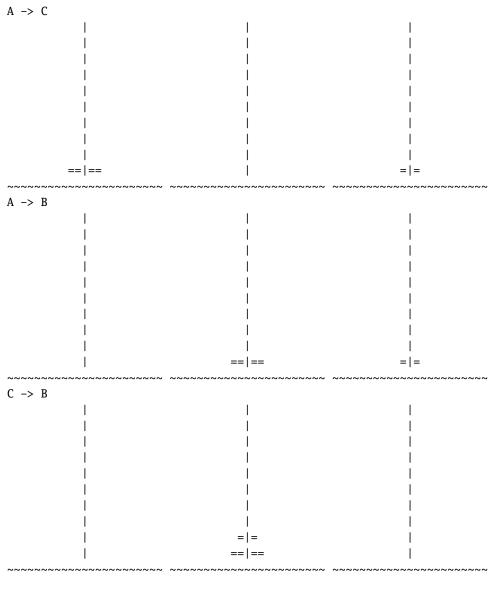
// initialise la tour A avec n disques
// et les autres tas sont vides
// CU : n doit être inférieur ou égal à MAX
procedure initialiserTours(const n : CARDINAL);

// deplace le disque au sommet de la tour d vers la tour a
// CU : la tour d ne doit pas être vide.
procedure deplacerDisque(const d,a : TOUR);
```

- la constante MAX donne le nombre maximal de disques que l'on peut mettre sur la tour A au départ;
- le type TOUR est un type pour décrire les tours possibles : intervalle des trois caractères A, B et C;
- la variable booléenne visualiser_tours : lorsque sa valeur est true, à chaque initialisation et à chaque déplacement de disques les trois tours sont visualisées, et lorsque sa valeur est false, seuls les déplacements sont affichés;
- la procédure **initialiserTours** permet d'initialiser la tour A avec n disques, les deux autres tours étant vides;
- la procédure deplacerDisque déplace le disque au sommet de la tour de départ d vers la tour d'arrivée a, à condition que la contrainte de non vacuité de la tour de départ soit respectée, sinon une exception est déclenchée.

Voici une trace de l'exécution du programme **exemple.pas** lorsque la variable **visualisation_tours** a la valeur **true** :





--- Hanoï - API2 - 2008 ---

et une trace du même programme lorsque cette variable a la valeur ${f false}$:

A -> C

A -> B

C -> B

--- Hanoï - API2 - 2008 ---

Question 1. Récupérez les fichiers de l'unité hanoiUtils.pas et du programme exemple.pas et vérifiez les affichages ci-dessus.

Question 2. Modifiez le programme exemple.pas en donnant la valeur 1 au lieu de 2 à la procédure initialiserTours. Exécutez le programme. Que constatez-vous? Expliquez!

Question 3. Modifiez le programme exemple.pas en donnant la valeur 11 au lieu de 2 à la procédure initialiserTours. Exécutez le programme. Que constatez-vous? Expliquez!

1.2 Programme de résolution

Question 4. En vous appuyant sur la résolution du problème des tours de Hanoï décrite en cours, réalisez un programme nommé hanoi.pas qui résoud le problème des tours de Hanoï pour un nombre de disques qui sera passé en paramètre sur la ligne de commande, de sorte qu'à l'exécution on ait (en supposant que la variable visualisation_tours ait la valeur false):

```
$ hanoi 2
A -> C
A -> B
C -> A
--- Hanoï - API2 - 2008 ---
```

Vous utilisez la fonction **paramstr**(i) qui donne le paramètre numéro i passé sur la ligne de commande, et la fonction **StrToInt** définie dans l'unité SysUtils qui convertit une chaîne de caractères représentant un entier en l'entier correspondant. Par exemple pour obtenir la valeur du premier paramètre passé sur la ligne de commande et la stocker dans une variable n de type entier vous pouvez utiliser l'instruction

```
n := strToInt(paramstr(1));
```

Question 5. Votre programme réalisé, vérifiez son bon fonctionnement pour n compris entre 1 et 10. Avez-vous réellement le courage de vérifier que votre programme respecte les règles du tour de Hanoï lorsque n dépasse 4?

1.3 Combien de déplacements de disques?

Vous allez maintenant chercher une expression du nombre de déplacements de disque en fonction du nombre n de disques initialement sur la tour A pour résoudre le problème des tours de Hanoï avec l'algorithme du cours. Ce nombre de déplacements sera noté h_n .

Question 6. À l'aide de votre programme donnez les valeurs de h_n pour n compris entre 1 et 5.

Question 7. Le travail que vous avez effectué « à la main » dans la question précédente pour trouver h_n devient fastidieux. Nous allons maintenant utiliser la commande Unix we pour calculer h_n avec n comprise entre 6 et 10.

La commande wc (abbréviation pour word count) donne le nombre de lignes, de mots et de caractères du fichier passé en paramètre. Par exemple, utilisée sur le fichier exemple, pas cette commande donne

```
$ wc exemple.pas
13 30 294 exemple.pas
```

qui nous renseigne que ce fichier contient 13 lignes, 30 mots et 294 caractères. Utilisez votre programme hanoi et redirigez la sortie qu'il produit vers un fichier texte.

\$ hanoi 2 > solution-hanoi-2.txt

puis déterminez avec la commande wc le nombre de lignes du fichier solution-hanoi-2.txt.

Quel rapport y a-t-il entre le nombre de lignes du fichier et le nombre de déplacements?

Vérifiez pour d'autres valeurs de n inférieures à 5.

Question 8. Déterminez maintenant les valeurs de h_n pour n compris entre 6 et 10.

Question 9. Induisez des valeurs trouvées pour h_n , $1 \le n \le 10$, une formule exprimant h_n en focation de n.

Question 10. Démontrez la formule que vous avez trouvée.

Question 11. (question difficile) Existe-t-il un autre algorithme de résolution plus rapide, c'est-à-dire faisant moins de déplacements de disque?

2 Fibonacci

La suite des nombres de Fibonacci est une suite de nombres entiers naturels que l'on définit en donnant ses deux premiers termes

$$F_0 = 0 (1)$$

$$F_1 = 1 (2)$$

et en définissant les termes d'indice $n \ge 2$ par la relation de récurrence

$$F_n = F_{n-1} + F_{n-2}. (3)$$

Vous allez programmer le calcul des termes de cette suite de deux façons différentes. Question 12. Calculez « à la main » les termes de cette suite pour n compris entre 0 et 10.

2.1 Programmation récursive

Question 13. Réalisez une fonction nommée fiborec qui prend en paramètre un indice n, et donne en réponse le nombre F_n . Vous programmerez cette fonction de manière récursive en suivant simplement la définition de la suite donnée par les équations ?? et ??.

Question 14. Vérifiez avec votre programme les valeurs de la suite jusqu'à l'indice 10.

Question 15. Utilisez votre programme pour calculer F_{20} , F_{30} et F_{40} . Quelle remarque pouvez-vous formuler sur l'exécution de votre programme?

2.2 Programmation itérative

Question 16. Réalisez une fonction nommée fiboiter qui prend en paramètre un indice n, et donne en réponse le nombre F_n . Vous programmerez cette fonction de manière itérative en mémorisant à l'aide de variables locales les deux dernièrs termes calculés.

Question 17. Vérifiez avec votre programme les valeurs de la suite jusqu'à l'indice 10.

Question 18. Utilisez votre programme pour calculer F_{20} , F_{30} et F_{40} .

Question 19. Calculez F_{46} , F_{47} et F_{48} . Que remarquez-vous pour ce dernier terme? En serait-il de même avec la version récursive? Expliquez!

2.3 Comparaison des deux versions

Question 20. Comparez les deux versions en terme de simplicité de programmation et d'efficacité.

2.4 Étude de la complexité

Vous allez étudiez le nombre d'additions de deux entiers effectuées dans le calcul de F_n par l'algorithme récursif (fonction **fiborec**). On note a_n ce nombre.

Question 21. Que valent a_0 et a_1 ?

Question 22. Justifiez que pour tout entier $n \geq 2$, on a

$$a_n = a_{n-1} + a_{n-2} + 1. (4)$$

Question 23. Programmez une fonction qui calcule a_n , l'entier n étant passé en paramètre. Allez-vous choisir une méthode récursive ou itérative de programmation. Justifiez votre choix.

Question 24. À l'aide de la fonction ainsi programmée, calculez a_n pour n compris entre 0 et 10.

Question 25. Calculez a_{40} . Puis concluez sur le calcul de F_{40} par l'algorithme récursif.

On note b_n le nombre d'additions de deux entiers effectuées dans le calcul de F_n par l'algorithme itératif (fonction fiboiter).

Question 26. Exprimez b_n en fonction de n.

Question 27. Comparez a_{40} et b_{40} . Concluez sur la comparaison des deux algorithmes récursif et itératif.