

Fichiers (Corrigé TD2)

1. Comptes bancaires

1.1 Création de fichier

- Définir sous forme d'enregistrement le type **compte** composé des champs suivants :
 - numéro de compte (entier);
 - nom du titulaire du compte (chaîne de 20 caractères)
 - adresse du titulaire (chaîne de 100 caractères)
 - solde en euros (réel)

structure compte

```
{  
    numéro: entier  
    nom: chaîne de 20 caractères  
    adresse: chaîne de 100 caractères  
    solde: réel  
}
```

- Écrire la fonction **saisirCompte** qui permet à l'utilisateur de créer un compte en saisissant au clavier les informations, puis qui retourne ce compte. Vérifier pendant la saisie que les informations sont cohérentes (par exemple que le champ « nom du titulaire » est non-vide, que le numéro du compte n'est pas un nombre négatif, etc.).

fonction saisirCompte(): compte

```
{  
    c: compte  
  
    répéter  
        afficher("Entrez le numéro du compte :")  
        saisir(c.numéro)  
    jusqu'à (c.numéro > 0)  
    répéter  
        afficher("Entrez le nom :")  
        saisir(c.nom)  
    jusqu'à (c.nom ≠ "")  
    répéter  
        afficher("Entrez l'adresse :")
```

```

        saisir(c.adresse)
    jusqu'à (c.adresse ≠ "")
    afficher("Entrez le solde :")
    saisir(c.solde)

    retourner c
}

```

- Écrire le programme principal qui permet à l'utilisateur de créer en utilisant la fonction **saisirCompte** un fichier appelé *comptes.dat*. On suppose qu'il n'y a pas plus de 100 comptes dans le fichier.

```

Création du fichier      // Programme principal
{
    c: compte
    nb: entier
    fc: Fichier
    rep: caractère
    nom_fichier: chaîne de 30 caractères

    afficher("nom du fichier :")
    saisir(nom_fichier) // nom_fichier : comptes.dat
    Créer (nom_fichier,fc)
    Ouvrir(fc,écriture)
    nb ← 0
    répéter
        afficher("Entrez un compte")
        c ← saisirCompte() // saisir un compte
        Ecrire(fd,c) // écrire un compte dans le fichier
        nb ← nb + 1
        afficher("Voulez-vous recommencer ? (O/N)")
        saisir(rep)
    jusqu'à (nb = 100) ou (rep = 'N')
    Fermer(fc)
}

```

1.2 Manipulation de fichier

- Écrire la procédure **ajouterCompte** qui permet à l'utilisateur de saisir un compte et de le sauvegarder à la fin d'un fichier appelé *comptes.dat*

```
procédure ajouterCompte(E fc: Fichier)
{
    c: compte

    Décaler(fc, Fin_fichier, 0) // placer le pointeur (curseur) en fin du fichier
    c ← saisirCompte()
    Ecrire(fc,c)
}
```

- Écrire la fonction **afficherComptes** qui lit le contenu du fichier *comptes.dat* puis l'affiche à l'écran, et renvoie le nombre de comptes affichés.

```
fonction afficherComptes(E fc: Fichier): entier
{
    nb: entier
    c: compte

    Décaler(fc, début_fichier, 0) // placer le pointeur (curseur) au début du fichier
    nb ← 0
    tant que (non Fin(fc)) faire
    {
        Lire(fc,c) // lire un compte dans le fichier
        afficher(c.numéro, c.nom, c.adresse, c.solde)
        nb ← nb + 1
    }
    retourner nb
}
```

- Écrire la procédure **trierComptes** qui transfère le contenu du fichier *comptes.dat* dans un tableau de comptes et qui trie ce tableau dans l'ordre croissant des soldes. Puis, on affiche le tableau.

```
procédure trierComptes(E fc: Fichier)
{
```

```

cptes: vecteur de 100 compte
tmp : compte
min : réel
i,j, ind_min,nb : entier

// Chargement fichier → tableau
nb ← 0
Décaler(fc, début_fichier, 0) // placer le pointeur (curseur) au début du fichier
tant que (non Fin(fc)) faire
{
    Lire(fd,cptes[nb]) // lire un compte dans le fichier
    nb ← nb + 1
}

// Tri des comptes dans l'ordre croissant des soldes
pour j de 0 à nb - 2 faire
{
    min ← cptes[j].solde // min = le 1er solde
    ind_min ← j
    pour i de j+1 à nb -1 faire
        si (cptes[i].solde < min) alors
            {
                min ← cptes[i].solde
                ind_min ← i
            }
    si (ind_min ≠ j) alors // permutation
    {
        tmp ← cptes[j]
        cptes[j] ← cptes[ind_min]
        cptes[ind_min] ← tmp;
    }
}

// Affichage
pour i de 0 à nb - 1 faire
    afficher(cptes[i].numero,cptes[i].nom,cptes[i].adresse,cptes[i].solde)
}

```

- Écrire la procédure **majComptes** qui prend en entrée un numéro de compte afin de mettre à jour les informations (solde et/ou adresse) correspondant au compte dans le

fichier *comptes.dat*.

procédure majComptes(**E** fc: **Fichier**, **E** num: **entier**)

```
{
    c: compte
    trouvé: booléen
    rep: caractère

    Décaler(fc, début_fichier, 0) // placer le pointeur (curseur) au début du fichier
    trouvé ← faux
    tant que (non Fin(fc)) et (non trouvé) faire
    {
        Lire(fc,c) // lire un compte dans le fichier
        si (c.numero = num) alors trouvé ← vrai
    }
    si (trouvé) alors
    {
        afficher("Nouvelle adresse ? (O/N)")
        saisir(rep)
        si (rep = 'O') alors
            répéter
                afficher("Entrez l'adresse :")
                saisir(c.adresse)
            jusqu'à (c.adresse ≠ "")
        afficher("Nouveau solde ? (O/N)")
        saisir(rep)
        si (rep = 'O') alors
            {
                afficher("Entrez le solde :")
                saisir (c.solde)
            }
        Décaler(fc,Position_courante,-1) // revenir sur l'emplacement du compte
        Ecrire(fc,c)
    }
}
```

- Écrire le programme principal qui permettra à l'utilisateur de choisir entre les opérations ci-dessus ou quitter le programme. L'ouverture et la fermeture du fichier seront faites dans le programme principal.

Manipulation de fichier // Programme principal

```
{
    rep, num,nb: entier
    fc: Fichier
    nom_fichier: chaîne de 30 caractères

    afficher("nom du fichier :")
    saisir(nom_fichier) // nom_fichier : comptes.dat
    Désigner(nom_fichier,fc)
    Ouvrir(fc,lecture/écriture)
    Répéter
        afficher("1: Ajout, 2: Affichage, 3: Tri, 4: MaJ, 5: Quitter")
        saisir(rep)
        selon rep
        {
            1: ajouterCompte(fc)
            2: nb ← afficherComptes(fc)
               afficher(nb," comptes affichés")
            3: trierComptes(fc)
            4: afficher("Entrez le numéro de compte")
               saisir(num)
               majComptes(fc,num)
        }
    jusqu'à (rep = 5)
    Fermer(fc)
}
```

2. Relevés météo

On considère un fichier *relevés.dat* contenant des relevés météo stockés sous la forme suivante :

NOM_VILLE HEURE1 TEMPERATURE1 HEURE2 TEMPERATURE2

Exemple de contenu : *Limoges 6 -1,5 18 12,5 Reykjavik 4 -15 16 3 ...*

NOM_VILLE est une chaîne de 20 caractères, *HEURE* est un entier et *TEMPERATURE* est un réel.

- Définir le type *relevés_météo*

structure *relevés_météo*

```
{  
    Nom: chaîne de 20 caractères  
    H1, H2: entier  
    T1, T2: réel  
}
```

- Ecrire une procédure permettant d'afficher à l'écran les informations lues dans le fichier sous la forme :

Limoges :

- 6h : -1,5 degrés

- 18h : 12,5 degrés

Afficher également la température moyenne calculée sur la base du premier relevé météo de chaque ville.

procédure *Affichage()*

```
{  
    f: fichier  
    i: entier  
    M: réel  
    r: relevés_météo  
  
    Désigner("releves.dat",f) } // en principe, dans le programme principal  
    Ouvrir(f,lecture)  
    i ← 0          M ← 0  
    tant que (non Fin(f)) faire  
    {  
        Lire(f,r)      // une lecture dans le fichier  
        afficher(r.Nom, r.H1, r.T1, r.H2, r.T2)  
        M ← M + r.T1  
        i ← i + 1  
    }
```

```

    }
    Fermer(f)
    afficher(M/i)
}

```

- Ecrire une procédure pour saisir le nom d'une ville (Limoges par exemple) et son code postal international (F87000 par exemple) afin de remplacer dans le fichier le nom saisi (s'il existe) par ce code.

Procédure Mise_à_jour()

```

{
    f: fichier
    Nom: chaîne de 20 caractères
    r: relevés_météo
    trouvé: booléen

    afficher("nom ?")
    Saisir(Nom)
    Désigner("relevés.dat",f) } // en principe, dans le programme principal
    Ouvrir(f,lecture/écriture) }
    trouvé ← faux
    tant que ((non Fin(f)) et (non trouvé) faire
    {
        Lire(f,r) // une lecture dans le fichier
        si (r.Nom = Nom) alors
        {
            trouvé ← vrai
            afficher("code postal ?")
            Saisir(r.Nom)
            Décaler(f,position_courante,-1) // revenir sur l'emplacement initial
            Ecrire(f,r) // une écriture dans le fichier
        }
    }
    Fermer(f)
}

```

3. Fichiers texte

On suppose que dans les exercices de cette partie on passe en argument des fonctions un pointeur sur un fichier contenant déjà des données ; l'ouverture et la fermeture du fichier seront faites dans le programme principal.

- Écrire la procédure **afficherFichier** qui affiche le contenu du fichier à l'écran.

```
procédure afficherFichier(E f: Fichier)
{
    c: caractère

    tant que (non Fin(f)) faire *
    {
        Lire(f,c)      // lecture d'un caractère dans le fichier
        afficher(c)
    }
}
```

- Écrire la procédure **copierFichier** qui copie les données vers un autre fichier dont le nom est saisi au clavier par l'utilisateur.

```
procédure copierFichier(E f_source: Fichier)
{
    f_destination: Fichier
    c: caractère
    nom: chaîne de caractères

    afficher("Entrez le nom du nouveau fichier")
    saisir(nom)
    Créer(nom, f_destination)
    Ouvrir(f_destination, écriture)
    tant que (non Fin(f_source)) faire
    {
        Lire(f_source,c)      //lecture d'un caractère dans le fichier source
        Ecrire(f_destination, c) //écriture d'un caractère dans le fichier destination
    }
}
```

* Contrairement au fichier de comptes bancaires utilisant **Décaler**(fc, **début_fichier**, 0) et **Décaler**(fc, **fin_fichier**, 0), dans ce programme on utilise uniquement **Décaler**(fc, **début_fichier**, 0) dans le programme principale.

```

Fermer(f_destination)
}

```

- Écrire la procédure **analyserFichier** qui affiche un rapport d'analyse contenant :
 - le nombre total de caractères dans le fichier ;
 - le nombre de caractères représentant des chiffres ('0', '1', etc.) ;
 - le nombre de voyelles ;
 - le nombre de lignes (une ligne est terminée par le caractère de contrôle *EOL*)

procédure analyserFichier(**E f: Fichier**)

```

{
  c: caractère
  nb, nbc, nbv, nbl: entier

  nb ← 0  nbc ← 0  nbv ← 0  nbl ← 0
  tant que (non Fin(f)) faire
  {
    Lire(f,c)
    nb ← nb + 1
    si (c = 'a') ou (c = 'e') ou ... alors nbv ← nbv + 1
    si (c = '0') ou (c = '1') ou ... alors nbc ← nbc + 1 //ou l'intervalle en ASCII
    si (c = '\n') alors nbl ← nbl + 1      //\n : EOL
  }
  afficher("Nb de caractères : ", nb)
  afficher("Nb de voyelles : ", nbv)
  afficher("Nb de chiffres : ", nbc)
  afficher("Nb de lignes : ", nbl)
}

```

- Modifier la procédure d'affichage pour que chaque ligne affichée contienne au plus 10 caractères.

procédure afficherFichier_ligne(**E f: Fichier**)

```

{
  c: caractère
  taille_ligne: entier

```

```

taille_ligne ← 0
tant que (non Fin(f)) faire
{
    Lire(f,c)
    afficher(c)
    si (c = '\n') alors taille_ligne ← 0
    sinon taille_ligne ← taille_ligne + 1
    si (taille_ligne = 10) alors
    {
        afficher('\n')
        taille_ligne ← 0
    }
}
}

```

- Modifier la procédure d'analyse pour afficher également :
 - le nombre de mots – on considère qu'un mot est une suite de caractères ne contenant ni ponctuation ni espace. Un espace peut être une suite de blancs, de tabulations (caractère de contrôle *TAB*) ou fins de ligne ;
 - le nombre de palindromes (mots qui peuvent se lire dans les deux sens).

```

fonction palindrome(E mot: chaîne de caractères, E taille: entier): booléen
{
    g,d: entier

    g ← 0      d ← taille-1;
    tant que ((g<d) et (mot[g]=mot[d])) faire    //avancer dans le mot à analyser
    {
        g ← g + 1      d ← d - 1
    }
    si (g ≧ d) alors retourner vrai
    sinon retourner faux
}

```

```

procédure nombre_mots(E f: Fichier)
{

```

```

c: caractère
nbmots, nbpal, taille_mot: entier
dans_un_mot: booléen
mot: chaîne de caractères

nbmots ← 0                nbpal ← 0
dans_un_mot ← faux       taille_mot ← 0
tant que (non Fin(f)) faire
{
    Lire(f,c)
    si (c = ' ') ou (c = '\t') ou (c = '\n') ou (c = '.') ou ... alors
    {
        si (dans_un_mot) alors    // fin d'un mot
        {
            nbmots ← nbmots + 1
            si (palindrome(mot, taille_mot)) alors nbpal ← nbpal + 1
            dans_un_mot ← faux
            taille_mot ← 0
        }
    }
    sinon
    {
        dans_un_mot ← vrai
        mot[taille_mot] ← c
        taille_mot ← taille_mot + 1
    }
}
afficher("nb. de mots: ", nbmots, "nb. de palindromes: ", nbpal)
}

```

- Écrire le programme principal qui doit permettre à l'utilisateur de choisir entre ces différentes opérations.

```

Fichier texte    // Programme principal
{

```

```

f: Fichier
nom: chaîne de caractères
rep: entier

afficher("Entrez le nom du fichier texte : ")
saisir(nom)
Désigner(nom, f)
Ouvrir(fc,lecture)
répéter
    Décaler(f, Début_fichier, 0)           // nécessaire après la 1ère itération
    afficher("1: afficher, 2: copier, 3: analyser, 4: afficher par ligne,
        5: afficher le nb. de mots, 6: quitter")
    saisir(rep)
    selon (rep)
    {
        1: afficherFichier(f)
        2: copierFichier(f)
        3: analyserFichier(f)
        4: afficherFichier_ligne(f)
        5: nombre_mots(f)
    }
jusqu'à (rep = 6)
Fermer(f)
}

```