UNIVERSITE IBN TOFAIL
Faculté des sciences
Département d'Informatique
Kenitra

Année: 2018/2019
Filières : SMI
Semestre : 3

Algorithmique II

Examen final

Corrigé

Exercice 1 : (Sur 6 points)

Un marchand de produits représente chaque *Produit* comme une structure contenant les champs suivants :

- ***CodeProduit*, une chaine de caractères de taille maximale 10 caractères, indiquant le code du Produi.**
- ***NomProduit*, une chaine de caractères de taille maximale 20 caractères, indiquant le nom du Produit.**
- ***PrixUnitaire*, un réel indiquant le prix unitaire du produit.**
- ***Quantité*, un entier indiquant la quantité à vendre du produit.**
- ***Stock*, un entier indiquant la quantité du produit se trouvant dans le stock.**

Pour établir la facture des produits vendus pour un client le marchand range la liste des produits vendus à ce client dans un tableau de Produits nommé *Tab*.

1. **a) Donner la déclaration du type *Produit* en tant que structure regroupant les champs ci-dessus.**
   b) Donner la déclaration de *Tab* en tant que variable de type tableau de *Produit*, de taille N (N est une constante).
2. **Ecrire une procédure *AfficherFacture (Tab ; NomClient ; M : Entier)* recevant le tableau *Tab*, une chaîne de caractères *NomClient* de taille maximale 20 caractères donnant le nom du client ainsi que le nombre M de produits vendus et la procédure affichera les produits du tableau, après avoir vérifié que pour chaque produit la quantité vendue est disponible en stock. L'affichage sera comme suit :**

Nom : NomClient
Code du Produit ***Nom du Produit*** ***Prix Unitaire*** ***Quantité*** ***Prix Total***
…….. …… ……... …… ……
Total : ……...

***"Prix Total"* est le prix de la quantité vendue du produit, si elle est disponible dans le stock et "Total" est la somme des prix totaux**

Corrigé

1. a. Type Produit = structure
CodeProduit : Caractere[1..10]
NomProduit : Caractere[1..20]
PrixUnitaire : Reel
Quantité : Entier
Stock : Entier
Fin Structure
b. Var Tab : Produit[1..N]

2.

```
Procedure AfficherFacture (Tab : Produit[1..N], NomClient : Caractere[1..20], M : Entier)
Var     i : Entier
        Total : Reel
Debut
        Ecrire("\nNom : ",NomClient,"\n")
        Ecrire("\n",Code du Produit,"\t",Nom du Produit,"\t",Prix Unitaire,"\t")
        Ecrire(Quantité,"\t",Prix total)
        Total ←0
        Pour i←1 a M Faire
                Si (Tab[i].Quantité <=Tab[i].Stock) Alors
                   Ecrire("\n",Tab[i].CodeProduit,"\t",Tab[i].NomProduit,"\t")
                   Ecrire(Tab[i].PrixUnitaire,"\t", Tab[i].Quantité,"\t")
                   Ecrire(Tab[i].Quantité*Tab[i].PrixUnitaire)
                   Tota[←Total + Tab[i].Quantité*Tab[i].PrixUnitaire
                Fin Si
        Fin Pour
        Ecrire("\n\n","Total : ", Total)
Fin
```

<u>Exercice</u> 2 : (Sur 7 points)

Soit Ch[1..n] et P[1..m] deux chaînes de caractères, avec 1≤m≤n On considère la fonction *Match (P ; Ch ; m ; n)* définie par :

```
Fonction  Match (P : Caractere[1..m]; Ch : Caractere[1..n]) : Entier
Var     L, r : Entier
        B : Boolleen
Debut
        L ←0
        B ←Faux
        Tant que (L≤(n-m) Et  B=Faux) Faire
                L ← L + 1
                r ← 1
                B ← Vrai
                Tant que (r ≤m Et B=Vrai)  Faire
                        Si (P[r] <> Ch[L+r-1]) Alors
                                B ← Faux
                        Fin Si
                        r ← r + 1
                Fin Tant que
                Si (B=Vrai) Alors
                        Retourner  L
                Fin Si
        Fin Tant que
        Retourner -1
Fin
```

1. **Expliquer brièvement le but de la fonction *Match ?***
2. **Calculez la complexité temporelle *C(m, n)* dans les pires des cas de la fonction *Match*, ceci en fonction de m et n.**
3. **Modifiez la fonction *Match* de manière à ce qu'elle fournisse le nombre d'occurrences de P dans Ch (Les occurrences peuvent se chevaucher).**

<u>Corrigé</u>

1. **But de la fonction Match : La fonction *Match* retourne l'indice du début de la première occurrence de la chaîne *P* dans la chaîne *Ch*, ceci si elle existe ; sinon elle retourne -1.**
2. ***Complexité dans les pires des cas de la fonction Match***

   Dans les pires des cas on passe dans la boucle externe (n-m) fois et pour chaque passage dans cette boucle on passe m fois dans la boucle interne, d'où la complexité :

   C(m, n) =2 taffect + 3 tcomp + (n-m)*[4 tcomp +3 taffect + tadd + m*(4 tcomp + taffect + tadd) + 4 tcomp]

   Donc C(m, n)=Θ(m(n-m))
3. **Pour déterminer toutes les occurrences de la chaîne P dans la chaîne Ch on utilise la fonction MatchModifié donnée par :**

```
Fonction  MatchModifié (P :Caractere[1..m]; Ch :Caractere[1..n]) : Entier
Var       L, r : Entier
          B : Boolleen
          Compteur : Entier←0
Debut
          L ←0
          B ←Faux
          Tant que (L<=(n-m) ) Faire
                    L ← L + 1
                    r ← 1
                    B ← Vrai
                    Tant que (r <=m Et B=Vrai)  Faire
                              Si (P[r] <> Ch[L+r-1]) Alors
                                        B ← Faux
                              Fin Si
                              r ← r + 1
                    Fin Tant que
                    Si (B=Vrai) Alors
                              Compteur←Compteur+1
                    Fin Si
          Fin Tant que
          Retourner Compteur
Fin
```

Exercice 3 : (Sur 7 points)

On considère la suite récurrente donnée par :

a_0=1 ; a_1=2 et a_n= a_{n-1} * a_{n-2} pour n>1

1. **Ecrire en pseudo-code une fonction récursive *SuiteRecursive ()* qui retourne le $n^{ème}$ élément de cette suite.**
2. **Ecrire en pseudo-code une fonction itérative *SuiteIterative ()* qui retourne le $n^{ème}$ élément de cette suite.**
3. **Laquelle des deux fonctions est plus efficace. Justifier votre réponse en calculant les complexités temporelles des deux fonctions.**

Corrigé

1.

```
Fonction SuiteRécursive(n : Entier) : Entier
Début
          Si (n<0) Alors
                    Sortir
          Fin Si
```

```
        Si (n=0) Alors
                Retourner 1
        Sinon
                Si (n=1) Alors
                        Retourner 2
                Sinon
                        Retourner SuiteRécursive(n-1) * SuiteRécursive(n-2)
                Fin Si
        Fin Si
Fin
```

2.

```
Fonction SuiteItérative(n : Entier) : Entier
Var     A0, A1, A2 : Entier
        i : Entier
Début
        Si (n<0) Alors
                Sortir
        Fin Si
        Si (n=0) Alors
                Retourner 1
        Sinon
                Si (n=1) Alors
                        Retourner 2
                Sinon
                        A0← 1
                        A1← 2
                        i←2
                        Tant Que (i<=n) Faire
                                A2 ← A0 * A1
                                A0 ←A1
                                A1 ← A2
                                i ←i+1
                        Fin Tant Que
                        Retourner A1
                Fin Si
        Fin Si
Fin
```

3. Comparaison des deux fonctions

Soient T1(n) et T2(n) les complexités temporelles respectives des fonctions SuiteRécursive(n) et SuiteItérative(n), on a :

T1(n) = 2tcomp + tretour = C1 si n=0
T1(n) = 3tcomp + tretour = C2 si n=1
T1(n) = 3tcomp + tretour + tmult +2T1(n-1) = C3 + 2T1(n-1) si n>1
En développant on obtient :
$T1(n) = \Theta(2^n)$
T2(n) = 2tcomp + tretour = C1 si n=0
T2(n) = 3tcomp + tretour = C2 si n=1
T2(n) = 3tcomp + tretour +(n-1)(tcomp + 4 taffect + tmult + tadd) + tcomp + tretour si n>1
$T2(n)= \Theta(n)$
Il est clair que la fonction T2(n) est très petit par rapport à T1(n). Donc la fonction itérative est plus efficace que la fonction récursive.