DS2 - Algorithmes et Structures de Données
Mardi 13 Janvier 2015

Durée 3H – Documents non autorisés

1. Pile - Tri rapide (quicksort) - (4 pts)

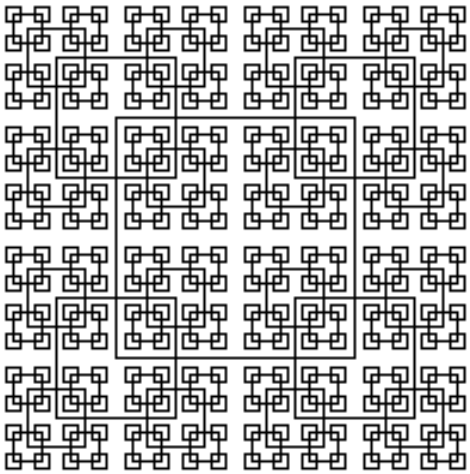
t: 4 10 8 18 12 2 16 0 14 6

Simuler la **pile** sur l'appel `tri-rapide(t,1,10)` {@0} en donnant aussi les valeurs intermédiaires de t après l'appel de `partionner`.

```
Const max = 100
Type tab = tableau [1...max] d'entier
Procédure tri-rapide(E/S t : tab, E inf,sup : entier)
Var p : entier
Début
  si inf<sup
     Alors partionner(t,inf,sup,p)
           tri-rapide(t,inf,p-1){@1}
           tri-rapide(t,p+1,sup){@2}
  Finsi
Fin
Procédure partitionner (E/S t : tab, E d,f : entier, S p : entier)
Var i,j,pivot : entier
Début
   pivot←t[d]
   i←d
   j←f
   TantQue i≤j Faire
      TantQue t[i]≤pivot et i≤j Faire
         i←i+1
      FinTantQue
      TantQue t[j]>pivot et i≤j Faire
         j←j-1
      FinTantQue
      Si i≤j alors échanger(t[i],t[j])
      FinSi
   FinTantQue
   p←j
   échanger(t[d],t[j])
Fin
```

2. Dessin récursif - (4 pts)

On souhaite dessiner la figure dont on a mis ci-dessous les premières étapes.



On dispose de la procédure `dessiner-carré(x,y,c : entier)` qui dessine le carré de centre `x,y` et de demi-côté `c`. Ecrire en pseudo-langage la procédure **récursive** `carré(x, y, c : entier)` qui dessine la figure entière (`c`>2).

3. Polynômes - (4 pts)

Ecrire en peudo-langage une fonction qui calcule la dérivée d'un polynôme.
`Fonction calculer-derivee (p : poly) : poly`
On s'intéressera aux deux cas suivants :
3.1. Le type `poly` est représenté par un tableau de réels où chaque élément est un terme du polynôme : l'indice (un entier) représente le degré du terme et la valeur (un réel) représente le coefficient.
```
Type poly = tableau [0...max] de réel
```
3.2. Le type est représenté par une liste chainée de termes où chaque terme est enregistrement contenant le degré (un entier), le coefficient (un réel) et un pointeur sur le terme suivant. La liste est triée dans l'ordre croissant des degrés.
```
Type terme = Enregistrement
                d : entier
                c : réel
                suiv : poly
             FinEnregistrement
     poly = ^terme
```

4. Jeu de UNO (Règles simplifiées) - (8 pts)

Le but du jeu est de se débarrasser de ses cartes (initialement sept par joueur) en jouant l'un après l'autre. Les 108 cartes sont caractérisées par une valeur (chiffre de 0 à 9 plus deux valeurs particulières : « sauter le tour », qui fait passer le tour du joueur suivant et « changer de sens », qui change le sens du jeu) et une couleur (rouge, vert, bleu ou jaune). A chaque tour, le joueur peut se séparer d'une carte uniquement si elle possède la même couleur **ou** la même valeur que la précédente. Un joueur ne pouvant jouer doit piocher une carte : si celle-ci est « jouable », elle peut alors être jouée immédiatement. Les joueurs qui terminent sont retirés du jeu au fur et à mesure et leur nom est affiché. La partie est terminée quand il ne reste plus qu'un seul joueur. On utilisera les structures de données suivantes :
```
Type carte = Enregistrement
               val : chaîne
               coul : chaine
               cartesuiv : ^carte
             FinEnregistrement
     joueur = Enregistrement
                nom : chaîne        {nom du joueur}
                lcarte : ^carte     {liste des cartes du joueur}
                jsuiv, jprec : ^joueur
              FinEnregistrement
     liste-joueur = ^joueur
     sens-suiv = booléen       {=vrai si on suit le sens jsuiv, =faux si on suit jprec}
     defausse = pile de carte  {pile de cartes jetées}
     pioche = pile de carte    {pile de cartes à piocher}
```

Attention liste-joueur est une liste doublement chaînée circulaire où chaque joueur pointe sur la liste de ses cartes ! Faites un dessin pour illustrer les algorithmes demandés.

4.1. Ecrire en peudo-langage une procédure qui ajoute la carte `c` dans le jeu du joueur pointé par `pj`.
`Procédure Ajouter-Carte (E c : carte, E/S pj : ^joueur)`
4.2. Ecrire en peudo-langage une procédure qui supprime la `i`ème carte du joueur pointé par `pj`.
`Procédure Supprimer-Carte (E i : entier, E/S pj : ^joueur)`
4.3. Ecrire en peudo-langage une procédure qui supprime le joueur pointé par `pj`.
`Procédure Supprimer-Joueur (E pj : ^joueur, S lj : liste-joueur)`
4.4. Ecrire en peudo-langage une procédure qui distribue les 7 cartes de départ pour chacun des `n` joueurs (dont on demandera les noms) en créant la liste `lj`. Les cartes sont prises dans `pioche` contenant initialement les 108 cartes mélangées. A chaque fois qu'une carte est distribuée, la procédure `Ajouter-Carte` est appelée.
`Procédure Distribuer (E n : entier, E/S p : pioche, S lj : liste-joueur)`
4.5. Ecrire en peudo-langage une fonction qui renvoie la position de la 1ère carte possible à jouer dans la liste de carte du joueur pointé par `pj` en fonction de la dernière carte jetée (sur la défausse) ; 0 si aucune carte n'est possible.
`Fonction Choisir-Carte (pj : ^joueur, d : defausse) : entier`
4.6. Bonus – Ecrire le programme en peudo-langage qui permet de jouer une partie complète.