Algorithmique & Programmation Orientée Objet
Semestre 2 ST
La récursivité

Tableaux de variables | Algorithmes de recherche et de tri | Précision, rapidité et complexité
Matrices de variables | Récursivité | Informations et Archives
Travaux dirigés et travaux pratiques | Evaluation intermédiaire | Sujets de projet

Cours | TD - Corrections | TP

Exercice n°1: Calcul de factoriel

La définition récurrente du calcul de n! est:
- 0! = 1
- n! = n * (n-1)!

Implanter un sous-algorithme de calcul de n! en utilisant la récursivité.

FactorielRecursif.lda

```
{ Fonction de calcul et retour                       }
{ de n! par methode recursive                        }
{ Définition:                                        }
{  - 0! = 1                                          }
{  - n! = (n-1)!*n                                   }
{ n : La valeur pour laquelle n! est calculé         }

entier fonction factoriel(n)
  Entrées
    entier n
  Locales
    entier res
  si n == 0 alors
    res <- 1
    sinon
    res <- factoriel(n-1)*n
  fsi
  retourner res
fin fonction
```

Clavier.class - Ecran.class - Exemple d'exécution

Exercice n°2: Inversion d'une chaîne de caractères par décomposition dichotomique

Utiliser la technique de décomposition dichotomique pour implanter un sous-algorithme récursif d'inversion d'une chaîne de caractères:
 - Une chaîne de 0 ou 1 caractère est déjà inversée.
 - Une chaîne s de n caractères où n est supérieur à 1 peut être inversée en concaténant s2 et s1 où s1 est la chaîne obtenue par inversion de la 1/2 sous-chaîne comportant les n/2 premiers caractères de s et s2 est la chaîne obtenue

par inversion de la 1/2 sous-chaîne comportant les n-n/2 derniers caractères de s.

On pourra utiliser la fonction suivante pour extraire une sous-chaîne de caractères d'une chaîne de caractères:

```
chaîne fonction sousChaine(s,indi,indf)
  Données
    s : chaîne
    indi : entier
    indf : entier
```

où s est la chaîne où l'extraction est réalisée, indi est l'indice du caractère de s à partir duquel l'extraction est réalisée et indf est l'indice du caractère qui suit immédiatement le dernier caractère extrait de s (i.e. le nombre de caractères extraits est égal à indf-indi).

InversionChaineMethodeDichotomique.lda

```
{ Fonction de calcul et retour de l'inverse   }
{ d'une chaine de caracteres                  }
{ st : la chaine de carateres à inverser      }

chaine fonction inversionChaine(st)
  Entrées
    chaine st
  Locales
    chaine s
    chaine s1
    chaine s2
    entier l1
    entier lst <- longueur(st)
  si ( lst == 0 ) ou ( lst == 1 ) alors
    s <- st
```

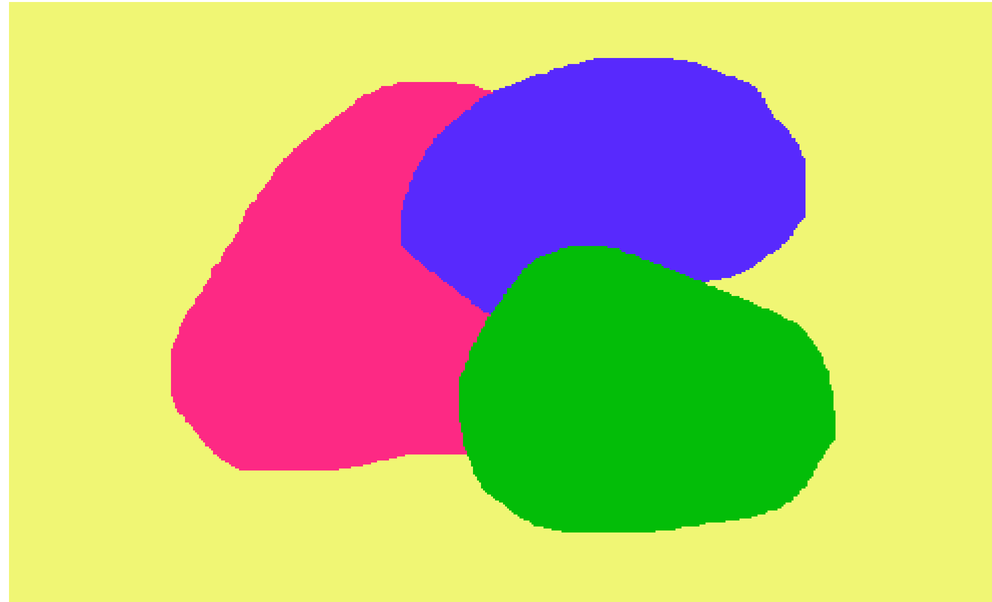
```
    sinon
    l1 <- lst/2
    s1 <- inversionChaine(sousChaine(st,0,l1))
    s2 <- inversionChaine(sousChaine(st,l1,lst))
    s <- s2+s1
  fsi
  retourner s
fin fonction
```

Clavier.class - Ecran.class - Exemple d'exécution

Exercice n°3: "Coloration" dans un tableau d'entiers

On considère un tableau d'entiers de taille NxM. Ce tableau code une image où chacun des entiers code la couleur d'un pixel.
On souhaite colorier des zones de pixels.

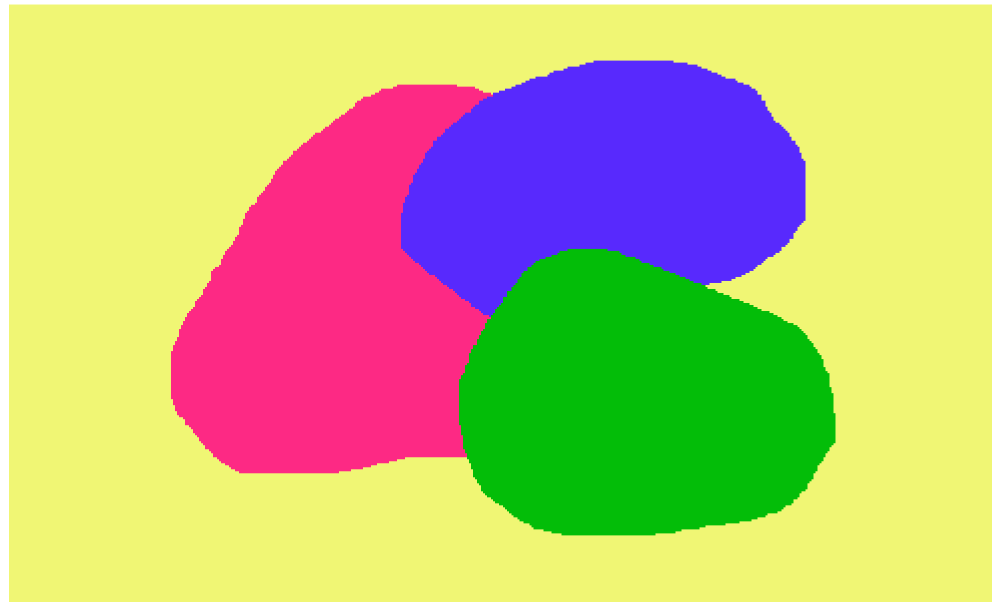
a) Développer un sous-algorithme de coloriage, au moyen d'une couleur, de la zone de pixels définie par les règles suivantes:
(1) Un premier pixel de coordonnées (x,y) est colorié s'il a une couleur différente de la couleur de tracé. Si ce n'est pas le cas le coloriage s'arrête.
(2) Un pixel de couleur identique à la couleur du premier pixel colorié et non encore colorié touche (par la gauche, par la droite, par le haut ou par le bas) un pixel qui a été colorié.
(3) Tant qu'il existe des pixels vérifiant la règle (2), on les colorie avec la couleur de tracé.
Ce sous-algorithme permet de "remplir" une tache de couleur uniforme uniformément avec une autre couleur.



Exemples
Appeler le sous-algorithme sur un pixel bleu aura pour conséquence
de remplir entièrement la tache bleue avec la couleur de tracé.
Appeler le sous-algorithme sur un pixel rose aura pour conséquence
de remplir entièrement la tache rose avec la couleur de tracé.
Appeler le sous-algorithme sur un pixel vert aura pour conséquence
de remplir entièrement la tache verte avec la couleur de tracé.
Appeler le sous-algorithme sur un pixel jaune aura pour conséquence
de remplir entièrement (jusqu'au bord) la tache jaune avec la couleur de tracé.
Appeler le sous-algorithme sur un pixel ayant la même couleur
que la couleur de tracé n'entrainera pas de remplissage.

b) Développer un sous-algorithme de coloriage, au moyen d'une couleur, de la zone de pixels définie par les règles suivantes:

(1) Un premier pixel de coordonnées (x,y) est colorié s'il a une couleur différente d'une couleur limite. Si ce n'est pas le cas le coloriage s'arrête.
(2) Un pixel de couleur différente de la couleur limite et non encore colorié touche (par la gauche, par la droite, par le haut ou par le bas) un pixel qui a été colorié.
(3) Tant qu'il existe des pixels vérifiant la règle (2), on les colorie avec la couleur de tracé.
Ce sous-algorithme permet de "remplir" uniformément avec une couleur une tache de couleur non-uniforme délimitée par une couleur uniforme.



Exemples
Appeler le sous-algorithme sur un pixel bleu, rose ou vert
avec comme couleur limite la couleur jaune aura pour conséquence
de remplir entièrement les taches bleue, rose et verte avec la couleur de tracé.
Si la couleur limite est le rose, désigner un pixel jaune, bleu ou vert
aura pour conséquence le remplissage des taches bleue et verte

ainsi que le remplissage de la tache jaune jusqu'au bord.
Désigner un pixel ayant la même couleur
que la couleur limite n'entrainera pas de remplissage.

ColoriageRecursif.lda

```
{ Action recursive de coloriage d'une zone     }
{ de pixels de valeurs identiques              }
{ px : La position en x du pixel germe         }
{ py : la position en y du pixel germe         }
{ t  : la matrice des valeurs des pixels       }
{ c  : la couleur de remplissage               }
{ ct : la couleur de la tache à remplir        }

action coloriageRecursif(px,py,t,c,ct)
  Entrées
    entier px
    entier py
    entier c
    entier ct
  Entrées / Sorties
    Tableau [][] de entier t
  Locales
    entier n <- longueur(1,t)
    entier m <- longueur(2,t)
  si ( px >= 0 ) et ( px < m ) et ( py >= 0 ) et ( py < n ) alors
    si ( t[py][px] <> c ) et ( t[py][px] == ct ) alors
      t[py][px] <- c
      coloriageRecursif(px+1,py,t,c,ct)
```

```
      coloriageRecursif(px-1,py,t,c,ct)
      coloriageRecursif(px,py+1,t,c,ct)
      coloriageRecursif(px,py-1,t,c,ct)
    fsi
  fsi
fin action

{ Action de coloriage d'une zone de pixels    }
{ de valeurs identiques                       }
{ px : La position en x du pixel germe        }
{ py : la position en y du pixel germe        }
{ t  : la matrice des valeurs des pixels      }
{ c  : la couleur de remplissage              }

action coloriage(px,py,t,c)
  Entrées
    entier px
    entier py
    entier c
  Entrées / Sorties
    Tableau [][] de entier t
  coloriageRecursif(px,py,t,c,t[py][px])
fin action

{ Action recursive de remplissage d'une zone  }
{ de pixels delimitee par une valeur          }
{ px : La position en x du pixel germe        }
{ py : la position en y du pixel germe        }
{ t  : la matrice des valeurs des pixels      }
{ c  : la couleur de remplissage              }
```

```
{ cl : la couleur de delimitation                }
{      de la tache à remplir                     }

action remplissageRecursif(px,py,t,c,cl)
  Entrées
    entier px
    entier py
    entier c
    entier cl
  Entrées / Sorties
    Tableau [][] de entier t
  Locales
    entier n <- longueur(1,t)
    entier m <- longueur(2,t)
  si ( px >= 0 ) et ( px < m ) et ( py >= 0 ) et ( py < n ) alors
    si ( t[py][px] <> c ) et ( t[py][px] <> cl ) alors
      t[py][px] <- c
      remplissageRecursif(px+1,py,t,c,cl)
      remplissageRecursif(px-1,py,t,c,cl)
      remplissageRecursif(px,py+1,t,c,cl)
      remplissageRecursif(px,py-1,t,c,cl)
    fsi
  fsi
fin action
```

Clavier.class - Ecran.class - Exemple d'exécution

Exercice n°4: Calcul de combinaisons

On considère n caractères. Développer un sous-algorithme récursif d'affichage de toutes les combinaisons de ces n

caractères.

AffichageCombinaisons.lda

```
{ Affichage de toutes les combinaisons       }
{ de n valeurs existant un ensemble          }
{ de n valeurs                               }
{ Application a un tableau de caracteres     }

action affichage(e,t,nb)
  Entrées
    Tableau[] de caractere e
    Tableau[] de caractere t
    nb entier
  Locales
    entier i
    caractere c
  si nb == longueur(t) alors
    pour i de 0 à longueur(t)-1 faire
      afficher(t[i])
    fait
    afficherln()
    sinon
    pour i de 0 à longueur(t)-nb-1 faire
      t[nb] <- e[i]
      c <- e[i]
      e[i] <- e[longueur(t)-nb-1]
      affichage(e,t,nb+1)
      e[longueur(t)-nb-1] <- e[i]
      e[i] <- c
```

```
    fait
  fsi
fin action
```

Clavier.class - Ecran.class - Exemple d'exécution

Exercice n°5: Affichages par récursivité

Précision: Pour les questions suivantes, on ne dispose ni du "pour" ni du "tant que".

a) On considère un nombre entier n positif ou nul. Développer un sous-algorithme permettant d'afficher en ordre décroissant la liste des nombres entiers compris dans l'intervalle [0,n].

b) On considère un nombre entier n positif ou nul. Développer un sous-algorithme permettant d'afficher en ordre croissant la liste des nombres entiers compris dans l'intervalle [0,n].

c) On considère un nombre entier n positif ou nul. Développer un sous-algorithme permettant d'afficher en ordre décroissant puis par ordre croissant la liste des nombres entiers compris dans l'intervalle [0,n].

d) On considère un nombre entier n positif ou nul. Développer un sous-algorithme permettant d'afficher en ordre croissant puis par ordre décroissant la liste des nombres entiers compris dans l'intervalle [0,n].

e) On considère un nombre entier n positif ou nul. Développer un sous-algorithme permettant d'afficher n par affichage individuel de ses chiffres.

La question (b) a été résolue en 2 variantes.
La première s'inspire directement de la solution de la question (a) où les affichages sont réalisés avant les relancent récursives. Dans ce cadre, (1) la connaisance de n à tous les niveaux de récursivité étant nécessaire, la valeur de n

est transmise en paramètre d'entête supplémentaire, (2) deux actions sont nécessaires, une première récursive à 2 paramètres, une seconde à 1 paramètre pour lancer la première.
La seconde est meilleure dans le sens où les affichages étant réalisés après les relancent résursives, il n'est plus nécessaire de gérer n en paramètre supplémentaire. Il devient donc possible de ne développer qu'une action.

AffichagesRecursifs.lda

```
{ Action d'affichage par ordre decroissant    }
{ des nombres entiers compris entre 0 et n    }
{ n : nombre limite pour l'affichage          }

action affichageDecroissant(n)
  Entrées
    n entier
  afficherln(n)
  si n > 0 alors
    affichageDecroissant(n-1)
  fsi
fin action

{ Action d'affichage par ordre croissant      }
{ des nombres entiers compris entre v et n    }
{ v : valeur minimale                         }
{ n : nombre limite pour l'affichage          }

action affichageCroissant(v,n)
  Entrées
    entier v
    entier n
```

```
  affichern(v)
  si v < n alors
    affichageCroissant(v+1,n)
  fsi
fin action

{ Action d'affichage par ordre croissant      }
{ des nombres entiers compris entre 0 et n    }
{ n : nombre limite pour l'affichage          }

action affichageCroissant(n)
  Entrées
    entier n
  affichageCroissant(0,n)
fin action

{ Action d'affichage par ordre décroissant    }
{ puis croissant des nombres entiers          }
{ compris entre 0 et n                        }
{ n : nombre limite pour l'affichage          }

action affichageDecroissantCroissant(n)
  Entrées
    entier n
  afficherln(n)
  si n > 0 alors
    affichageDecroissantCroissant(n-1)
  fsi
  afficherln(n)
fin action
```

```
{ Action d'affichage par ordre croissant      }
{ puis decroissant des nombres entiers        }
{ compris entre v et n                        }
{ v : valeur minimale                         }
{ n : nombre limite pour l'affichage          }

action affichageCroissantDecroissant(v,n)
  Entrées
    entier n
  afficherln(v)
  si v < n alors
    affichageCroissantDecroissant(v+1,n)
  fsi
  afficherln(v)
fin action

{ Action d'affichage par ordre croissant      }
{ puis decroissant des nombres entiers        }
{ compris entre 0 et n                        }
{ n : nombre limite pour l'affichage          }

action affichageCroissantDecroissant(n)
  Entrées
    entier n
  affichageCroissantDecroissant(0,n)
fin action

{ Action d'affichage par ordre croissant      }
{ des nombres entiers compris entre 0 et n    }
```

```
{ Solution "elegante"                              }
{ n : nombre limite pour l'affichage               }

action affichageCroissant2(n)
  Entrées
    entier n
  si n > 0 alors
    affichageCroissant2(n-1)
  fsi
  afficherln(n)
fin action

{ Action d'affichage des chiffres                  }
{ d'un nombre entier                               }
{ n : nombre entier a afficher                     }

action affichage(n)
  Entrées
    entier n
  si n >= 10 alors
    affichage(n/10)
  fsi
  afficher(n%10)
fin action
```

Clavier.class - Ecran.class - Exemple d'exécution

Auteur: Nicolas JANEY
UFR Sciences et Techniques
Université de Besançon
16 Route de Gray, 25030 Besançon

nicolas.janey@univ-fcomte.fr