Corrigé de l'examen de Structures de Fichiers 2021/2022

1) Donnez les déclarations des structures de données nécessaires.

```
// un type d'enregistrements quelconque …
Type    Tenreg = struct
                    cle:typecle
                    …
        fin
// le type des blocs et buffers …
Type    Tbloc = struct
                    tab : tableau[ b ] de Tenreg
                    nb : entier
        fin
// les variables fichiers, buffers, …
Var                                          /* tete   , queue , NbBlc , NbEnreg */
        F1 : fichier de Tbloc buffer buf1 Entete( entier , entier , entier , entier )
        F2 : fichier de Tbloc buffer buf2 Entete( entier , entier , entier , entier )
        M : typecle
```

2) Donnez un algorithme pour une recherche rapide d'un enregistrement ayant une clé C donnée.

```
Rech( F, c, var trouv, var i, var j )
debut   // supposons que F est déjà ouvert …
        T ← Entete( F,1) ; Q ← Entete( F,2)        // la tête et la queue du tableau circulaire
        N ← Entete(F,3)                            // le nombre total de blocs réservés pour le fichier
        stop ← faux ; trouv ← faux
        bi ← 1 ; bs ← ( Q – (T-N) + 1) mod N       // le nombre de blocs utilisés = Q-(T-N)+1 (c-a-d Q-T+1 en circulaire)
        TQ ( non stop et non trouv et bi ≤ bs
                m ← (bi + bs) div 2                // le numéro logique du milieu
                i ← [(T+m-2) mod N] + 1            // le numéro réel du bloc associé à m
                LireDir( F, i, buf )
                SI ( c < buf.tab[1].cle )
                        bs ← m-1
                SINON
                        SI (c > buf.tab[ buf.nb ].cle )
                                bi ← m+1
                        SINON
                                rech_interne(c , trouv, j) ;  stop ← vrai
                        FSI
                FSI
        FTQ
        SI ( bi > bs )
                i ← [(T+bi-2) mod N] + 1                    // le numéro réel du bloc associé à bi
                SI ((Q mod N)+1 <> i) LireDir( F, i, buf )  // et son contenu dans buf
                SINON buf.nb ← 0                            // cas où i dépasse la queue Q
                FSI
                j ← 1
        FSI
fin  // Rech

rech_interne( c, var trouv , var j )
debut
        trouv ← faux ; inf ← 1 ; sup ← buf.nb
        TQ ( inf ≤ sup et non trouv )
                j ← (inf+sup) div 2
                SI ( c < buf.tab[j].cle )    sup ← j-1
                SINON SI ( c > buf.tab[j].cle ) inf ← j+1  SINON trouv ← vrai FSI
                FSI
        FTQ
        SI ( inf > sup )    j ← inf   FSI
fin // rech_interne
```

3) Donnez un algorithme pour l'insertion d'un nouvel enregistrement *e* (avec une clé < M) générant le moins de décalages inter-blocs possibles.

Les décalages se font dans la direction qui minimise le nombre d'accès disque (soit vers la queue, soit vers la tête, selon la distance qui sépare le bloc courant des 2 extrémités)
Les décalages s'arrêtent dès qu'on trouve un bloc non plein
Dans le cas où tous les blocs visités (par les décalages) étaient plein à 100 %, un nouveau bloc sera rajouté au fichier :
- soit en incrémentant la queue Q ← (Q mod N)+1 [décalages vers la fin]
- soit en décrémentant la tête T ← T-1 ; SI (T = 0) T ← N FSI [décalages vers le début]

```
Ins( e )
debut
        insertionOK ← vrai
        // supposant que le fichier F1 est déjà ouvert …
        T ← Entete(F1, 1) ;  Q ← Entete(F1, 2) ; N ← Entete(F1, 3)
        SI ( non Vide(F1) )                    // si F1 contient au moins un bloc ...
                rech( F1 , e.cle , trouv , i , j )
                SI ( non trouv )
                   SI ( i = (Q mod N)+1 ) // cas particulier d'une insertion dans un nouv bloc à la fin du fichier ...
                        SI (non Plein(F1) )
                                Q ← (Q mod N) +1 ; buf.nb ← 1 ; buf.tab[1] ← e ; EcrireDir( F1, Q, buf )
                        SINON
                                ecrire(« Fichier plein : insertion impossible »)
                                insertionOK ← faux
                        FSI
                   SINON                       // cas général : le bloc i se trouve entre T et Q
                        // on doit insérer e dans le bloc i à la position j …
                        SI ( buf.nb < b )  // insertion dans le bloc i
                                buf.nb++ ;  k ← buf.nb
                                TQ ( k > j )
                                        buf.tab[k]← buf.tab[k-1] ; k--
                                FTQ
                                buf.tab[ j ] ← e
                                EcrireDir( F1 , i , buf )
                        SINON
                                // le bloc i est déjà plein, donc insertion par décalages inter-blocs :
                                // la direction des décalages dépend de la distance avec T et avec Q
                                // dq:distance entre i et Q   ,   dt:distance entre i et T
                                SI ( T ≤ Q )
                                        dq ← |i-Q| ;   dt ← |i-T|
                                SINON // donc T > Q
                                        SI ( i ≤ Q )        dq ← |i-Q| ;  dt ← |T-N-i|
                                        SINON               dq ← |i-N-Q| ; dt ← |i-T|
                                        FSI
                                FSI
                                SI ( dt < dq )      Decalages_vers_debut( i , j-1 , T , Q , N )
                                SINON               Decalages_vers_fin( i , j , T , Q , N )
                                FSI
                        FSI // ( buf.nb < b)
                   FSI // ( i = (Q mod N)+1 )
                FSI // (non trouv )
        SINON  // F1 est vide …
                T ← 1 ;  Q← 1 ; N ← 1 ; buf.tab[1] ← e ;   buf.nb ← 1 ;  EcrireDir( F1 , T , buf )
        FSI // ( non vide(F1) )
        SI ( insertionOK )
                Aff_Entete( F1, 1 , T ) ; Aff_Entete( F1, 2 , Q ) ; Aff_Entete( F1, 4 , Entete(4)+1 )
        FSI
fin // Ins

Vide( F ) : bool
debut   //  |___|___|_Q_|_T_|___|___| le fichier est vide si T=Q+1
        SI ( (Q mod N)+1 <> T )   retourner faux   SINON retourner vrai  FSI
fin // Vide
```

```
Plein( F ) : bool
debut
        x ← T-1 ; SI (x = 0) x ← N FSI
        // on sacrifie un bloc entre Q et T ( … |__|__|_Q_|////|_T_|__|__| … )
        SI ( Q mod N)+1 <> x ) retourner faux FSI
        retourner vrai
fin // Plein

Decalages_vers_debut( i, j, T, Q, N )
// décalages vers le début, à partir du bloc i (qui est plein), jusqu'à trouver un bloc non plein
// ou alors jusqu'à atteindre le début du fichier (le bloc T) si tous les blocs sont pleins
debut
        stop ← faux ;
        SI ( j = 0 ) // insertion avant la position 1 du bloc i ⇒ insertion à la dernière position de i-1
                SI ( i <> T )       // si i n'est pas le 1er bloc ...
                        i ← i-1 ;  SI ( i = 0 ) i ← N FSI
                        LireDir( F1 , i , buf )
                        j ← buf.nb
                        SI ( buf.nb < b )
                                buf.tab[ j+1 ] ← e ; buf.nb++ ; EcrireDir(F1 , i , buf) ;  stop ← vrai
                        FSI
                SINON // ( i = T )
                        stop ← vrai
                        SI ( non Plein(F1) )   // si le fichier n'est pas plein
                                // étendre le fichier par la tête …
                                T ← T-1 ;  SI ( T = 0 ) T ← N FSI
                                buf.nb = 1 ;   buf.tab[1] ← e
                                EcrireDir( F1 , T , buf )
                        SINON
                                ecrire(« Fichier Plein : insertion impossible »)
                                insertionOK ← faux
                        FSI
                FSI
        FSI // (j=0)
        TQ ( non stop )
                SI ( buf.nb  < b )
                        buf.nb++ ; buf.tab[ buf.nb ] ← e
                        EcrireDir( F1 , i , buf )
                        stop ← vrai
                SINON
                        sauv_prem ← buf.tab[1]
                        POUR (k = 1 , j-1)  buf.tab[k] ← buf.tab[k+1]   FP
                        buf.tab[ j ] ← e ;  e ← sauv_prem
                        EcrireDir( F1 , i , buf )
                        // passer au bloc précédent ...
                        SI ( i = T )
                                SI ( non Plein(F1) )   // si le fichier n'est pas plein
                                        // étendre le fichier par la tête …
                                        T ← T-1 ;  SI ( T = 0 ) T ← N FSI
                                        buf.nb ← 1 ; buf.tab[1] ← e ; EcrireDir(F1, T, buf)
                                SINON
                                        ecrire(« Fichier plein : insertion impossible »)
                                        insertionOK ← faux
                                FSI
                                stop ← vrai
                        SINON
                                i ← i-1 ; SI ( i = 0 )  i ← N  FSI
                                LireDir( F1, i, buf )
                                j ← buf.nb
                        FSI
                FSI // (buf.nb < b)
        FTQ // (non stop)
fin // Decalages_vers_debut
```

```
Decalages_vers_fin( i, j, T, Q, N )
// décalages vers la fin, à partir du bloc i (qui est plein), jusqu'à trouver un bloc non plein
// ou alors jusqu'à atteindre la queue du fichier (le bloc Q) si tous les blocs sont pleins
debut
        stop ← faux ;
        TQ ( non stop )
                sauv_dern ← buf.tab[ buf.nb ]
                k ← buf.nb
                TQ ( k > j )   buf.tab[k] ← buf.tab[k-1] ; k-- FTQ
                buf.tab[ j ] ← e
                SI ( buf.nb  < b )
                        buf.nb++
                        buf.tab[ buf.nb ] ← sauv_dern
                        EcrireDir( F1 , i , buf )
                        stop ← vrai
                SINON
                        e ← sauv_dern
                        // passer au bloc suivant ...
                        SI ( i = Q )
                                SI ( non Plein(F1) )   // si le fichier n'est pas plein
                                        // étendre le fichier par la queue …
                                        Q ← (Q mod N) +1
                                        buf.nb ← 1 ; buf.tab[1] ← e ; EcrireDir( F1, Q, buf )
                                SINON
                                        ecrire(« Fichier plein : insertion impossible »)
                                        insertionOK ← faux
                                FSI
                                stop ← vrai
                        SINON
                                i ← (i mod N) + 1
                                LireDir( F1, i, buf )
                                j ← 1
                        FSI
                FSI // (buf.nb < b)
        FTQ // (non stop)

fin // Decalages_vers_fin
```

4) On voudrait maintenir un certain équilibre, en nombre d'enregistrements, entre le fichier **F1** et le fichier **F2**. Pour cela, lorsque la différence en nombre d'enregistrements de **F1** et **F2** dépasse un certain seuil **D**, on effectue une opération de rééquilibrage. Cette dernière, consiste à transférer des enregistrements du fichier le plus chargé vers le moins chargé afin d'avoir approximativement le même nombre d'enregistrements dans **F1** et dans **F2**. Durant cette opération, on fera en sorte que les nouveaux blocs rajoutés éventuellement au fichier de destination, soient remplis à 70 % de leur capacité maximale. A la fin de cette opération, la valeur de **M** sera aussi mise à jour en conséquence.
- Donnez un algorithme, le plus efficace possible, pour effectuer un rééquilibrage entre **F1** et **F2**, sachant que le nombre d'enregistrements de **F1** dépasse celui de **F2** d'une quantité supérieure au seuil **D.**

Si NbEnr1 > NbEnr2 + D
  Transférer les X derniers enreg de F1 vers le début de F2 ⇒ Q1 recule (mod N) et T2 recule (mod N)
    avec X = NbEnr1 – (NbEnr1 + NbEnr2)/2

```
EquilibrageF1versF2( D : entier )
// réalise un transfert d'enregistrements de F1 vers F2 si la différence dépasse D
debut
  // supposant que les fichiers F1 et F2 sont déjà ouverts …
  T1 ← Entete(F1,1) ; Q1 ← Entete(F1,2) ; N1 ← Entete(F1,3) ; NbEnr1 ← Entete(F1,4)
  T2 ← Entete(F2,1) ; Q2 ← Entete(F2,2) ; N2 ← Entete(F2,3) ; NbEnr2 ← Entete(F2,4)
  SI ( NbEnr1 > NbEnr2 + D )      // faire un transfert vers F2 de X enregistrements
        X ← NbEnr1 – (NbEnr1 + NbEnr2)/2
        LireDir( F1, Q1, buf1 )
        LireDir( F2, T2 , buf2 )
        EquilibrageOK ← vrai
```

```
        TQ ( X > 0 )
            SI ( buf2.nb < (b * 0.7) )  // remplir les nouveaux bloc à 70 %
                POUR (k=buf2.nb+1, 2, -1)  buf2.tab[k] ← buf2.tab[k-1] FP
                buf2.tab[ 1 ] ← buf1.tab[ buf1.nb ]
                buf2.nb++
                buf1.nb--
                SI ( buf1.nb = 0 )
                    // on libère le bloc de queue dans F1 …
                    Q1 ← Q1 – 1 ; SI ( Q1 = 0 ) Q1 ← N1  FSI
                    LireDir( F1 , Q1 , buf1 )
                FSI
            SINON
                // étendre le fichier F2 par la tête …
                SI ( non Plein(F2) )
                    M ← buf2.tab[ 1 ].cle
                    EcrireDir( F2 , T2 , buf2 )
                    T2 ← T2 – 1 ; SI ( T2 = 0 ) T2 ← N2 FSI
                    buf2.nb ← 0
                    x++  // pour ne pas comptabiliser cette itération
                SINON
                    ecrire(« il n'y a plus d'espace pour le rééquilibrage »)
                    EquilibrageOK ← faux
                FSI // (non Plein(F2))
            FSI // ( buf2.nb < b*0.7 )
            x--
        FTQ ( x > 0 )
        SI ( buf2.nb > 0 )  // dernière écriture si le buffer n'est pas vide
            M ← buf2.tab[ 1 ].cle
            EcrireDir( F2 , T2 , buf2 )
        SINON
            // on avance T2 vers le 1er bloc non vide
            T2 ← ( T2 mod N2 ) + 1
        FSI
        SI ( EquilibrageOK )
            Aff_entete( F1, 1, T1 ) ; Aff_entete( F1, 2, Q1 ) ; Aff_entete( F1, 4, NbEnr1-X )
            Aff_entete( F2, 1, T2 ) ; Aff_entete( F2, 2, Q2 ) ; Aff_entete( F2, 4, NbEnr2+X )
        FSI
    FSI // ( NbEnr1 > NbEnr2 )

Fin // Equilibrage
```

- Donnez une formulation estimant le coût d'une opération de rééquilibrage.

Posons F1 le fichier source et F2 le fichier destination.
Il faut lire environs **X/(b*u)** blocs (c-a-d 1 + X div (b*u)) du fichier source et écrire environs **X/(b*0.7)** blocs (c-a-d 1 + X div (b*0.7)) dans le fichier destination. Comme le premier bloc du fichier de destination est aussi lu la première fois, on peut rajouter +1 au nombre de lectures total.
u est le facteur de chargement moyen du fichier source. On peut l'estimer par : **NbEnr1 / (N1*b)** pour le cas de F1
Comme **X = NbEnr1 – (NbEnr1 + NbEnr2)/2**
Le **coût total** sera donc au voisinage de :

2 +	**(NbEnr1 – (NbEnr1 + NbEnr2)/2) * N1 div NbEnr1**	**lectures disque**
+ 1 +	**(NbEnr1 – (NbEnr1 + NbEnr2)/2) div (b*0.7)**	**écritures disque**

Si on pose **X** = NbEnr1 – (NbEnr1 + NbEnr2)/2
Le coût total sera au voisinage de : **3 + X * (N1 div NbEnr1 + 7 div b)**