Algorithmique et Base de la programmation

Durée : *3h00*
Documents autorisés : **AUCUN**

Remarques :
- Veuillez lire attentivement les questions avant de répondre.
- Le barème donné est un barème indicatif qui pourra évoluer lors de la correction.
- Rendez une copie propre.

1 Liste chaînée (6 points)

1. Rappelez le TAD ListeChainee ;
2. Donnez les signatures des fonctions et procédures permettant d'utiliser ce type ;
3. Donnez l'algorithme récursif d'une fonction qui permet de calculer le nombre d'éléments d'une liste chaînée. Votre algorithme est-il récursif terminal ou non-terminal ? Justifiez ;
4. Donnez l'algorithme d'une procédure qui permet de concaténer une liste chaînée à une autre ;
5. Donnez l'algorithme d'une procédure qui permet de supprimer toutes les occurrences d'un élément.

Solution proposée :

1. Voir Cours
2. Voir Cours
3. Voir TD
4. Voir TD
5.

```
procédure supprimer (E/S l : ListeChainee ; E e : Element)
   Déclaration   temp : ListeChainee
debut
   si non estVide(l) alors
      temp ← obtenirListeSuivante(l)
      supprimer(temp,e)
      si obtenirElement(l)=e alors
         l ← temp
      sinon
         fixerListeSuivante(l,temp)
      finsi
   finsi
fin
```

2 The *Chuck Norris of numbers* (3 points)

Dans l'un des épisodes de la série « *The Big Bang Theory* », Sheldon Cooper demande à ses amis « *What is the best number ?* ». Ses amis ne trouvant pas la bonne réponse, il leur dit que :

« *The best number is 73. Why ? 73 is the 21st prime number. Its mirror (37) is the 12th and its mirror (21) is the product of multiplying, hang on to your hats, 7 and 3. ... In binary, 73 is a palindrome, 1001001 which backwards is 1001001.* »

Proposez une analyse descendante de l'opération qui permet de savoir si un nombre naturel possède toutes ces caractéristiques.

Solution proposée :

bestNumber **NaturelNonNul** → **Booleen**
- estPremier **NaturelNonNul** → **Booleen**
- rangNbPremier **NaturelNonNul** ⇸ **NaturelNonNul**
- mirroir **Naturel** → **Naturel**
  - representationDecimal **Naturel** → **Chaine de caracteres**
    - representationNaire **Naturel** → **Chaine de caracteres**
  - inverserChaine **Chaine de caracteres** → **Chaine de caracteres**
  - chaineEnNaturel **Chaine de caracteres** × 2..16 ⇸ **Naturel**
    - chiffreEnNaturel **Caractere** × 2..16 ⇸ **Naturel**
- representationBinaire **Naturel** → **Chaine de caracteres**
  - representationNaire **Naturel** → **Chaine de caracteres**
- estPalindrome **Chaine de caracteres** → **Booleen**

3 Des étoiles (4 points)

Supposons que la procédure suivante permette de dessiner un segment sur un graphique (variable de type `Graphique`) :

- **procédure** ligne **(E/S** g : Graphique,**E** x1,y1,x2,y2 : **Reel)**

Supposons que l'on possède également les fonctions mathématiques suivantes :

- **fonction** cos (angleEnRadian : **Reel) : Reel**
- **fonction** sin (angleEnRadian : **Reel) : Reel**
- **procédure** translater **(E** x,y,xVecteur,yVecteur : **Reel**, **S** xResultat,yResultat : **Reel)**
- **procédure** faireRotation **(E** x,y,xCentre,yCentre,angleEnRadian : **Reel**, **S** xResultat,yResultat : **Reel)**

L'objectif est de concevoir une procédure `dessinerEtoiles` qui permet de dessiner sur un graphique des étoiles à cinq branches [1] telles que présentées par la figure 1.

La signature de cette procédure est :

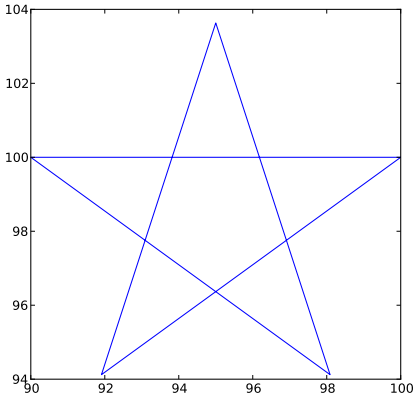
- **procédure** dessinerEtoiles **(E/S** g : Graphique, **E** xDepart,yDepart : **Reel**, niveauRecursion : **Naturel**, taille,direction : **Reel)**

tel que :

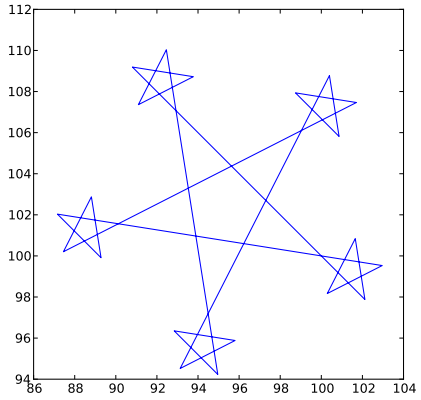
- *g* est le graphique sur lequel on va faire le dessin ;
- *xDepart* et *yDepart* les coordonnées du premier sommet d'une étoile tel que défini par la figure 2 ;
- *taille* est la distance entre deux sommets successifs (lors d'un appel récursif, la taille est divisée par 3) ;
- *direction* est l'angle initial entre le premier et le deuxième sommet de l'étoile.

Sachant que l'angle que forme trois points consécutifs d'une étoile est de $-\pi/5$, sachant que l'on possède la constante `PI`, proposez l'algorithme de la procédure `dessinerEtoiles`.

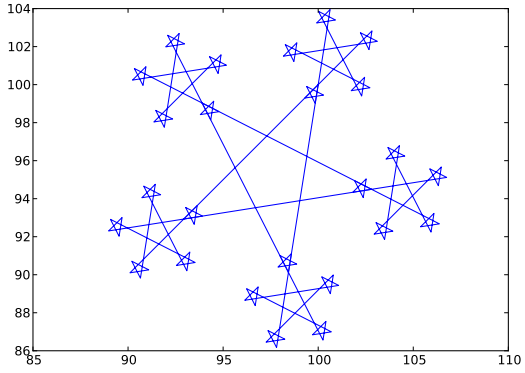
1. Pour des étoiles à cinq branches, l'angle non orienté entre les deux arrêtes d'un sommet est de $\pi/5$



(a) $dessinerEtoiles(g, 100, 100, 1, 10, \pi)$



(b) $dessinerEtoiles(g, 100, 100, 2, 10, 3 * \pi/4)$



(c) $dessinerEtoiles(g, 100, 100, 3, 10, 5 * \pi/4)$

FIGURE 1 – Des exemples de graphiques

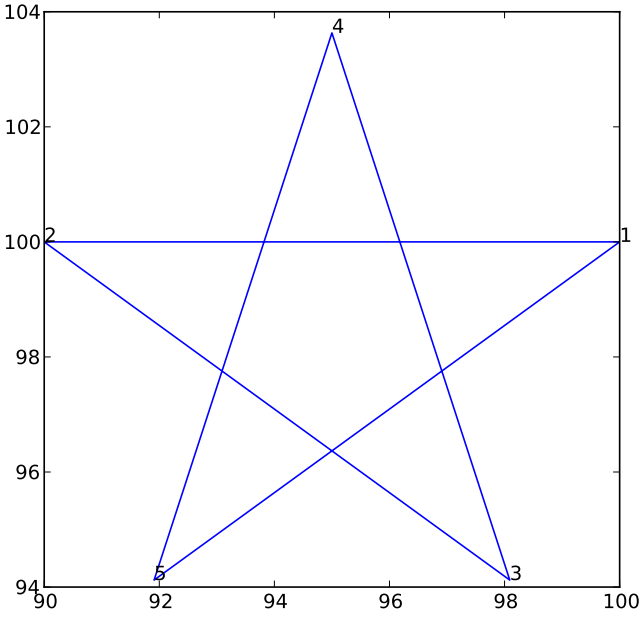


FIGURE 2 – Ordre de dessin des points d'une étoile (dessinée par $dessinerEtoiles(g, 100, 100, 1, 10, \pi)$)

Solution proposée :

procédure dessinerEtoiles (**E/S** g : Graphique, **E** xDepart,yDepart : **Reel**, niveauRecursion : **Naturel**, taille,direction : **Reel**)
debut
  si niveauRecursion>0 **alors**
    x1 ← xDepart
    y1 ← yDepart
    translation(x1,y1,taille*cos(direction),taille*sin(direction),x2,y2)
    ligne(g,x1,y1,x2,y2)
    direction ← direction-PI/5
    dessinerEtoiles(g,x2,y2,niveauRecursion-1,taille/3,direction)
    pour i ←1 **à** 4 **faire**
      direction ← direction+4*PI/5
      faireRotation(x1,y1,x2,y2,-PI/5,x3,y3)
      ligne(g,x2,y2,x3,y3)
      dessinerEtoiles(x3,y3,niveauRecursion-1,taille/3),direction
      x1 ← x2
      y1 ← y2
      x2 ← x3
      y2 ← y3
    finpour
  finsi
fin

4 AVL (7 points)

Pour rappel un AVL est un arbre binaire de recherche qui est toujours équilibré (la différence de hauteur entre le sous arbre gauche et le sous arbre droit est d'au plus 1). On définit donc le type AVL :

- **Type** AVL = ArbreBinaire

4.1 Question de cours (3 points)

1. Rappelez le TAD ArbreBinaire ainsi que la signature des fonctions et procédures permettant de manipuler ce type de données.
2. Rappelez à l'aide de dessins quels sont les objectifs des opérations de simple et double rotation vues en cours (en choisissant celles à gauche ou celles à droite).

Solution proposée :

1. Voir cours
2. Voir TD

4.2 L'insertion (4 points)

Pour rappel l'algorithme d'insertion d'un élément dans un AVL [2] est :
procédure insérer (**E/S** a : AVL, **E** e : Element)

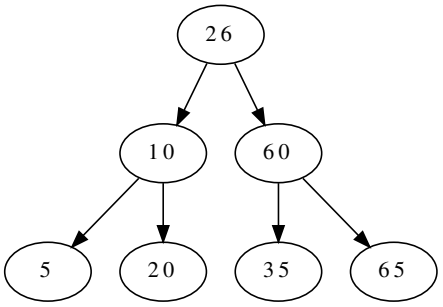
Déclaration temp : ABR

2. Le type `Element` possède un ordre total

```
debut
   si estVide(a) alors
      a ← ajouterRacine(arbreBinaireRecherche(),arbreBinaireRecherche(),e)
   sinon
      si e≤obtenirElementRacine(a) alors
         temp ← obtenirFilsGauche(a)
         insérer(temp,e)
         fixerFilsGauche(a,temp)
         equilibrer(a)
      sinon
         temp ← obtenirFilsDroit(a)
         insérer(temp,e)
         fixerFilsDroit(a,temp)
         equilibrer(a)
      finsi
   finsi
fin
```

1. Dessinez les arbres, en partant d'un arbre initialement vide, issus de l'insertion successive des valeurs : $26, 60, 35, 10, 5, 20, 65$.
2. Donnez l'algorithme de la procédure *equilibrer*.

Solution proposée :

1. 

2. Voir TD