UNIVERSITE IBN TOFAIL
Faculté des sciences
Département d'Informatique
Kenitra

Année : 2021/2022
Filières : SMI
Semestre : 3

Algorithmique II

Examen de rattrapage

Corrigé

Exercice 1 : (Sur 8 points)

On souhaite écrire un algorithme qui permet de gérer un ensemble d'étudiants. Chaque type Etudiant est une structure composée des champs suivants :

- Nom, une chaine de caractères
- DateNaissance, un tableau de trois entiers (jour, mois et année)
- MoyenneGénérale, un réel

1. Donner le type Etudiant et déclarer le tableau Tab, comme variable globale, constituée de N Etudiant, où N est une constante donnée.
2. Écrire les fonctions et procédures qui permettent de réaliser les tâches suivantes :
   - La fonction *Créer(nom : chaine ; j, m, an : Entier ; moy : Réel)* qui permet de créer une variable de type Etudiant, saisir le nom de cet étudiant (nom), sa date de naissance (j, m, an), sa moyenne (moy) et retourner la valeur de la variable ainsi crée.
   - La procédure *SaisirTableau()* qui permet de remplir le tableau Tab déclaré ci-dessus, en lisant les données de chaque étudiant, et en utilisant la fonction *Créer*, définie ci-dessus.
   - La procédure *CalculeMoyenne()* qui permet de calculer et afficher la moyenne de la classe constituée des étudiants du tableau Tab.

Corrigé

1. Type ETUDIANT = STRUCTURE

```
Type ETUDIANT = STRUCTURE
        Nom : CHAINE
        DateNaissance : ENTIER[1..3]
        Moyenne : REEL
Fin structure
CONST  N=100
VAR    Tab : ETUDIANT[1..N]
```

2. Procédures et fonctions :

```
Fonction Créer(nom : CHAINE ; j , m, an : ENTIER ; moy : REEL) : ETUDIANT
VAR     X : ETUDIANT
        D : ENTIER[1..3]
Début
        D[1]←j
        D[2]←m
        D[3]←an
        X.Nom ←nom
        X.DateNaissance←D
        X.Moyenne←moy
        Retourner X
Fin

Procédure SaisirTableau()
VAR     i, j, m, an : ENTIER
        moy : REEL
        nom : CHAINE
```

```
        X : ETUDIANT
Début
   Pour (i←1 à N) Faire
        ECRIRE("\nDonner le nom de l'étudiant se trouvant au rang ", i)
        LIRE(nom)
        ECRIRE("\nDate de naissance de l'étudiant se trouvant au rang ", i)
        ECRIRE("\nDonner un entier indiquant le jour de naissance :")
        LIRE(j)
        ECRIRE("\nDonner un entier indiquant le mois de naissance :")
        LIRE(m)
        ECRIRE("\nDonner un entier indiquant l'année de naissance :")
        LIRE(an)
        ECRIRE("\nDonner la moyenne de l'étudiant se trouvant au rang ", i)
        LIRE(moy)
        X←Créer(nom, j, m, an, moy)
        Tab[i]←X
   Fin Pour
Fin

Procédure CalculeMoyenne()
VAR     moyenne : REEL
        i : ENTIER
Début
   moyenne←0          //moyenne de la classe
   Pour (i←1 à N) Faire
        moyenne ← moyenne + Tab[i].Moyenne
   Fin Pour
   ECRIRE("\nLa moyenne de la classe est : ", moyenne/N)
Fin
```

Exercice 2 : (Sur 5 points)

Pour déterminer et afficher les N premiers nombres premiers c'est à dire la suite : 2, 3, 5, 7, 11, ... (où N est une constante donnée), on utilise l'algorithme *nombresPremiers ()* suivant :
On utilise un tableau d'entiers T[1 .. N], dans lequel on range les nombres premiers découverts au fur et à mesure.
On note par m le nombre d'entiers premiers connus et rangés dans T à une étape donnée du déroulement de l'algorithme.
On note par k le plus grand entier dont on a testé la primalité.

1. On considère les affectations suivantes : T[1] ← 2, m ← 2 et k ← 2.
2. On répète ce qui suit tant que m ≤ N.
   - On incrémente k (k ← k + 1).
   - Si k n'est divisible par aucun des éléments T[1],..., T[m-1], alors k est premier, et dans ce cas on affecte k à T[m] (T[m] ← k) et on incrémente m (m ← m + 1) ;
   Sinon on retourne en 2)
3. On affiche tous les éléments du tableau T.

Ecrire en pseudo-code l'algorithme *nombresPremiers (),* décrit ci-dessus.

Corrigé

```
Algorithme nombresPremiers()
Const   N = 100
Var     T : Entier[1..N]
        i, k, m : Entier
        premier : Boolleen
Début
```

```
    T[1] ← 2
    k←2                    //k est le plus grand entier placé dans T
    m←1          //m est le nombre d'entiers premiers connus et rangés dans T
    Tant que (m<N) Faire
        premier ← Vrai
        i ← 1
        Tant que (i<= m Et premier) Faire
            Si (k mod T[i]=0) Alors
                premier←Faux
            Fin Si
            i ← i+1
        Fin Tant que
        Si (premier) Alors
            m ← m+1
            T[m] ← k
        Fin Si
        k ← k+1
    Fin Tant que
    //Affichage des éléments de T
    Ecrire("\nLes ", N, " nombres premiers en tête des nombres entiers sont :")
    Pour (i ← 1 à N) Faire
        Ecrire(T[i], "\t")
    Fin pour
Fin
```

Exercice 3 : (Sur 7 points)

On considère la fonction vérifier (Ch : Chaine, deb : Entier, fin : Entier) donnée par :

```
Fonction vérifier (Ch : Chaine, deb : Entier, fin : Entier) : boolleen
Var     i, j : Entier
Début
        Si (deb >= fin) Alors
                Retourner (Vrai)
        Sinon
                Si (Ch[deb] <> Ch[fin]) Alors
                        Retourner (Faux)
                Sinon
                        Retourner (vérifier(Ch, deb+1, fin-1))
                Fin Si
        Fin Si
Fin
```

1. **Appeler la fonction *vérifier(Ch, deb, fin)* dans le cas suivant :**
   Ch = "bonjob", deb=0 et fin=5
   Et donner la valeur retournée par cet appel.
2. **Quel est le but de la fonction *vérifier(Ch, 0, n-1)*, où n est la longueur de la chaine Ch ?**
3. **Déterminer la complexité temporelle de la fonction *vérifier(Ch, 0, n-1)* en fonction de n.**

Corrigé

1. - Valeur retournée par vérifier(Ch,deb,fin) pour Ch="bonjob", deb=0 et fin=5

Comme deb<fin et Ch[deb]=Ch[fin] ='b' alors
vérifier(Ch,0,5) = vérifier(Ch,deb+1,fin-1) = vérifier(Ch,1,4)
Comme deb<fin et Ch[deb]=Ch[fin] = 'o' alors

vérifier(Ch,1,4) = vérifier(Ch,deb+1,fin-1) = vérifier(Ch,2,3)
Comme deb<fin et Ch[deb] $\neq$ Ch[fin] alors
vérifier(Ch,2,3) = Faux
Finalement, vérifier(Ch,0,5)= vérifier(Ch,2,3) = Faux

2. On montre par récurrence sur la longueur de la chaine Ch que le but de la fonction vérifier est de regarder si la chaine Ch est un palindrome, c'est-à-dire que Ch est identique à son inverse.
La propriété est vraie pour n=0 et n=1, supposons qu'elle est vraie pour n>0 et montrons qu'elle est vraie pour n+1
Soit Ch une chaine de caractères de longueur n+1, montrons que :
    - vérifier(Ch,0,n)= Vrai si Ch est identique à son inverse,
    - et vérifier(Ch,0,n) = Faux si Ch n'est pas identique à son inverse

Deux cas sont possible :
1er cas : `Ch[deb]<>Ch[fin]`

Dans ce cas, Ch n'est pas identique à son inverse, et conformément à sa description vérifier(Ch,0,n) va retourner Faux
2ème cas : `Ch[deb]= Ch[fin]`
Dans ce cas, la nature de Ch dépend de la nature de la sous chaine de Ch obtenue en éliminant les deux caractères du début et de la fin de Ch. Autrement dit, si la sous chaine est identique à son inverse alors Ch est identique à son inverse, donc vérifier(Ch,0,n)=vérifier(Ch,1,n-1)=Vrai. Si la sous chaine n'est pas identique à son inverse alors Ch n'est pas identique à son inverse, donc vérifier(Ch,0,n)=vérifier(Ch,1,n-1)=Faux.

3. Complexité temporelle de la fonction vérifier(Ch,0,n)
Soit t(n) la complexité temporelle de cette fonction alors :
t(n)=tcomp + tretour=t0, une constantesi n=1 ou n=0
t(n)=2*tcomp + tretour + t(n-2)=t1 + t(n-2)   si n>=2, où t1 est une constante.

On suppose que k= n Div 2 on a :

t(n)	= t1 + t(n-2)
t(n-2)	= t1 + t(n-4)
---------------------------------------	
t(n-2*(k-1))	= t1 + t(n-2*k)

t(n)=k*t1 +t0 $\cong$ ($\lfloor n/2 \rfloor$)t1 +t0 = θ(n)