Licence de Mathématiques et Informatique — Université de Perpignan Via Domitia
Semestre 2

Algorithmique

Correction de l'examen de seconde session 2014/2015

Durée : 2 heures. Aucun document autorisé.

Modalités : Répondre uniquement dans les cadres prévus à cet effet. Le barême est indiqué à droite de chaque question. La qualité de la rédaction sera prise en compte dans la notation.

Exercice 1. (22 points) **/22**

1. Écrire le corps de la procédure suivante qui permute (échange) les valeurs de `a` et `b` :
   `fonction swap(a : in out entier, b : in out entier)` **/1**

```
fonction swap(a : in out entier, b : in out entier)
  t : entier // variable locale de tranfert
debut
  t = a
  a  = b
  b = a
fin fonction
```

2. Écrire un algorithme qui utilise `swap` pour permuter puis afficher deux valeurs entrées au clavier.
   /1

```
declare
  v1, v2 : entiers
debut
  lire(v1)
  lire(v2)
  swap(v1, v2)
  afficher(v1)
  afficher(v2)
fin
```

On souhaite inverser l'ordre des valeurs d'un tableau donné (taille et valeurs connues) : le tableau $[a_0, a_1, ..., a_{n-1}]$ sera renversé en $[a_{n-1}, ..., a_1, a_0]$.

3. Donner le principe d'un algorithme qui réalise cette inversion en utilisant la procédure `swap`. Bien différencier les cas selon la parité de la longueur n du tableau pour identifier le parcours à effectuer. **/2**

```
On commence par permuter le couple (première valeur, dernière valeur),
soit (t[0], t[n-1]).
Puis on fait de même pour les valeurs en position 2 et n-2 :  (t[1],
t[n-2]).
Et on répète ces permutations jusqu'à atteindre la moitié du tableau.
La moitié dépend de la parité de la longueur n du tableau.
Si n est pair et s'écrit n=2p, alors le dernier échange s'effectue
entre t[p-1] et t[p].
Sinon n=2p+1. t[p] ne bouge pas et t[p-1] et t[p+1] sont les dernières
valeurs permutées.
Le parcours de la moitié de t s'effectue donc de t[0] à t[p-1] où p
est le quotient de n sur 2 (division euclidienne).
```

4. Écrire un algorithme qui réalise cette inversion de l'ordre des valeurs d'un tableau donné. Le tableau sera par exemple de taille n=10 initialisé à votre goût. **/3**

```
declare
  n : entier = 10   // taille de t
  t[n] : tableau d'entiers = [1,2,3,4,5,6,7,8,9,10]
  i : entier : itérateur pour
debut
  pour i de 0 à p-1 faire
    swap(t[i], t[n-1-i])
  fin pour
fin
```

5. Écrire l'en-tête d'un sous-programme (fonction ou procédure) `renverse` qui effectue le traitement précédent sur un tableau passé en paramètre. **/1**

```
renverse est une procédure qui modifie le tableau t.
fonction renverse(n : entier, t[n] : in out tableau d'entier)
```

6. Écrire un algorithme qui utilise `renverse` pour inverser l'ordre des valeurs d'un tableau entrées au clavier. Le tableau inversé sera affiché.

/2

```
declare
  n : entier = 10   // taille de t
  t[n] : tableau d'entiers
  i : entier        // itérateur pour
debut
  pour i de 0 à n faire  // entrées
    lire(t[i])
  fin pour
  renverser(n, t)        // traitement
  pour i de 0 à n faire  // sorties
    afficher(t[i])
  fin pour
fin
```

7. Écrire le corps du sous-programme `renverse`. **/1**

```
fonction renverse(n : entier, t[n] : in out tableau d'entier)
  i : entier //itérateur pour
debut
  pour i de 0 à p-1 faire
    swap(t[i], t[n-1-i])
  fin pour
fin
```

8. Proposer un paramètre de complexité pour `renverse` ainsi que l'expression de sa fonction de complexité. **/2**

```
renverse effectue des appels à swap dont le nombre dépend de la taille
n du tableau.
swap effectue 3 affectations uniquement.
Le paramètre de complexité est donc le nombre d'affectations.
La fonction de complexité C(n) = 3n.
```

9. Caractériser la complexité asymptotique de `renverse`. Donner un eexmple d'algorithme de complexité asymptotique similaire. **/2**

```
La complexité asymptotique de renverse est linéaire comme celle de
l'algorithme de recherche d'une valeur (présence, position, nombre
d'occurrences)  dans un tableau.
```

10. En supposant que `swap` est correcte, justifier qu'un invariant de boucle permet de prouver la correction de `renverse`. **/1**

```
renverse est constitué d'une boucle pour qui appelle swap.
Prouver un invariant de cette boucle doit permettre de prouver la
correction de renverse.
```

11. Proposer un invariant de boucle qui permet de prouver la correction de `renverse`. **/3**

```
Avant l'itération i :
- le sous-tableau gauche t[0..i-1] contient, en ordre inversé, les i
valeurs intialement présentes dans le sous-tableau droit t[n-1-i..n-1].
- le sous-tableau droit t[n-1-i..n-1] contient, en ordre inversé, les
i valeurs intialement présentes dans le sous-tableau gauche t[0..i].
```

12. Prouver cet invariant. **/3**

```
Initialisation : avant l'itération i=0:
- le sous-tableau gauche t[0..-1] est vide et contient donc les 0
valeurs intialement présentes dans le sous-tableau droit t[n..n-1] qui
est aussi vide.

Conservation : supposons qu'avant l'itération i < p :
- le sous-tableau gauche t[0..i-1] contient, en ordre inversé, les i
valeurs intialement présentes dans le sous-tableau droit t[n-i..n-1].
- le sous-tableau droit t[n-i..n-1] contient, en ordre inversé, les i
valeurs intialement présentes dans le sous-tableau gauche t[0..i-1].

L'itération i permute t[i] et t[n-1-i] ; ainsi :
- le sous-tableau gauche t[0..i] contient maintenant les i valeurs
intialement présentes dans le sous-tableau droit t[n-1-i..n-1]
- le sous-tableau droit t[n-i..n-1] contient aussi les i+1 valeurs
intialement présentes dans le sous-tableau gauche t[0..i],
et ce en ordre inversé de chaque côté.

Terminaison : l'itération n'est pas exécutée pour i=p=n/2.
. si n est pair, alors la moitié (exactement) gauche de t contient, en
ordre inversé, les p valeurs initialement dans la moitié droite. Les
2p=n valeurs ont bien été inversées.
. si n=2p+1 est impair, seul t[p] n'a pas été modifié par la
boucle pour et la propriété attendue est vraie pour n-1=2p.
Donc renverse termine et a bien effectué le traitement attendu.
```

Exercice 2. (10 points) **/10**

Les coefficients binomiaux $\binom{n}{p}$, $0 \leq p \leq n$, utiles au calcul de $(a+b)^n$ sont définis comme suit :

$$\binom{n}{p} = \frac{n!}{p!(n-p)!}.$$

Ils sont aussi calculables avec la relation de Pascal :

$$\binom{n}{0} = \binom{n}{n} = 1 \text{ et } \binom{n}{p} = \binom{n-1}{p-1} + \binom{n-1}{p} \text{ pour } 0 < p < n.$$

1. Ecrire la fonction récursive `f` qui calcule la fonction factorielle. Bien préciser en-tête et corps. **/2**

```
. en-tête :
fonction f(n:entier) retourne entier

. corps :
fonction f(n:entier) retourne entier
debut
  si n==0 ou n==1
    retourne 1
  sinon
    retourne n*f(n-1)
fin fonction
```

2. Écrire (l'en-tête et le corps) d'une fonction `binom` qui calcule $\binom{n}{p}$ pour deux valeurs entières n et p. La valeur non significative -1 sera retournée le cas échéant. **/2**

```
fonction binom(n:entier, p:entier) retourne entier
// calcule (n p) avec fonction factorielle
// retourne -1 si n ou p <0 ou si n<p
  res : entier
  utilise f(n : entier) retourne entier
  res : entier = -1
debut
  si n >= p et n >=0 et p >=0 alors
    res = f(n)/(f(p)*f(n-p))
  fin si
  retourner res
fin fonction
```

3. Écrire un algorithme qui calcule et affiche $\binom{n}{p}$ pour deux valeurs n et p entrées au clavier. **/1**

```
// demande, calcule et affiche (n p)
declare
  n, p, res : entiers
  utilise binom(n : entier) retourne entier
debut
  lire(n)
  lire(p)
  res = binom(n,p)
  afficher(res)
```

4. Expliciter la suite d'appels nécessaire au calcul de $\binom{4}{3}$. Comptabiliser le nombre total d'appels à `f`. Comptabiliser et expliciter les appels éventuellement répétés. Qu'en penser ? **/3**

```
binom(4,3) = f(4)/(f(3)*f(1))
f(4) -> f(3) -> f(2) -> f(1)
f(3) -> f(2) -> f(1)
f(1)
Nbre total d'appels de f dans binom(4,3) = 8
Nbre de répétitions de f(3)=2, f(2)=2, f(1)=3
Ces répétitions sont coûteuses mais imposées par la définition
récursive de factorielle.
binom pourrait être calculé en simplifiant ces factorielles :
binom(n,p)=n(n-1)...(n-p+1)/(n-p)(n-p-1)...2
```

5. Expliciter et commenter deux principes du calcul de $\binom{n}{k}$ grâce à la relation de Pascal. **/2**

```
La relation de Pascal est naturellement récursive :
binom(n,p) = binom(n-1,p-1) + binom(n-1,p).
Sa terminaison est assurée par :
binom(1,0) = binom(1,1) = 1

Un solution itérative de ce calcul est possible.
Elle impose d'introduire un tableau 1D (au moins) pour sauvegarder
tout ou partie de la ligne précédente du triangle de Pascal.

Ces deux solutions effectuent exactement le même calcul avec le même
nombres d'appels à factorielle.
```