Algorithmique

Contrôle continu du 30 mars 2015
(1 heure)
Correction et barême

Connaissances évaluées
- structures de contrôle de base
- tableaux 1D et premiers traitements simples
- sous-programmes et sous-programmes avec des tableaux 1D
- recherche séquentielle
- complexité en temps
- preuve de terminaison et de correction

Exercice 1. (2 points)
Écrire l'algorithme de recherche séquentielle d'un caractère c dans une chaîne de caractères s de longueur n.

```
//role : rechercher séquentiellement le caractère c dans la chaine de
//caractères s de longueur n. Met à jour le booléen trouvé à Vrai et à Faux sinon.
declare
  c : caractère
  n : entier
  s[n] : tableau de caractères // ou s : tableau de caractères[0,n-1]
  trouvé : booléen = Faux
  i : entier = 0 // itérateur tantque
debut
  tantque (trouvé == Faux) et (i<n) faire
    si s[i] == c alors
      trouvé = Vrai
    fin si
    i = i+1
  fin tantque
fin
```

Exercice 2. (8 points dont $\star$ = 3)
On représente un vecteur de taille n et à coefficients entiers sous la forme d'un tableau 1D d'entiers indicé de 0 à $n-1$. Le produit scalaire de deux vecteurs de même taille n, $x = [x(i)]_i$ et $y = [y(i)]_i$, est l'entier s :

$$s = \sum_{i=0}^{n-1} x(i) \times y(i) = x(0) \times y(0) + x(1) \times y(1) + \cdots + x(n-1) \times y(n-1).$$

1. Écrire une fonction *prod* qui calcule s pour deux vecteurs x et y de taille arbitraire.

```
En-tête:
fonction prod(x: tableau d'entiers, y: tableau d'entiers, n: entier)
retourne entier

Corps:
fonction prod(x: tableau d'entiers, y: tableau d'entiers, n: entier)
retourne entier
  res : entier = 0
  i : entier // itérateur pour
debut
  pour i de 0 à n-1 faire
    res = res + x[i]*y[i]
```

```
  finpour
  retourne res
fin fonction
```

2. La complexité en temps de ce calcul est mesurée par le nombre d'opérations arithmétiques. Quel est le paramètre de la fonction de complexité ? Quelle est cette fonction de complexité ? Préciser la complexité asymptotique de ce calcul.

   - Le paramètre de la fonction de complexité est la taille n de chaque vecteur car celui-ci fait varier le nombre d'opérations arithmétiques effectuées.
   - C(n) = 2n car le calcul comporte n répétitions de la multiplication x[i]*y[i] et de l'addition (accumulation) dans res.
   - La complexité asymptotique est linéaire en la taille des vecteurs : C(n) = θ(n)

3. Rappeler la définition d'un invariant de boucle et son utilité.

   - Voir cours pour détails.
   - Un invariant de boucle est une propriété vraie avant, pendant et après l'itération de la boucle.
   - Il sert à prouver la correction d'un algorithme.

4. ($\star$) Proposer un invariant de la boucle de calcul de s.

   I(i): avant l'itération i de la boucle pour, res = $\sum_{k=0}^{i-1}$x[k]*y[k].

5. ($\star$) Prouver cet invariant de boucle et conclure.

   - Initialisation: avant l'itération 0, res = 0 (initialisation) ce qui correspond (par convention) à la somme de 0 valeurs entières.
   - Conservation: supposons I(i), soit res = $\sum_{0}^{i-1}$x[k]*y[k] avant l'itération i. L'exécution de cette itération calcule x[i]*y[i] et l'accumule dans res. res vaut maintenant $\sum_{0}^{i-1}$x[k]*y[k] + x[i]*y[i] = $\sum_{0}^{i}$x[k]*y[k]. Ainsi avant l'itération suivante, i+1, res vérifie bien I(i+1).
   - Terminaison: i==n est la première valeur qui arrête l'itération et I(n) donne res = $\sum_{0}^{n}$x[k]*y[k]. Cette valeur de res est retournée par prod qui ainsi termine et a bien calculé le produit scalaire de x par y.

Exercice 3. (10 points dont $\star$ = 5)
On veut calculer $y = (2 \times x)^n$ pour une valeur flottante x et un entier positif n. On utilise pour cela l'expression suivante :

$$y = \underbrace{(2 \times x) \times (2 \times x) \times \cdots \times (2 \times x)}_{n \text{ fois}}. \tag{1}$$

1. Écrire un algorithme qui :
   (a) "lit au clavier" une valeur de x et de n,
   (b) calcule y en utilisant **exclusivement** l'expression (1),
   (c) "affiche à l'écran" la valeur de y.

```
// rôle : lit x et n, calcule  (2x)^n et affiche le résultat
declare
  x : flottant
  n : entier
  y : flottant = 1.  // initialisation pour accumulation des produits
  i : entier         // itérateur pour
début
  lire(x)
  lire(n)

  pour i de 1 à n faire
```

```
    y = y * (2. * x) // 2.0 peut aussi être écrit 2 entier ...
  fin pour

  affiche(y)
fin
```

2. (a) Écrire l'en-tête de la fonction f qui reprend le calcul de la question 1b.
   (b) Écrire un algorithme principal qui utilise f pour calculer et afficher les valeurs de $y_1 = 4^3$ et de $y_2 = 3^4$.
   (c) Écrire le corps de la fonction f.

```
En-tête:
fonction f(x : flottant, n : entier) retourne flottant

Appel:
// rôle : calcule $4^3$ et $3^4$ (avec f) et affiche les résultats
declare
  y1, y2 : flottant
début
  y1 = f(2., 3.)       // 4^3
  y1 = f(3./2., 4.)    // 3^4 avec 3/2 flottant, 1.5 accepté cependant

  affiche(y1, y2)
fin


Corps :
fonction f(x : flottant, n : entier) retourne flottant
  y : flottant = 1.0
  i : entier // itérateur pour
début
  pour i de 1 à n faire
    y = y * (2.*x)
  fin pour
  retourne(y)
fin
```

3. Pour une valeur arbitraire de x, on souhaite calculer les valeurs :

$$(2 \times x), (2 \times x)^2, (2 \times x)^3, \ldots, (2 \times x)^{10}. \tag{2}$$

   (a) Écrire une fonction g qui utilise f pour calculer et retourner ces 10 valeurs sous la forme d'un tableau.

```
fonction g(x: flottant) retourne tableau de flottants
  t[10] : tableau de flottants
  utilise fonction f(x : flottant, n : entier) retourne flottant
debut
  pour i de 1 à 10 faire
    t[i-1] = f(x, i)
  finpour
  retourne t
fin fonction
```

   (b) Écrire un algorithme qui utilise g pour calculer les dix valeurs (2) pour $x = 1$ et $x = 5$. Cet algorithme affichera ces valeurs **après** que les 10 valeurs aient été calculées.

```
declare
  t1[10] : tableau de flottants
  t5[10] : tableau de flottants
début
  t1 = g(1.)  // affectation globale dans un tableau
```

```
  t5 = g(5.)  // grâce à f
  pour i de 1 à 10 faire
    afficher(t1[i])
  finpour
  pour i de 1 à 10 faire
    afficher(t5[i])
  finpour
fin
```

(c) (⋆) Proposer une fonction *g_opt* qui minimise le nombre d'opérations arithmétiques nécessaires au calcul du tableau résultat.

```
fonction g_opt(x: flottant, n: entier) retourne tableau de flottants
  t[n] : tableau de flottants
  deux_x : flottant
debut
  deux_x = 2.*x          // deux_x pas nécessaire si t[0] (=2.*x)
  t[0] = deux_x          // est réutilisé dans la boucle
  pour i de 1 à n faire
    t[i] = t[i-1] * deux_x
  finpour
  retourne t
fin fonction
```

(d) (⋆) Pour des valeurs arbitraires de x et de n, on souhaite maintenant calculer :

$$(2 \times x), (2 \times x)^2, (2 \times x)^3, \ldots, (2 \times x)^n \tag{3}$$

Effectuer une analyse de la complexité de ce calcul selon qu'on utilise g ou g_opt.

- On compte les opérations arithmétiques donc le paramètre de complexité est n.

- Le calcul de g effectue n=10 appels à f avec les valeurs 1..n
- Chaque appel de f(., i) effectue 2*i opérations arithmétiques
- Donc le calcul de g effectue 2*(1+2+ ...+ n) = n*(n+1) op.arithm.

- Le calcul de g_opt obtient deux_x en une multiplication.
- La boucle effectue n itérations dont chacune réalise une multiplication
- Le calcul de g_opt effectue (n+1) op.arithm.

Conclusion : g est quadratique tandis que g_opt est linéaire.

4. (⋆) On reprend le calcul de g de la question 3a. En supposant que le calcul de f est correct, prouver la correction du calcul de g.

Prouvons l'invariant de la boucle pour suivant:
I(i): avant l'itération i, les i-1 premières valeurs du tableau t valent 2*x, $(2*x)^2$, ..., $(2*x)^{i-1}$.

- Initialisation: avant l'itération i==1, le tableau est vide ce qui correspond à 0 premières valeurs.

- Conservation: On suppose que I(i) est vrai. On effectue l'itération i et f est supposée correcte. L'appel à f termine et retourne $(2x)^i$. Cette valeur est stockée à la i-ème position dans t et I(i) fournit les (i-1)-ème précédentes. Les i premières valeurs de t sont donc 2*x, $(2*x)^2$, ..., $(2*x)^{i-1}$ et $(2x)^i$. On passe à l'itération suivante, i+1, et I(i+1) est vrai.

- Terminaison: la valeur i==n+1 est la première valeur qui n'exécute pas le corps de la boucle et l'algorithme termine ainsi en retournant le tableau t. Ce tableau t est de taille n et il contient les n valeurs cherchées.

Barême (sur 21)

Exo 1: 2

Exo 2 : (9)

1. en-tête : 1, corps : 2
2. 0.5 (param.) + 1 (2n) + 0.5 (linéaire)
3. 1 (déf)
4. 1 (I(i))
5. 2 (preuve)

Exo 3 :(10)

1. 1 (algo)
2. 0.5 (en-tête f) + 1 f(2), f(3/2) + 1 (corps f)
3. 1 (g) + 0.5 (g(1), g(5)) + 1 (g_opt)
4. 1 + 1 (g, g_opt)
5. 1 (invariant) + 1 (sa preuve)