Devoir S F S D

Les algorithmes de :

* **Recherche**
* **Insertion**
* **Suppression**
* **Requête a intervalle**

Type : TOF

type tbloc : structure

tab : tableau [1..b] de tenreg

NB : entier

fin structure

type tenreg : structure

cle : entier

efface : booleen

champ1 :typeQLq …

fin structure

f = fichier de tbloc buffer buf ENTETE (entier , entier)

// 1 car = nombre des blocs et 2 car = nombre des éléments dans le fichier

```
Rech( c:typeqlq; nomfich:chaine; var Trouv:bool; var i,j:entier )
var
    bi, bs, inf, sup : entier;
    trouv, stop : booleen;

DEBUT
        Ouvrir( F, nomfich, 'A' );
        bs := entete( F,1 );      // la borne sup (le num du dernier bloc de F)
        bi : = 1;                 // la borne inf  (le num du premier bloc de F)
        Trouv := faux;  stop := faux; j := 1;

        TQ  ( bi <= bs  et  Non Trouv et Non stop )
           i := (bi + bs) div 2;     // le bloc du milieu entre bi et bs
           LireDir( F, i, buf );

           SI ( c >= buf.tab[1].cle  et  c <= buf.tab[buf.NB].cle )
                 // recherche dichotomique à l'intérieur du bloc (dans la variable buf)...
                 inf := 1;  sup := buf.NB;
                 TQ  inf <= sup  et  Non Trouv
                          j : = (inf + sup) div 2;
                          SI  c  = buf.tab[j].cle:  Trouv := vrai
                          SINON
                                   SI c  < buf.tab[j].cle:      sup := j-1
                                   SINON                        inf := j+1
                                   FSI
                          FSI
                 FTQ
                 SI ( Non Trouv )
                          j := inf
                 FSI
                 // fin de la recherche interne. j indique l'endroit où devrait se trouver c
                 stop := vrai

           SINON  // non ( c >= buf.tab[1].cle  et  c <= buf.tab[buf.NB].cle )
                 SI ( c < buf.tab[1].cle )
                          bs := i-1
                 SINON   // c > buf.tab[buf.NB].cle
                          bi := i+1
                 FSI
           FSI
        FTQ

        SI ( Non Trouv )
                 i := bi
        FSI
        fermer( F )
FIN
```

```
Inserer( e:Tenreg; nomfich:chaine )
var
    trouv : booleen;
    i,j,k : entier;
    e,x : Tenreg;

DEBUT
   // on commence par rechercher la clé e.cle avec le module précédent pour localiser l'emplacement (i,j)
   // où doit être insérer e dans le fichier.
   Rech( e.cle, nomfich, trouv, i, j );
   SI  ( Non trouv )                    // e doit être inséré dans le bloc i à la position j
         Ouvrir( F,nomfich, 'A');       // en décalant les enreg j, j+1, j+2, ... vers le bas
         continu := vrai;
                                                // si i est plein, le dernier enreg de i doit être inséré dans i+1
         TQ ( continu  et  i<= entete(F,1) )    // si le bloc i+1 est aussi plein son dernier enreg sera
               LireDir( F, i, buf );            // inséré dans le bloc i+2, etc ... donc une boucle TQ.
               // avant de faire les décalages, sauvegarder le dernier enreg dans une var x ...
               x := buf.tab[buf.NB];

               // décalage à l'intérieur de buf ...
               k := buf.NB;
               TQ  k > j
                     buf.tab[k] := buf.tab[k-1];
                     k := k-1
               FTQ

               // insérer e à la pos j dans buf ...
               buf.tab[j] := e;

               // si buf n'est pas plein, on remet x à la pos NB+1 et on s'arrête ...
               SI ( buf.NB < b )            // b est la capacité max des blocs (une constante)
                     buf.NB := buf.NB+1;
                     buf.tab[buf.NB] := x;
                     EcrireDir( F, i, buf );
                     continu := faux;

               SINON // si buf est plein, x doit être inséré dans le bloc i+1 à la pos 1 ...
                     EcrireDir( F, i, buf );
                     i := i+1;
                     j := 1;
                     e := x;  // cela se fera (l'insertion) à la prochaine itération du TQ
               FSI  // non ( buf.NB < b )
         FTQ

         // si on dépasse la fin de fichier, on rajoute un nouveau bloc contenant un seul enregistrement e
         SI i > entete( F, 1 )
               buf.tab[1] := e;
               buf.NB := 1;
               EcrireDir( F, i, buf );    // il suffit d'écrire un nouveau bloc à cet emplacement
               Aff-entete( F, 1, i );     // on sauvegarde le num du dernier bloc dans l'entete 1
         FSI

         Aff-entete( F, 2 , entete(F,2)+1 );        // on incrémente le compteur d'insertions
         Fermer( F );
   FSI

FIN
```

La suppression logique consiste à rechercher l'enregistrement et positionner le champs 'effacé' à vrai :

```
Suppression( c:typeqlq; nomfich:chaine )
var
    trouv : booleen;
    i,j : entier;
DEBUT
   // on commence par rechercher la clé c pour localiser l'emplacement (i,j) de l'enreg à supprimer
   Rech( c, nomfich, trouv, i, j );
   // ensuite on supprime logiquement l'enregistrement
   SI  ( trouv )
         Ouvrir( F,nomfich, 'A');
         LireDir( F, i, buf );       // lecture pas vraiment nécessaire à cause de l'effet de bord de Rech sur buf
         buf.tab[j].effacé := VRAI;
         EcrireDir( F, i, buf );
         Fermer( F )
   FSI
FIN // suppression
```

suppression_physique(c : typeqlq ; nomfichier : chaine)

Var

Trouv : booleen ; i,j,k : entier

Debut

Rech(c,nomfichier,trouv,i,j)

Si (trouv) alors

Ouvrir(f,nomfichier,'r+')

Liredir(f,i,buf)

k := j

Si (j != buf.NB) alors

Tq (k < buf.NB) faire

buf.tab[k] := buf.tab[k+1]

k := k +1

fin tq

(1) : i := i + 1

si (i < entete(f,1)) alors

```
                    buf1 := lire(f,i)
                    buf.tab[buf.NB] := buf1.tab[1]
                fin si
                ecriredir(f,i-1,buf)
                buf := buf1
                Tq ( i < entete(f,1)) faire
                    k := 1
                    Tq (k < buf.NB) faire
                        buf.tab[k] := buf.tab[k+1]
                        k := k +1
                    fin tq
                    i := i + 1
                    si ( i < entete(f,1)) alors
                        liredir(f,i,buf1)
                        buf.tab[buf.NB] := buf1.tab[1]
                    fin si
                    ecriredir(f,i-1,buf)
                    buf := buf1
                fin tq
            sinon
                aller a (1)
            fin si
        i := entete(f,1)
        lirebloc(f,i,buf)
        si (buf.NB == 1 )          // si le dernier bloc contient une seule élément donc
lorsque en faire le décalage le dernier bloc vas supprimer
            nb := entete(f,1)
            affecte_entete(f,1,nb-1)
        finSI
        n := entete(f,2)
```

```
        affecte_entete(f,2,n-1)
        sinon
                Ecrire (" L' élément n'existe pas ! ")
Fermer(f)

Fin

articles : structure
        tab : tableau[1..100]de typeEnreg
        nb : entier
fin structure
```

Procédure Requete(a:entier, b:entier , nomfichier : chaine ; var articles : articles)

```
Var
Trouv : booleen ;       i,j : entier
Debut
        Rech(a,nomfichier,trouv,i,j)
        k := 1
        nb := 0
        si (trouv) alors
                Ouvrir(f,nomfichier,'r+')
                tantque(i<=entete(f,1))faire
                        Liredir(f,i,buf)
                        tantque(j<=buf.NB)faire
                                si( buf.tab[j].cle <= b )
                                        articles.tab[k] := buf.tab[j]
                                        k := k + 1
                                        j := j + 1
                                        articles.nb := articles.nb + 1
```

```
                    sinon
                        aller a (2)
                fin tq
                j := 1
                i := i + 1
            fin tq

        sinon
            Ecrire("L' élément n'existe pas!")
        finsi
 (2) fermer(f)
Fin
```

Type TNOF

```
type tbloc : structure
        tab : tableau [1..b] de tenreg
        NB : entier
fin structure
type tenreg : structure
        cle : entier
        efface : booleen
        champ1 :typeQLq  …
fin structure
f = fichier de tbloc buffer buf ENTETE (entier , entier)
            // 1 car = nombre des blocs et  2 car = nombre des éléments dans  le fichier
```

Procédure Recherche(c:entier ; nomfichier : chaine ; var trouv :booleen ; i,j:entier)

```
Var
i := 1 ; trouv := faux ;
Debut
        ouvrir(f,nomfichier,'r')
        TQ ( i<=entete(f,1) et trouv == faux) faire
                liredir(f,I ,buf)
                j := 1
                TQ ( j<=buf.NB et trouv == faux) faire
                        si ( buf.tab[j].cle == c  et buf.tab[j].efface == faux) alors
                                trouv := vrai
                        sinon
                                j := j + 1
                fin TQ
                i := i + 1
        fin TQ
        si ( trouv := faux ) alors
                i := entete(f,1)
                j := buf.NB
        fermer(f)
Fin
```

insertion(c:entier ; nomfichier : chaine ; element : typeEnreg)

```
Var
Trouv : booleen ;       i,j : entier
Debut
        ouvrir(f,nomfichier,'w')
        Recherche( c , nomfichier , trouv , i , j )
        si (non trouv) alors // i = dernier bloc et j = dernier element dans le bloc
```

```
            liredir(f,i,buf)
            si(j<b) alors
                  buf.tab[j+1] = element
            sinon
                  i = i + 1
                  affecte_entete(f,1,i)
                  buf.tab[1] = element
                  buf.NB = 1
                  ecriredir(f,i,buf)
            fin si
            n := entete(f,2)
            affecte_entete(f,2,n+1)

      sinon
            ecrire ("L'élément existe déjà ! " )

      fermer(f)
Fin
```

Suppression Logique c'est la même dans type TOF

suppression_physique(c : typeqlq ; nomfichier : chaine)

```
Var
      Trouv : booleen ;       i,j : entier
Debut
      Rech(c,nomfichier,trouv,i,j)
      Si (trouv) alors
            Ouvrir(f,nomfichier,'r+')
            Liredir(f,i,buf)
```

```
            si ( i == entete(f,1) et j == buf.NB) alors
                buf.tab[j] := #
                buf.NB := buf.NB -1
                si (buf.NB == 0 ) alors
                    affecte_entete(f,1,i-1)
                sinon
                    ecriredir(f,i,buf)
                finsi

            sinon
                i1 = entete(f,1)
                Liredir(f,i1,buf1)
                j1 := buf1.NB
                buf.tab[j] := buf1.tab[j1]
                ecriredir(f,i,buf)
                buf1.tab[j1] := #
                buf1.NB := buf1.NB -1
                si (buf1.NB == 0 ) alors
                    affecte_entete(f,1,i1-1)
                sinon
                    ecriredir(f,i1,buf1)
                finsi
            finsi
            n := entete(f,2)
            affecte_entete(f,2,n-1)
        sinon
            Ecrire (" L' élément n'existe pas ! ")
Fermer(f)
```

Fin

articles : structure

tab : tableau[1..100]de typeEnreg

nb : entier

fin structure

Procédure Requete(a:entier, b:entier , nomfichier : chaine ; var articles : articles)

Var

i :=1

Debut

ouvrir(f,nomfichier,'r')

TQ (i<=entete(f,1)) faire

liredir(f,I,buf)

j := 1 ; k := 1

TQ (j<=buf.NB) faire

si (buf.tab[j].cle >= a et buf.tab[j].cle <=b) alors

articles.tab[k] := buf.tab[j]

k := k + 1

articles.nb := articles.nb + 1

sinon

j := j + 1

fin TQ

i := i + 1

```
        fin TQ
fermer(f)
Fin
```

Type LNOF

```
type tbloc : structure
        tab : tableau [1..b] de tenreg
        NB : entier
        suivant : entier
fin structure
type tenreg : structure
        cle : entier
        efface : booleen
        champ1 :typeQLq  …
fin structure
f = fichier de tbloc buffer buf ENTETE (entier , entier)
                // 1 car = nombre des blocs et  2 car = nombre des éléments dans  le fichier
```

Procédure Recherche(c:entier ; nomfichier : chaine ; var trouv :booleen , i , j)

```
Var
i := 1 , trouv := faux ; tete : pointeur sur tbloc
Debut
        ouvrir(f,nomfichier,'r')
        TQ (i <= entete(f,1) et trouv == faux) faire
                liredir(f,i,buf)
                j := 1
                TQ ( j<=buf.NB et trouv == faux) faire
```

```
                si ( buf.tab[j].cle == c et buf.tab[j].efface == faux) alors
                        trouv := vrai
                sinon
                        j := j + 1
            fin TQ
            i := buf.suivant
        fin TQ
        si ( trouv := faux ) alors
            i := entete(f,1)                    //i c'est le dernier bloc
            j := buf.NB                 // j c'est le dernier élément dans le dernier bloc
        fin si
        fermer(f)
fin
```

insertion(c:entier ; nomfichier : chaine ; element : typeEnreg)

```
Var
        Trouv : booleen ;          i,j,k : entier;          x : tenreg
Debut
        ouvrir(f,nomfichier,'r+')
        Recherche( c , nomfichier , trouv ,  i , j)
        si (non trouv) alors                // i= dernier bloc et j = dernier element dans le bloc
            liredir(f,i,buf)
            si(j<b) alors
                buf.tab[j+1] = element
                ecrirebloc(f,i,buf)
            sinon
                i := buf.suivant
                affecte_entete(f,1,i)
                allouer(nouveau)
```

```
                    nouveau.tab[1] := element
                    nouveau.NB := 1
                    nouveau.suivant := -1
                    ecrirebloc(f,i, nouveau)
            fin si
            n := entete(f,2)
            affecte_entete(f,2,n+1)
        sinon
            ecrire ("L'élément existe déjà! " )

        fermer(f)
Fin
```

Suppression Logique c'est la même dans type TOF

suppression_physique(c : typeqlq ; nomfichier : chaine)

```
Var
Trouv : booleen ;       i,j : entier
Debut
        Rech(c,nomfichier,trouv,i,j)
        Si (trouv) alors
            Ouvrir(f,nomfichier,'r+')
            liredir(f,i,buf)
            si ( i == entete(f,1) et j == buf.NB) alors
            // supprimer le dernier element dans le dernier bloc
                buf.tab[j] := #
                buf.NB := buf.NB -1
                si (buf.NB == 0 ) alors
                    i := entete(f,1)  - 1
                    lirebloc(f,i,buf)
```

```
                    buf.suiv := -1
                    ecrirebloc(f,i,buf)
                    affecte_entete(f,1,i)
                sinon
                    ecriredir(f,i,buf)
                finsi

        sinon
                i1 = entete(f,1)
                liredir(f,i1,buf1)
                j1 := buf1.NB
                buf.tab[j] := buf1.tab[j1]
                ecriredir(f,i,buf)
                buf1.tab[j1] := #
                buf1.NB := buf1.NB -1
                si (buf1.NB == 0 ) alors
                    i := entete(f,1)  - 1
                    lirebloc(f,i,buf)
                    buf.suiv := -1
                    ecrirebloc(f,i,buf)
                    affecte_entete(f,1,i)
                sinon
                    ecriredir(f,i1,buf1)
                finsi
        finsi
n := entete(f,2)
affecte_entete(f,2,n-1)
sinon
        Ecrire (" L' élément n'existe pas ! ")
```

```
Fermer(f)

Fin

articles : structure
        tab : tableau[1..100]de typeEnreg
        nb : entier
fin structure
```

Procédure Requete(a:entier, b:entier , nomfichier : chaine ; var articles : articles)

```
Var
   i,j,k : entier
Debut
        ouvrir(f,nomfichier,'r')
        i :=1
        TQ ( i <= entete(f,1) ) faire
                liredir(f,i,buf)
                j := 1 ; k := 1
                TQ ( j<=buf.NB) faire
                        si ( buf.tab[j].cle >= a et buf.tab[j].cle <=b ) alors
                                articles.tab[k] := buf.tab[j]
                                k := k + 1
                                articles.nb := articles.nb + 1
                        finSi
                        j := j + 1
                fin TQ
                i := buf.suivant
```

```
        fin TQ
fermer(f)
Fin
```

Type LOF

```
type tbloc : structure
        tab : tableau [1..b] de tenreg
        NB : entier
        suivant : entier
        prec : entier
fin structure
type tenreg : structure
        cle : entier
        efface : booleen
        champ1 :typeQLq  …
fin structure
f = fichier de tbloc buffer buf ENTETE (entier , entier)
                // 1 car = nombre des blocs et  2 car = nombre des éléments dans  le fichier
```

```
Rech( c:typeqlq, var Trouv:bool, var i,j:entier )
var
   bi, bs, inf, sup : entier
   trouv, stop : booleen

DEBUT
        /* on suppose que le fichier est déjà ouvert */
        bs ← entete( F,1 )          // la borne sup (le num du dernier bloc de F)
        bi ← 1                      // la borne inf  (le num du premier bloc de F)

        // boucle pour la recherche dichotomique externe (dans le fichier F)
        Trouv ← faux;   stop ← faux;   j ← 1
        TQ  ( bi ≤ bs  et  Non Trouv et Non stop )
          i ← (bi + bs) div 2       // le bloc du milieu entre bi et bs
          LireDir( F, i, buf )
          SI ( c ≥ buf.tab[1].cle  et  c ≤ buf.tab[ buf.NB ].cle )
                // boucle pour la recherche dichotomique interne dans le bloc i (dans buf)
                inf ← 1;  sup ← buf.NB
                TQ  (inf ≤ sup  et  Non Trouv)              // recherche interne
                        j ← (inf + sup) div 2
                        SI  (c  = buf.tab[ j ].cle)  Trouv ← vrai
                        SINON
                                SI (c  < buf.tab[ j ].cle)    sup ← j-1
                                SINON                         inf ← j+1
                                FSI
                        FSI
                FTQ
                SI ( inf > sup )  j ← inf  FSI              // fin de la recherche interne.
                                                            // j : la position où devrait se trouver c dans buf.tab
                stop ← vrai
          SINON  // non ( c ≥ buf.tab[1].cle  et  c ≤ buf.tab[ buf.NB ].cle )
                SI ( c < buf.tab[1].cle )
                        bs ← buf.prec
                SINON  // donc c > buf.tab[ buf.NB ].cle
                        bi ← buf.suivant
                FSI
          FSI
        FTQ
        SI ( bi > bs )  i ← bi ;   j ← 1  FSI               // fin de la recherche externe.
                                                            // i : num du bloc où devrait se trouver c
FIN // Recherche
```

insertion(c:entier ; nomfichier : chaine ; element : typeEnreg)

Var

Trouv : booleen ; i,j,k : entier; x : tenreg

Debut

Recherche(c , nomfichier , trouv , i , j)

si (non trouv) alors

```
        ouvrir(f,nomfichier,'r+')
        liredir(f,i,buf)
        x := buf.tab[buf.NB]
        k := buf.NB
        TQ (k > j) faire
                Buf.tab[k] := buf.tab[k-1]
                k:= k – 1
        FinTQ
        Buf.tab[j] := element
        si(buf.NB<b) alors
                buf.NB := buf.NB + 1
                buf.tab[Buf.NB] = element
                ecrirebloc(f,i,buf)
        sinon
                allouer(nouveau)
                nouveau.tab[1] := element
                nouveau.NB := 1
                buf.suiv := nouveau
                i := i + 1
                si (i<= entete(f,1)) alors
                        lirebloc(f,i,buf1)
                        nouveau.suivant := buf1
                sinon
                        nouveau.suivant := -1
                affecte_entete(f,1,i)
        fin si
        n := entete(f,2)
        affecte_entete(f,2,n+1)
sinon
```

```
            ecrire ("L'élément existe déjà ! " )

        fermer(f)
Fin
```

Suppression Logique c'est la même dans type TOF

suppression_physique(c : typeqlq ; nomfichier : chaine)

```
Var
        Trouv : booleen ;          i,j,k : entier;
Debut
        Rech(c,nomfichier,trouv,i,j)
        Si (trouv) alors
                Ouvrir(f,nomfichier,'r+')
                liredir(f,i,buf)
                k := j
                Si ( j != buf.NB) alors
                        Tq (k < buf.NB) faire
                                buf.tab[k] := buf.tab[k+1]
                                k := k +1
                        fin tq
                        (1)  : i := buf.suivant
                        si ( i < entete(f,1)) alors
                                buf1 := lire(f,i)
                                buf.tab[buf.NB] := buf1.tab[1]
                        fin si
                        ecriredir(f,i-1,buf)
                        buf := buf1
                        Tq ( i < entete(f,1)) faire
```

```
                    k := 1
                    Tq (k < buf.NB) faire
                        buf.tab[k] := buf.tab[k+1]
                        k := k +1
                    fin tq
                    i := buf.suivant
                    si ( i < entete(f,1)) alors
                        liredir(f,i,buf1)
                        buf.tab[buf.NB] := buf1.tab[1]
                    fin si
                    ecriredir(f,i-1,buf)
                    buf := buf1
                fin tq
            sinon
                aller a (1)
            fin si
 (buf.NB == 1 )        // si le dernier bloc contient un seul élément donc lorsque en faire le décalage le dernier bloc vas supprimer
            nb := entete(f,1) - 1
            affecte_entete(f,1,nb)
            lirebloc(f,nb,buf)
            buf.suivant := -1
            ecriredir(f,i-1,buf)

finSI

            n := entete(f,2)
            affecte_entete(f,2,n-1)
sinon
            Ecrire (" L' élément n'existe pas ! ")
```

```
Fermer(f)

Fin

articles : structure
        tab : tableau[1..100]de typeEnreg
        nb : entier
fin structure
```

Procédure Requete(a:entier, b:entier , nomfichier : chaine ; var articles : articles)

```
Var
Trouv : booleen ;       i,j : entier
Debut
        Rech(a,nomfichier,trouv,i,j)
        k := 1
        articles.nb := 0
        si (trouv) alors
                Ouvrir(f,nomfichier,'r+')
                tantque(i<=entete(f,1))faire
                        Liredir(f,i,buf)
                        tantque(j<=buf.NB)faire
                                si( buf.tab[j].cle <= b )
                                        articles.tab[k] := buf.tab[j]
                                        k := k + 1
                                        j := j + 1
                                        articles.nb := articles.nb + 1
                                sinon
                                        aller a (2)
```

```
                    fin tq
                    j := 1
                    i := buf.suivant
                fin tq

        sinon
                Ecrire("L' élément n'existe pas!")
        finsi
 (2) fermer(f)
Fin
```

Type LnoVc

```
type tbloc : structure
        tab : tableau [1..b] de caractères
        NB : entier
        suivant : entier
fin structure
f = fichier de tbloc buffer buf ENTETE(entier , entier ,entier ,entier)
// 1 car = num du premier bloc
 2 car = num de dernier bloc
3 car =la 1ere position libre dans le dernier bloc
4 car le nombre de caractères perdus suite aux suppressions logiques
```

l'enregistrement peut avoir le forme suivante :

bbb | b | bbbbbbbbbbbbbbbbbbb |bbbb…….b

3 caractère pour la taille de l'enregistrement

1 caractère pour indiquer l'effacement logique

20 caractère pour la clé de l'enreg

N car pour les autres informations

Procédure Recherche(c:entier ; nomfichier : chaine ; var trouv :booleen , i , j)

```
Var
i , j : entier ; trouv := booléen
Debut
        ouvrir(f,nomfichier,'r')
        i := 1
        j := 1
        trouv := faux
        liredir(f,i,buf)
        TQ (non trouv) et (i <= entete(f,2) ou i == entete(f,2) et j != entete(f,3)) faire
                Recuperer_chaine(3,i,j,longeur) ; Recuperer_chaine(1,i,j,efface)
                Recuperer_chaine(20,i,j,cle_rech)
                Si (cle_rech == c) et (efface == 'F') alors
                        Trouv = vrai
                Sinon
                         j := j + nombre( longeur)-24
                        si ( j > b ) alors
                                j := j – b
                                i := buf.suivant
                                liredir(f,i,buf)
                        finSI
                finSi
        finTQ
        fermer(f)
fin
```

Procédure recuperer_chaine(n :entier : Var i,j :entier ; Var ch : chaine)

```
Var k :entier
Debut
Pour (k := 1 ,n ) faire
        Si (j <= b ) alors
                Ch[k] := buf.tab[j]
                j := j + 1
        sinon           //chevauchement
                i := buf.suivant
                liredir (f,i,buf)
                ch[k] := buf.tab[1]
                j := 2
        finsi
finPour
Fin
```

insertion(c:chaine[20] ; nomfichier : chaine; element : typeEnreg)

```
Var
        Trouv : booleen ;          i,j,k : entier
Debut
        Recherche( c , nomfichier , trouv ,  i , j)
        Si (trouv == faux ) alors
                ouvrir(f,nomfichier,'A')
                i := entete (f,2)
                j := entete(f,3)
                chaine_longueur := element[1 :3]
                //convertir la longueur en chaine de 3 car
                Ecrire_chaine(3,chaine_longueur,i,j)
                Ecrire_chaine(1,'F',i,j)
                chaine_longueur := entier(chaine_longueur )
                Ecrire_chaine(chaine_longueur- 4 ,element[4 : ],i,j)
```

```
                buf.suivant := -1
                ecriredir(f,i,buf)
                si ( i != entete(f,2)) alors
                        affecte_entete(f,2,i)
                finSI
                affecte_entete(f,3,j)
        sinon
                ecrire ("L'élément existe déjà ! " )

        finSi
        fermer(f)
Fin
```

Procédure Ecrire_chaine (n :entier , ch : chaine , Var i,j : entier)

Var

K : entier ; i1 : entier

Debut

```
        Pour ( k := 1 , n ) faire
                Si ( j<=b ) alors
                        Buf.tab[j] := ch[k]
                        j := j + 1
                sinon
                        i1 : allocbloc(f)
                        buf.suivant := i1
                        ecriredir(f,i,buf)
                        i := i1
```

```
                    buf.tab [1] := ch[k]
                    j := 2
                finSI
        finPour
```

Fin

Procédure Suppression_logique(c : chaine[20] , nomfichier :chaine)

Var

Trouv : booleen ; i,j : entier

Debut

```
        Ouvrir(f,nomfichier,'A')
        Recherche(c,nomfichier,trouv,i,j)
        Si (trouv ) alors
                j := j + 3
                buf.tab[j] := ' V '
                ecriredir(f,i,j)

        Sinon
                Ecrire ( ''L'élément n'existe pas ! '')
        finSI
```

Fin

```
articles : structure
        tab : tableau[tailleMax]de caractères
        nb : entier                //nombre des enreg
fin structure
```

Procédure Requete(a:entier, b:entier , nomfichier : chaine ; var articles : articles)

```
Var
   i,j,k : entier
Debut
        ouvrir(f,nomfichier,'r')
        i :=1
        TQ ( i < entete(f,2) ou i == entete(f,2) et j != entete(f,3)) faire
                liredir(f,i,buf)
                j := 1 ; k := 1
                Recuperer_chaine(3,i,j,longeur) ; Recuperer_chaine(1,i,j,efface)
                Recuperer_chaine(20,i,j,cle_rech)
                cle_rech := entier (cle_rech)          //convertir
                longueur := entier (longueur)          //convertir
                j := j – 24
                Si (cle_rech >= a et cle_rech <= b) et (efface == 'F') alors
                        Pour ( n := 1 , longueur ) faire
                                articles.tab[k] := buf.tab[j]
                                k := k + 1
                                j := j + 1
                        finPour
                        articles.nb := articles.nb + 1

                Sinon
                         j := j + nombre( longeur)
                        si ( j > b ) alors
                                j := j – b
                                i := buf.suivant
                        finSI
```

finSi

finTQ

fermer(f)

Fin

Type LnoVnc

type tbloc : structure

tab : tableau [1..b] de caractères

NB : entier

suivant : entier

fin structure

f = fichier de tbloc buffer buf ENTETE(entier , entier ,entier ,entier)
// 1 car = num du premier bloc

2 car = num de dernier bloc

3 car =la 1ere position libre dans le dernier bloc

4 car le nombre de caractères perdus suite aux suppressions logiques

l'enregistrement peut avoir le forme suivante :

bbb | b | bbbbbbbbbbbbbbbbbbbb |bbbb…….b

3 car pour la taille de l'enregistrement

1 car pour indiquer l'effacement logique

20 car pour la clé de l'enreg

N car pour les autres informations

Procédure Recherche(c:entier ; nomfichier : chaine ; var trouv :booleen , i , j)

Var

i , j : entier ; trouv := booléen

```
Debut
        ouvrir(f,nomfichier,'r')
        i := 1
        j := 1
        trouv := faux
        liredir(f,i,buf)
        TQ (non trouv) et (i <= entete(f,2) ou i == entete(f,2) et j != entete(f,3)) faire
                Recuperer_chaine(3,i,j,longeur) ; Recuperer_chaine(1,i,j,efface)
                Recuperer_chaine(20,i,j,cle_rech)
                Si (cle_rech == c) et (efface == 'F') alors
                        Trouv = vrai
                Sinon
                         j := j + nombre( longeur)-24
                        si ( j > b ) alors
                                j := 1
                                i := buf.suivant
                                liredir(f,i,buf)
                        finSI
                finSi
        finTQ
        fermer(f)
fin
```

Procédure recuperer_chaine(n :entier : Var i,j :entier ; Var ch : chaine)

Var k :entier

Debut

```
Pour (k := 1 ,n ) faire
        Si (j <= b ) alors
                Ch[k] := buf.tab[j]
                j := j + 1
```

```
        finsi
finPour
Fin
insertion( c:chaine[20] ; nomfichier : chaine; element : typeEnreg )
Var
        Trouv : booleen ;          i,j,k : entier
Debut
        Recherche( c , nomfichier , trouv ,  i , j)
        Si (trouv == faux ) alors
                ouvrir(f,nomfichier,'A')
                i := entete (f,2)
                j := entete(f,3)
                chaine_longueur := element[1 :3]
                //convertir la longueur en chaine de 3 car
                SI (j +entier(chaine_longueur) <= b ) alors
                        Ecrire_chaine(3,chaine_longueur,i,j)
                        Ecrire_chaine(1,'F',i,j)
                        chaine_longueur := entier(chaine_longueur)
                        Ecrire_chaine(chaine_longueur-4 ,element[4 : ],i,j)
                        buf.suivant := -1
                        ecriredir(f,i,buf)
                Sinon
                        i := buf.suivant
                        j := 1
                        Ecrire_chaine(3,chaine_longueur,i,j)
                        Ecrire_chaine(1,'F',i,j)
                        chaine_longueur := entier(chaine_longueur)
                        Ecrire_chaine(chaine_longueur-4 ,element[4 : ],i,j)
                        buf.suivant := -1
```

```
                ecriredir(f,i,buf)
        finSI

        si ( i != entete(f,2)) alors
                affecte_entete(f,2,i)
        finSI
        affecte_entete(f,3,j)
    sinon
        ecrire ("L'élément existe déjà ! " )

    finSi
    fermer(f)
Fin
```

```
Procédure Ecrire_chaine  (n :entier , ch : chaine , Var i,j : entier )
Var
K : entier ; i1 : entier
Debut
    Pour ( k := 1 , n ) faire
        Si ( j<=b ) alors
            Buf.tab[j] := ch[k]
            j := j + 1
        finSI
    finPour

Fin
```

```
Procédure Suppression_logique(c : chaine[20] , nomfichier :chaine)
Var
        Trouv : booleen ; i,j : entier
Debut
        Ouvrir(f,nomfichier,'A')
        Recherche(c,nomfichier,trouv,i,j)
        Si (trouv ) alors
                j := j + 3
                buf.tab[j] := ' V '
                ecriredir(f,i,j)

        Sinon
                Ecrire ( ''L'élément n'existe pas ! '')
        finSI
Fin
articles : structure
        tab : tableau[tailleMax]de caractères
        nb : entier                //nombre des enreg
fin structure

Procédure Requete( a:entier, b:entier , nomfichier : chaine ; var  articles : articles)
Var
   i,j,k : entier
Debut
        ouvrir(f,nomfichier,'r')
        i :=1
        TQ ( i < entete(f,2) ou i == entete(f,2) et j != entete(f,3)) faire
```

```
            liredir(f,i,buf)
            j := 1 ; k := 1
            Recuperer_chaine(3,i,j,longeur) ; Recuperer_chaine(1,i,j,efface)
            Recuperer_chaine(20,i,j,cle_rech)
            cle_rech := entier (cle_rech)          //convertir
            longueur := entier (longueur)          //convertir
            j := j – 24
            Si (cle_rech >= a et cle_rech <= b) et (efface == 'F') alors
                  Pour ( n := 1 , longueur ) faire
                        articles.tab[k] := buf.tab[j]
                        k := k + 1
                        j := j + 1
                  finPour
                  articles.nb := articles.nb + 1

            Sinon
                  j := j + nombre( longeur)-24
                  si ( j > b ) alors
                        j := 1
                        i := buf.suivant
                  finSI
            finSi
      finTQ

fermer(f)
Fin
```

Type LoVc

type tbloc : structure

```
        tab : tableau [1..b] de caractères
        NB : entier
        suivant : entier
        prec : entier
fin structure
f = fichier de tbloc buffer buf ENTETE(entier , entier ,entier ,entier ,entier)
// 1 car = num du premier bloc
 2 car = num de dernier bloc
3 car = la 1ere position libre dans le dernier bloc
4 car =  le nombre de caractères perdus suite aux suppressions logiques
5 car = position de dernier element dans le bloc
```

l'enregistrement peut avoir le forme suivante :

bbb | b | bbbbbbbbbbbbbbbbbbbb |bbbb…….b

3 car pour la taille de l'enregistrement

1 car pour indiquer l'effacement logique

20 car pour la clé de l'enreg

N car pour les autres informations

Procédure Recherche(c:entier ; nomfichier : chaine ; var trouv :booleen , i , j)

```
Var
bi , bs , inf , sup : entier ;
trouv , stop:= booléen
Debut
        ouvrir(f,nomfichier,'r')
        bs := entete(f,2)
        bi := 1
        trouv := faux ; stop := faux ; j := 1
        TQ ( bi <= bs et Non trouv et Non stop )
```

```
i := (bi + bs ) div 2
liredir(f , i , buf)
j := j + 4
Recuperer_chaine(20,i,j,cle1)
j := entete(f, 5)
j := j + 4
si ( j > b ) alors
        j := j – b
        i := buf.suivant
finSI
Recuperer_chaine(20,i,j,cle2)
cle1 := entier(cle1) ; cle2 := entier(cle2)
Si ( c >= cle1 et c <= cle2 ) alors
        inf := 1 ; sup := nombre_element(i)
        // nombre_element == fonction qui calcule nombre des éléments
dans le bloc
        TQ (inf <= sup et non trouv )
                Milieu(i,inf,sup, k)
                Recuperer_chaine(3,i,k,longueur)
                Recuperer_chaine(1,i,k,efface)
                Recuperer_chaine(20,i,k,cle)
                longueur := entier(longueur)
                cle := entier(cle)
                longueur := entier(longueur)
                Si ( c == cle ) alors
                        Trouv := vrai
                Sinon
                        Si (c < cle ) alors
                                Sup := k -24
                        sinon
```

```
                                        inf := k + longueur – 24
                                    finSI
                                finSI
                        finTQ
                        Si( inf > sup) alors
                                j := inf
                        finSI
                        stop := vrai
        Sinon
                Si (c < cle1 )
                        bs := buf.prec
                sinon
                        bi := buf.suivant
                finSI
        finSI
        finTQ
        SI (bi > bs ) alors
                i := bi
                j := 1
        finSi
fermer(f)
Fin
```

Procédure recuperer_chaine(n :entier : Var i,j :entier ; Var ch : chaine)

Var k :entier

Debut

```
Pour (k := 1 ,n ) faire
        Si (j <= b ) alors
                Ch[k] := buf.tab[j]
```

```
            j := j + 1
        sinon            //chevauchement
            i := buf.suivant
            liredir (f,i,buf)
            ch[k] := buf.tab[1]
            j := 2
        finsi
finPour
```

Fin

Procédure milieu (i : entier , j : entier , sup : entier ; var k : entier)

Var

Debut

```
        nenreg := nombre_element(i,j,sup)
        nenreg := (nenreg) div 2
        cpt := 0
        TQ (cpt < nenreg) faire
            Recuperer_chaine(3,bloc,j,longueur)
            longueur := entier(longueur)
            j := j + longueur – 3
            cpt := cpt + 1
        finTQ
```

k := j

Fin

Procédure nombre_element (i : entier , j : entier, sup : entier)

Var

J : entier ; cpt : entier

Debut

```
        cpt := 0
        TQ (j < sup) faire
                Recuperer_chaine(3,bloc,j,longueur)
                longueur := entier(longueur)
                j := j + longueur – 3
                cpt := cpt + 1
        finTQ
retourner cpt
```

Fin

insertion(nomfichier : chaine; element : typeEnreg)

```
Var
        Trouv,continu : booleen ;     i,j,k : entier
Debut
        c := element [5 .. 24]
        Recherche( c , nomfichier , trouv ,  i , j)
        Si (trouv == faux ) alors
                ouvrir(f,nomfichier,'A')
I               liredir(f,i,buf)
                L := element [1..3]     // L : taille de l'enreg
                t := taille(buf)  //nombre des caractères dans le bloc
                Si ( L + t < b ) alors              //buffer n'est pas plein
                        K :=0
                        N := j + 1
                        // sauvegarder les elemnts apres la postion j
                        Tantque ( N  < b ) faire
                                Articles [K ] := buf.tab[N]
                                K := K + 1
                                N := N + 1
```

```
        finTQ
        cpt := 1
        tantque(cpt <= L ) faire
                buf.tab[j ] := element[cpt]
                j := j+ 1
                cpt := cpt + 1
        finTQ
        cpt := 1
        tantque(cpt <= k ) faire
                buf.tab[j ] := articles[cpt]
                cpt := cpt + 1
                j := j + 1
        finTQ
Sinon
        K :=0
        N := j + 1
        // sauvegarder les elemnts apres la postion j
        Tantque ( N  < b ) faire
                Articles [K ] := buf.tab[N]
                K := K + 1
                N := N + 1
        finTQ
        cpt := 1
        tantque(cpt <= L ) faire
                buf.tab[j ] := element[cpt]
                j := j+ 1
                cpt := cpt + 1
        finTQ
        cpt := 1
```

```
                tantque(j <=b ) faire
                        buf.tab[j ] := articles[cpt]
                        cpt := cpt + 1
                        j := j+ 1
                finTQ
                i := buf .suiv

        finSi

    sinon
        ecrire ("L'élément existe déjà ! " )

    finSi
    fermer(f)
Fin
```

Procédure Ecrire_chaine (n :entier , ch : chaine , Var i,j : entier)

Var

K : entier ; i1 : entier

Debut

```
    Pour ( k := 1 , n ) faire
        Si ( j<=b ) alors
            Buf.tab[j] := ch[k]
            j := j + 1
        sinon
            i1 : allocbloc(f)
```

```
                buf.suivant := i1
                ecriredir(f,i,buf)
                i := i1
                buf.tab [1] := ch[k]
                j := 2
        finSI
finPour
```

Fin

Procédure Suppression_logique(c : chaine[20] , nomfichier :chaine)

Var

Trouv : booleen ; i,j : entier

Debut

```
Ouvrir(f,nomfichier,'A')
Recherche(c,nomfichier,trouv,i,j)
Si (trouv ) alors
        j := j + 3
        si ( j > b ) alors
                j := j – b
                i := buf.suivant
        finSI
        buf.tab[j] := ' V '
        ecriredir(f,i,j)

Sinon
        Ecrire ( ''L'élément n'existe pas ! '')
finSI
```

Fin

```
articles : structure
        tab : tableau[tailleMax]de caractères
        nb : entier              //nombre des enreg
fin structure
```

Procédure Requete(a:entier, b:entier , nomfichier : chaine ; var articles : articles)

```
Var
  i,j,k : entier
Debut
        ouvrir(f,nomfichier,'r')
        Rech(a,nomfichier,trouv,i,j)
        TQ ( i < entete(f,2) ou i == entete(f,2) et j != entete(f,3)) faire
                liredir(f,i,buf)
                 k := 1
                Recuperer_chaine(3,i,j,longeur) ; Recuperer_chaine(1,i,j,efface)
                Recuperer_chaine(20,i,j,cle_rech)
                cle_rech := entier (cle_rech)          //convertir
                longueur := entier (longueur)          //convertir
                j := j – 24
                Si (cle_rech >= a et cle_rech <= b) et (efface == 'F') alors
                        Pour ( n := 1 , longueur ) faire
                                articles.tab[k] := buf.tab[j]
                                k := k + 1
                                j := j + 1
                        finPour
                        articles.nb := articles.nb + 1
```

```
            Sinon
                    j := j + nombre( longeur)
                    si ( j > b ) alors
                            j := j – b
                            i := buf.suivant
                    finSI
            finSi
    finTQ

fermer(f)
Fin
```

Type LoVnc

```
type tbloc : structure
        tab : tableau [1..b] de caractères
        NB : entier
        suivant : entier
        prec : entier
fin structure
f = fichier de tbloc buffer buf ENTETE(entier , entier ,entier ,entier ,entier)
// 1 car = num du premier bloc
 2 car = num de dernier bloc
3 car = la 1ere position libre dans le dernier bloc
4 car =  le nombre de caractères perdus suite aux suppressions logiques
5 car = position de dernier element dans le bloc
```

l'enregistrement peut avoir le forme suivante :

bbb | b | bbbbbbbbbbbbbbbbbbbb |bbbb…….b

3 car pour la taille de l'enregistrement

1 car pour indiquer l'effacement logique

20 car pour la clé de l'enreg

N car pour les autres informations

Procédure Recherche(c:entier ; nomfichier : chaine ; var trouv :booleen , i , j)

```
Var
bi , bs , inf , sup : entier ;
trouv , stop:= booléen
Debut
        ouvrir(f,nomfichier,'r')
        bs := entete(f,2)
        bi := 1
        trouv := faux ; stop := faux ; j := 1
        TQ ( bi <= bs et Non trouv et Non stop )
                i := (bi + bs ) div 2
                liredir(f , i , buf)
                j := j + 4
                Recuperer_chaine(20,i,j,cle1)
                j := entete(f, 5)
                Recuperer_chaine(20,i,j,cle2)
                cle1 := entier(cle1) ; cle2 := entier(cle2)
                Si ( c >= cle1 et c <= cle2 ) alors
                        inf := 1 ; sup := nombre_element(i)
                        // nombre_element == fonction qui calcule nombre des éléments
                dans le bloc
                        TQ (inf <= sup et non trouv )
                                Milieu(i,inf,sup, k)
                                Recuperer_chaine(3,i,k,longueur)
                                Recuperer_chaine(1,i,k,efface)
```

```
                    Recuperer_chaine(20,i,k,cle)
                    longueur := entier(longueur)
                    cle := entier(cle)
                    longueur := entier(longueur)
                    Si ( c == cle ) alors
                        Trouv := vrai
                    Sinon
                        Si (c < cle ) alors
                            Sup := k -24
                        sinon
                            inf := k + longueur – 24
                        finSI
                    finSI
                finTQ
                Si( inf > sup) alors
                    j := inf
                finSI
                stop := vrai
Sinon
        Si (c < cle1 )
                bs := buf.prec
        sinon
                bi := buf.suivant
        finSI
finSI
finTQ
SI (bi > bs ) alors
        i := bi
        j := 1
```

```
        finSi
fermer(f)
Fin
```

```
Procédure recuperer_chaine(n :entier : Var i,j :entier ; Var ch : chaine )
Var k :entier
Debut
Pour (k := 1 ,n ) faire
        Si (j <= b ) alors
                Ch[k] := buf.tab[j]
                j := j + 1
        finsi
finPour
Fin
Procédure milieu ( i : entier , j : entier , sup : entier ; var k : entier )
Var
Debut
        nenreg := nombre_element(i,j,sup)
        nenreg := (nenreg) div 2
        cpt := 0
        TQ (cpt < nenreg) faire
                Recuperer_chaine(3,bloc,j,longueur)
                longueur := entier(longueur)
                j := j + longueur – 3
                cpt := cpt + 1
        finTQ
        k := j
Fin
```

```
Procédure nombre_element ( i : entier , j : entier, sup : entier)
Var
J : entier ; cpt : entier
Debut
        cpt := 0
        TQ (j < sup) faire
                Recuperer_chaine(3,bloc,j,longueur)
                longueur := entier(longueur)
                j := j + longueur – 3
                cpt := cpt + 1
        finTQ
retourner cpt
Fin

insertion(nomfichier : chaine; element : typeEnreg )
Var
         Trouv,continu : booleen ;      i,j,k : entier
Debut
        c := element [5 .. 24]
        Recherche( c , nomfichier , trouv ,  i , j)
        Si (trouv == faux ) alors
                ouvrir(f,nomfichier,'A')
l               liredir(f,i,buf)
                 L := element [1..3]     // L : taille de l'enreg
                t := taille(buf)  //nombre des caractères dans le bloc
                Si ( L + t < b ) alors                //buffer n'est pas plein
                        K :=0
```

```
        N := j + 1
        // sauvegarder les elemnts apres la postion j
        Tantque ( N  < b ) faire
                Articles [K ] := buf.tab[N]
                K := K + 1
                N := N + 1
        finTQ
        cpt := 1
        tantque(cpt <= L ) faire
                buf.tab[j ] := element[cpt]
                j := j+ 1
                cpt := cpt + 1
        finTQ
        cpt := 1
        tantque(cpt <= k ) faire
                buf.tab[j ] := articles[cpt]
                cpt := cpt + 1
                j := j + 1
        finTQ
Sinon
        K :=0
        N := j + 1
        // sauvegarder les elemnts apres la postion j
        Tantque ( N  < b ) faire
                Articles [K ] := buf.tab[N]
                K := K + 1
                N := N + 1
        finTQ
        cpt := 1
```

```
                    tantque(cpt <= L ) faire
                        buf.tab[j ] := element[cpt]
                        j := j+ 1
                        cpt := cpt + 1
                    finTQ
                    cpt := 1
                    tantque(j <=b ) faire
                        buf.tab[j ] := articles[cpt]
                        cpt := cpt + 1
                        j := j+ 1
                    finTQ
                    i := buf .suiv

                finSi

        sinon
                ecrire ("L'élément existe déjà ! " )

        finSi
        fermer(f)
Fin
```

Procédure Ecrire_chaine (n :entier , ch : chaine , Var i,j : entier)

Var

K : entier ; i1 : entier

Debut

```
Pour ( k := 1 , n ) faire
        Si ( j<=b ) alors
                Buf.tab[j] := ch[k]
                j := j + 1
        finSI
finPour
```

Fin

Procédure Suppression_logique(c : chaine[20] , nomfichier :chaine)

Var

Trouv : booleen ; i,j : entier

Debut

```
Ouvrir(f,nomfichier,'A')
Recherche(c,nomfichier,trouv,i,j)
Si (trouv ) alors
        j := j + 3
        si ( j < b ) alors
         buf.tab[j] := ' V '
        ecriredir(f,i,j)
        finSI

Sinon
        Ecrire ( ''L'élément n'existe pas ! '')
finSI
```

Fin

```
articles : structure
        tab : tableau[tailleMax]de caractères
```

```
        nb : entier                //nombre des enreg
fin structure
```

Procédure Requete(a:entier, b:entier , nomfichier : chaine ; var articles : articles)

```
Var
  i,j,k : entier
Debut
        ouvrir(f,nomfichier,'r')
        Rech(a,nomfichier,trouv,i,j)
        TQ ( i < entete(f,2) ou i == entete(f,2) et j != entete(f,3)) faire
                liredir(f,i,buf)
                 k := 1
                Recuperer_chaine(3,i,j,longeur) ; Recuperer_chaine(1,i,j,efface)
                Recuperer_chaine(20,i,j,cle_rech)
                cle_rech := entier (cle_rech)          //convertir
                longueur := entier (longueur)          //convertir
                j := j – 24
                Si (cle_rech >= a et cle_rech <= b) et (efface == 'F') alors
                        Pour ( n := 1 , longueur ) faire
                                articles.tab[k] := buf.tab[j]
                                k := k + 1
                                j := j + 1
                        finPour
                        articles.nb := articles.nb + 1

                Sinon
                         j := j + nombre( longeur)
                        si ( j > b ) alors
```

```
                    j := 1
                    i := buf.suivant
                finSI
            finSi
        finTQ

fermer(f)
Fin
```

Type TnoVc

```
type tbloc : structure
        tab : tableau [1..b] de caractères
        NB : entier
fin structure
f = fichier de tbloc buffer buf ENTETE(entier , entier ,entier ,entier)
// 1 car = num du premier bloc
 2 car = num de dernier bloc
3 car =la 1ere position libre dans le dernier bloc
4 car le nombre de caractères perdus suite aux suppressions logiques
```

l'enregistrement peut avoir le forme suivante :

bbb | b | bbbbbbbbbbbbbbbbbbb |bbbb…….b

3 car pour la taille de l'enregistrement

1 car pour indiquer l'effacement logique

20 car pour la clé de l'enreg

N car pour les autres informations

```
Procédure Recherche( c:entier ; nomfichier : chaine ; var trouv :booleen , i , j)
Var
```

```
i , j : entier ; trouv := booléen
Debut
        ouvrir(f,nomfichier,'r')
        i := 1
        j := 1
        trouv := faux
        liredir(f,i,buf)
        TQ (non trouv) et (i <= entete(f,2) ou i == entete(f,2) et j != entete(f,3)) faire
                Recuperer_chaine(3,i,j,longeur) ; Recuperer_chaine(1,i,j,efface)
                Recuperer_chaine(20,i,j,cle_rech)
                Si (cle_rech == c) et (efface == 'F') alors
                        Trouv = vrai
                Sinon
                         j := j + nombre( longeur)-24
                        si ( j > b ) alors
                                j := j – b
                                i := i + 1
                                liredir(f,i,buf)
                        finSI
                finSi
        finTQ
        fermer(f)
fin
```

```
Procédure recuperer_chaine(n :entier : Var i,j :entier ; Var ch : chaine )
Var k :entier
Debut
Pour (k := 1 ,n ) faire
        Si (j <= b ) alors
                Ch[k] := buf.tab[j]
```

```
            j := j + 1
        sinon            //chevauchement
            i := i + 1
            liredir (f,i,buf)
            ch[k] := buf.tab[1]
            j := 2
        finsi
finPour
```

Fin

insertion(c:chaine[20] ; nomfichier : chaine; element : typeEnreg)

```
Var
     Trouv : booleen ;          i,j,k : entier
Debut
        Recherche( c , nomfichier , trouv ,  i , j)
        Si (trouv == faux ) alors
            ouvrir(f,nomfichier,'A')
            i := entete (f,2)
            j := entete(f,3)
            chaine_longueur := element[1 :3]
            //convertir la longueur en chaine de 3 car
            Ecrire_chaine(3,chaine_longueur,i,j)
            Ecrire_chaine(1,'F',i,j)
            chaine_longueur := entier(chaine_longueur )
            Ecrire_chaine(chaine_longueur- 4 ,element[4 : ],i,j)
            ecriredir(f,i,buf)
            si ( i != entete(f,2)) alors
                affecte_entete(f,2,i)
            finSI
            affecte_entete(f,3,j)
```

```
        sinon
                ecrire ("L'élément existe déjà ! " )

        finSi
        fermer(f)
Fin
```

Procédure Ecrire_chaine (n :entier , ch : chaine , Var i,j : entier)

Var

K : entier ; i1 : entier

Debut

```
        Pour ( k := 1 , n ) faire
                Si ( j<=b ) alors
                        Buf.tab[j] := ch[k]
                        j := j + 1
                sinon
                        ecriredir(f,i,buf)
                        i := i + 1
                        buf.tab [1] := ch[k]
                        j := 2
                finSI
        finPour
```

Fin

Procédure Suppression_logique(c : chaine[20] , nomfichier :chaine)

```
Var
        Trouv : booleen ; i,j : entier
Debut
        Ouvrir(f,nomfichier,'A')
        Recherche(c,nomfichier,trouv,i,j)
        Si (trouv ) alors
                j := j + 3
                buf.tab[j] := ' V '
                ecriredir(f,i,j)

        Sinon
                Ecrire ( ''L'élément n'existe pas ! '')
        finSI
Fin
articles : structure
        tab : tableau[tailleMax]de caractères
        nb : entier                //nombre des enreg
fin structure
```

Procédure Requete(a:entier, b:entier , nomfichier : chaine ; var articles : articles)

```
Var
   i,j,k : entier
Debut
        ouvrir(f,nomfichier,'r')
        i :=1
        TQ ( i < entete(f,2) ou i == entete(f,2) et j != entete(f,3)) faire
                liredir(f,i,buf)
                j := 1 ; k := 1
```

```
Recuperer_chaine(3,i,j,longeur) ; Recuperer_chaine(1,i,j,efface)
Recuperer_chaine(20,i,j,cle_rech)
cle_rech := entier (cle_rech)          //convertir
longueur := entier (longueur)          //convertir
j := j – 24
Si (cle_rech >= a et cle_rech <= b) et (efface == 'F') alors
        Pour ( n := 1 , longueur ) faire
                articles.tab[k] := buf.tab[j]
                k := k + 1
                j := j + 1
        finPour
        articles.nb := articles.nb + 1

Sinon
        j := j + nombre( longeur)
        si ( j > b ) alors
                j := j – b
                i := i + 1
        finSI
finSi
finTQ

fermer(f)
Fin
```

Type TnoVnc

```
type tbloc : structure
        tab : tableau [1..b] de caractères
        NB : entier
```

```
        suivant : entier
fin structure
f = fichier de tbloc buffer buf ENTETE(entier , entier ,entier ,entier)
// 1 car = num du premier bloc
 2 car = num de dernier bloc
3 car =la 1ere position libre dans le dernier bloc
4 car le nombre de caractères perdus suite aux suppressions logiques
```

l'enregistrement peut avoir le forme suivante :

bbb | b | bbbbbbbbbbbbbbbbbbbb |bbbb…….b

3 car pour la taille de l'enregistrement

1 car pour indiquer l'effacement logique

20 car pour la clé de l'enreg

N car pour les autres informations

Procédure Recherche(c:entier ; nomfichier : chaine ; var trouv :booleen , i , j)

```
Var
i , j : entier ; trouv := booléen
Debut
        ouvrir(f,nomfichier,'r')
        i := 1
        j := 1
        trouv := faux
        liredir(f,i,buf)
        TQ (non trouv) et (i <= entete(f,2) ou i == entete(f,2) et j != entete(f,3)) faire
                Recuperer_chaine(3,i,j,longeur) ; Recuperer_chaine(1,i,j,efface)
                Recuperer_chaine(20,i,j,cle_rech)
                Si (cle_rech == c) et (efface == 'F') alors
                        Trouv = vrai
```

```
            Sinon
                j := j + nombre( longeur)-24
                si ( j > b ) alors
                    j := 1
                    i := i + 1
                    liredir(f,i,buf)
                finSI
            finSi
        finTQ
        fermer(f)
fin
```

Procédure recuperer_chaine(n :entier : Var i,j :entier ; Var ch : chaine)

Var k :entier

Debut

```
Pour (k := 1 ,n ) faire
        Si (j <= b ) alors
            Ch[k] := buf.tab[j]
            j := j + 1
        finsi
finPour
```

Fin

insertion(c:chaine[20] ; nomfichier : chaine; element : typeEnreg)

```
Var
        Trouv : booleen ;          i,j,k : entier
Debut
        Recherche( c , nomfichier , trouv ,  i , j)
        Si (trouv == faux ) alors
            ouvrir(f,nomfichier,'A')
            i := entete (f,2)
```

```
            j := entete(f,3)
            chaine_longueur := element[1 :3]
            //convertir la longueur en chaine de 3 car
            SI (j +entier(chaine_longueur) <= b ) alors
                  Ecrire_chaine(3,chaine_longueur,i,j)
                  Ecrire_chaine(1,'F',i,j)
                  chaine_longueur := entier(chaine_longueur)
                  Ecrire_chaine(chaine_longueur-4 ,element[4 : ],i,j)
                  ecriredir(f,i,buf)
            Sinon
                  i := i + 1
                  j := 1
                  Ecrire_chaine(3,chaine_longueur,i,j)
                  Ecrire_chaine(1,'F',i,j)
                  chaine_longueur := entier(chaine_longueur)
                  Ecrire_chaine(chaine_longueur-4 ,element[4 : ],i,j)
                  ecriredir(f,i,buf)
            finSI

            si ( i != entete(f,2)) alors
                  affecte_entete(f,2,i)
            finSI
            affecte_entete(f,3,j)
      sinon
            ecrire ("L'élément existe déjà ! " )

      finSi
      fermer(f)
Fin
```

```
Procédure Ecrire_chaine  (n :entier , ch : chaine , Var i,j : entier )
Var
K : entier ; i1 : entier
Debut
        Pour ( k := 1 , n ) faire
                Si ( j<=b ) alors
                        Buf.tab[j] := ch[k]
                        j := j + 1
                finSI
        finPour

Fin

Procédure Suppression_logique(c : chaine[20] , nomfichier :chaine)
Var
        Trouv : booleen ; i,j : entier
Debut
        Ouvrir(f,nomfichier,'A')
        Recherche(c,nomfichier,trouv,i,j)
        Si (trouv ) alors
                j := j + 3
                buf.tab[j] := ' V '
                ecriredir(f,i,j)

        Sinon
```

```
            Ecrire ( ‘’L’élément n’existe pas ! ‘’)
        finSI
```

Fin

```
articles : structure
        tab : tableau[tailleMax]de caractères
        nb : entier                 //nombre des enreg
fin structure
```

Procédure Requete(a:entier, b:entier , nomfichier : chaine ; var articles : articles)

```
Var
   i,j,k : entier
Debut
        ouvrir(f,nomfichier,'r')
        i :=1
        TQ ( i < entete(f,2) ou i == entete(f,2) et j != entete(f,3)) faire
                liredir(f,i,buf)
                j := 1 ; k := 1
                Recuperer_chaine(3,i,j,longeur) ; Recuperer_chaine(1,i,j,efface)
                Recuperer_chaine(20,i,j,cle_rech)
                cle_rech := entier (cle_rech)           //convertir
                longueur := entier (longueur)           //convertir
                j := j – 24
                Si (cle_rech >= a et cle_rech <= b) et (efface == ‘F’) alors
                        Pour ( n := 1 , longueur ) faire
                                articles.tab[k] := buf.tab[j]
                                k := k + 1
                                j := j + 1
                        finPour
```

```
                articles.nb := articles.nb + 1


        Sinon
                j := j + nombre( longeur)-24
                si ( j > b ) alors
                        j := 1
                        i := i + 1
                finSI
        finSi
    finTQ

fermer(f)
Fin
```

Type ToVc

```
type tbloc : structure
        tab : tableau [1..b] de caractères
        NB : entier
        suivant : entier
        prec : entier
fin structure
f = fichier de tbloc buffer buf ENTETE(entier , entier ,entier ,entier ,entier)
// 1 car = num du premier bloc
 2 car = num de dernier bloc
3 car = la 1ere position libre dans le dernier bloc
4 car =  le nombre de caractères perdus suite aux suppressions logiques
5 car = position de dernier element dans le bloc
```

l'enregistrement peut avoir le forme suivante :

bbb | b | bbbbbbbbbbbbbbbbbbb |bbbb…….b

3 car pour la taille de l'enregistrement

1 car pour indiquer l'effacement logique

20 car pour la clé de l'enreg

N car pour les autres informations

Procédure Recherche(c:entier ; nomfichier : chaine ; var trouv :booleen , i , j)

```
Var
bi , bs , inf , sup : entier ;
trouv , stop:= booléen
Debut
        ouvrir(f,nomfichier,'r')
        bs := entete(f,2)
        bi := 1
        trouv := faux ; stop := faux ; j := 1
        TQ ( bi <= bs et Non trouv et Non stop )
                i := (bi + bs ) div 2
                liredir(f , i , buf)
                j := j + 4
                Recuperer_chaine(20,i,j,cle1)
                j := entete(f, 5)
                j := j + 4
                si ( j > b ) alors
                        j := j – b
                        i := i + 1
                finSI
                Recuperer_chaine(20,i,j,cle2)
                cle1 := entier(cle1) ; cle2 := entier(cle2)
                Si ( c >= cle1 et c <= cle2 ) alors
```

```
            inf := 1 ; sup := nombre_element(i)
            // nombre_element == fonction qui calcule nombre des éléments
        dans le bloc
            TQ (inf <= sup et non trouv )
                Milieu(i,inf,sup, k)
                Recuperer_chaine(3,i,k,longueur)
                Recuperer_chaine(1,i,k,efface)
                Recuperer_chaine(20,i,k,cle)
                longueur := entier(longueur)
                cle := entier(cle)
                longueur := entier(longueur)
                Si ( c == cle ) alors
                    Trouv := vrai
                Sinon
                    Si (c < cle ) alors
                        Sup := k -24
                    sinon
                        inf := k + longueur – 24
                    finSI
                finSI
            finTQ
            Si( inf > sup) alors
                j := inf
            finSI
            stop := vrai
    Sinon
        Si (c < cle1 )
            bs := i – 1
        sinon
            bi := i + 1
```

```
                finSI
        finSI
        finTQ
        SI (bi > bs ) alors
                i := bi
                j := 1
        finSi
fermer(f)
Fin
```

Procédure recuperer_chaine(n :entier : Var i,j :entier ; Var ch : chaine)

Var k :entier

Debut

```
Pour (k := 1 ,n ) faire
        Si (j <= b ) alors
                Ch[k] := buf.tab[j]
                j := j + 1
        sinon           //chevauchement
                i := i + 1
                liredir (f,i,buf)
                ch[k] := buf.tab[1]
                j := 2
        finsi
finPour
```

Fin

Procédure milieu (i : entier , j : entier , sup : entier ; var k : entier)

Var

Debut

```
        nenreg := nombre_element(i,j,sup)
        nenreg := (nenreg) div 2
        cpt := 0
        TQ (cpt < nenreg) faire
                Recuperer_chaine(3,bloc,j,longueur)
                longueur := entier(longueur)
                j := j + longueur – 3
                cpt := cpt + 1
        finTQ
        k := j
Fin

Procédure nombre_element ( i : entier , j : entier, sup : entier)
Var
J : entier ; cpt : entier
Debut
        cpt := 0
        TQ (j < sup) faire
                Recuperer_chaine(3,bloc,j,longueur)
                longueur := entier(longueur)
                j := j + longueur – 3
                cpt := cpt + 1
        finTQ
retourner cpt
Fin

insertion(nomfichier : chaine; element : typeEnreg )
Var
```

```
Trouv,continu : booleen ;     i,j,k : entier
Debut
        c := element [5 .. 24]
        Recherche( c , nomfichier , trouv ,  i , j)
        Si (trouv == faux ) alors
                ouvrir(f,nomfichier,'A')
                liredir(f,i,buf)
                L := element [1..3]     // L : taille de l'enreg
                t := taille(buf)  //nombre des caractères dans le bloc
                Si ( L + t < b ) alors                  //buffer n'est pas plein
                        K :=0
                        N := j + 1
                        // sauvegarder les elemnts apres la postion j
                        Tantque ( N  < b ) faire
                                Articles [K ] := buf.tab[N]
                                K := K + 1
                                N := N + 1
                        finTQ
                        cpt := 1
                        tantque(cpt <= L ) faire
                                buf.tab[j ] := element[cpt]
                                j := j+ 1
                                cpt := cpt + 1
                        finTQ
                        cpt := 1
                        tantque(cpt <= k ) faire
                                buf.tab[j ] := articles[cpt]
                                cpt := cpt + 1
                                j := j + 1
```

```
            finTQ
        Sinon
            K :=0
            N := j + 1
            // sauvegarder les elemnts apres la postion j
            Tantque ( N  < b ) faire
                Articles [K ] := buf.tab[N]
                K := K + 1
                N := N + 1
            finTQ
            cpt := 1
            tantque(cpt <= L ) faire
                buf.tab[j ] := element[cpt]
                j := j+ 1
                cpt := cpt + 1
            finTQ
            cpt := 1
            tantque(j <=b ) faire
                buf.tab[j ] := articles[cpt]
                cpt := cpt + 1
                j := j+ 1
            finTQ
            i := i+1

        finSi

sinon
        ecrire ("L'élément existe déjà ! " )
```

```
        finSi
        fermer(f)
Fin
```

```
Procédure Ecrire_chaine  (n :entier , ch : chaine , Var i,j : entier )
Var
K : entier ; i1 : entier
Debut
        Pour ( k := 1 , n ) faire
                Si ( j<=b ) alors
                        Buf.tab[j] := ch[k]
                        j := j + 1
                sinon
                        ecriredir(f,i,buf)
                        i := i + 1
                        buf.tab [1] := ch[k]
                        j := 2
                finSI
        finPour

Fin

Procédure Suppression_logique(c : chaine[20] , nomfichier :chaine)
Var
        Trouv : booleen ; i,j : entier
Debut
```

```
        Ouvrir(f,nomfichier,'A')
        Recherche(c,nomfichier,trouv,i,j)
        Si (trouv ) alors
                j := j + 3
                si ( j > b ) alors
                        j := j – b
                        i := i + 1
                finSI
                buf.tab[j] := ' V '
                ecriredir(f,i,j)

        Sinon
                Ecrire ( ''L'élément n'existe pas ! '')
        finSI
Fin
articles : structure
        tab : tableau[tailleMax]de caractères
        nb : entier                //nombre des enreg
fin structure
```

Procédure Requete(a:entier, b:entier , nomfichier : chaine ; var articles : articles)

```
Var
   i,j,k : entier
Debut
        ouvrir(f,nomfichier,'r')
        Rech(a,nomfichier,trouv,i,j)
        TQ ( i < entete(f,2) ou i == entete(f,2) et j != entete(f,3)) faire
                liredir(f,i,buf)
```

```
            k := 1
            Recuperer_chaine(3,i,j,longeur) ; Recuperer_chaine(1,i,j,efface)
            Recuperer_chaine(20,i,j,cle_rech)
            cle_rech := entier (cle_rech)          //convertir
            longueur := entier (longueur)          //convertir
            j := j – 24
            Si (cle_rech >= a et cle_rech <= b) et (efface == 'F') alors
                  Pour ( n := 1 , longueur ) faire
                        articles.tab[k] := buf.tab[j]
                        k := k + 1
                        j := j + 1
                  finPour
                  articles.nb := articles.nb + 1

            Sinon
                  j := j + nombre( longeur)
                  si ( j > b ) alors
                        j := j – b
                        i := i + 1
                  finSI
            finSi
      finTQ

fermer(f)
Fin
```

Type ToVnc

```
type tbloc : structure
      tab : tableau [1..b] de caractères
```

```
        NB : entier
        suivant : entier
        prec : entier
fin structure
f = fichier de tbloc buffer buf ENTETE(entier , entier ,entier ,entier ,entier)
// 1 car = num du premier bloc
 2 car = num de dernier bloc
3 car = la 1ere position libre dans le dernier bloc
4 car =  le nombre de caractères perdus suite aux suppressions logiques
5 car = position de dernier element dans le bloc
```

l'enregistrement peut avoir le forme suivante :

bbb | b | bbbbbbbbbbbbbbbbbbbb |bbbb…….b

3 car pour la taille de l'enregistrement

1 car pour indiquer l'effacement logique

20 car pour la clé de l'enreg

N car pour les autres informations

Procédure Recherche(c:entier ; nomfichier : chaine ; var trouv :booleen , i , j)

```
Var
bi , bs , inf , sup : entier ;
trouv , stop:= booléen
Debut
        ouvrir(f,nomfichier,'r')
        bs := entete(f,2)
        bi := 1
        trouv := faux ; stop := faux ; j := 1
        TQ ( bi <= bs et Non trouv et Non stop )
                i := (bi + bs ) div 2
```

```
liredir(f , i , buf)
j := j + 4
Recuperer_chaine(20,i,j,cle1)
j := entete(f, 5)
j := j + 4
Recuperer_chaine(20,i,j,cle2)
cle1 := entier(cle1) ; cle2 := entier(cle2)
Si ( c >= cle1 et c <= cle2 ) alors
        inf := 1 ; sup := nombre_element(i)
        // nombre_element == fonction qui calcule nombre des éléments
dans le bloc
        TQ (inf <= sup et non trouv )
                Milieu(i,inf,sup, k)
                Recuperer_chaine(3,i,k,longueur)
                Recuperer_chaine(1,i,k,efface)
                Recuperer_chaine(20,i,k,cle)
                longueur := entier(longueur)
                cle := entier(cle)
                longueur := entier(longueur)
                Si ( c == cle ) alors
                        Trouv := vrai
                Sinon
                        Si (c < cle ) alors
                                Sup := k -24
                        sinon
                                inf := k + longueur – 24
                        finSI
                finSI
        finTQ
        Si( inf > sup) alors
```

```
                    j := inf
                finSI
                stop := vrai
        Sinon
            Si (c < cle1 )
                bs := i - 1
            sinon
                bi := i + 1
            finSI
        finSI
        finTQ
        SI (bi > bs ) alors
            i := bi
            j := 1
        finSi
fermer(f)
Fin
```

Procédure recuperer_chaine(n :entier : Var i,j :entier ; Var ch : chaine)

Var k :entier

Debut

```
Pour (k := 1 ,n ) faire
        Si (j <= b ) alors
            Ch[k] := buf.tab[j]
            j := j + 1
        finsi
finPour
```

Fin

Procédure milieu (i : entier , j : entier , sup : entier ; var k : entier)

```
Var
Debut
        nenreg := nombre_element(i,j,sup)
        nenreg := (nenreg) div 2
        cpt := 0
        TQ (cpt < nenreg) faire
                Recuperer_chaine(3,bloc,j,longueur)
                longueur := entier(longueur)
                j := j + longueur – 3
                cpt := cpt + 1
        finTQ
        k := j
Fin

Procédure nombre_element ( i : entier , j : entier, sup : entier)
Var
J : entier ; cpt : entier
Debut
        cpt := 0
        TQ (j < sup) faire
                Recuperer_chaine(3,bloc,j,longueur)
                longueur := entier(longueur)
                j := j + longueur – 3
                cpt := cpt + 1
        finTQ
retourner cpt
Fin
```

insertion(nomfichier : chaine; element : typeEnreg)

```
Var
        Trouv,continu : booleen ;     i,j,k : entier
Debut
        c := element [5 .. 24]
        Recherche( c , nomfichier , trouv ,  i , j)
        Si (trouv == faux ) alors
                ouvrir(f,nomfichier,'A')
I               liredir(f,i,buf)
                 L := element [1..3]     // L : taille de l'enreg
                t := taille(buf)  //nombre des caractères dans le bloc
                Si ( L + t < b ) alors              //buffer n'est pas plein
                        K :=0
                        N := j + 1
                        // sauvegarder les elemnts apres la postion j
                        Tantque ( N  < b ) faire
                                Articles [K ] := buf.tab[N]
                                K := K + 1
                                N := N + 1
                        finTQ
                        cpt := 1
                        tantque(cpt <= L ) faire
                                buf.tab[j ] := element[cpt]
                                j := j+ 1
                                cpt := cpt + 1
                        finTQ
                        cpt := 1
                        tantque(cpt <= k ) faire
                                buf.tab[j ] := articles[cpt]
```

```
                cpt := cpt + 1
                j := j + 1
            finTQ
        Sinon
            K :=0
            N := j + 1
            // sauvegarder les elemnts apres la postion j
            Tantque ( N  < b ) faire
                Articles [K ] := buf.tab[N]
                K := K + 1
                N := N + 1
            finTQ
            cpt := 1
            tantque(cpt <= L ) faire
                buf.tab[j ] := element[cpt]
                j := j+ 1
                cpt := cpt + 1
            finTQ
            cpt := 1
            tantque(j <=b ) faire
                buf.tab[j ] := articles[cpt]
                cpt := cpt + 1
                j := j+ 1
            finTQ
            i := buf .suiv

        finSi

sinon
```

```
            ecrire ("L'élément existe déjà ! " )

        finSi
        fermer(f)
Fin
```

Procédure Suppression_logique(c : chaine[20] , nomfichier :chaine)

Var

Trouv : booleen ; i,j : entier

Debut

```
        Ouvrir(f,nomfichier,'A')
        Recherche(c,nomfichier,trouv,i,j)
        Si (trouv ) alors
            j := j + 3
            si ( j < b ) alors
             buf.tab[j] := ' V '
            ecriredir(f,i,j)
            finSI

        Sinon
            Ecrire ( ''L'élément n'existe pas ! '')
        finSI
```

Fin

```
articles : structure
        tab : tableau[tailleMax]de caractères
        nb : entier                 //nombre des enreg
fin structure
```

Procédure Requete(a:entier, b:entier , nomfichier : chaine ; var articles : articles)

```
Var
   i,j,k : entier
Debut
        ouvrir(f,nomfichier,'r')
        Rech(a,nomfichier,trouv,i,j)
        TQ ( i < entete(f,2) ou i == entete(f,2) et j != entete(f,3)) faire
                liredir(f,i,buf)
                 k := 1
                Recuperer_chaine(3,i,j,longeur) ; Recuperer_chaine(1,i,j,efface)
                Recuperer_chaine(20,i,j,cle_rech)
                cle_rech := entier (cle_rech)          //convertir
                longueur := entier (longueur)          //convertir
                j := j – 24
                Si (cle_rech >= a et cle_rech <= b) et (efface == 'F') alors
                        Pour ( n := 1 , longueur ) faire
                                articles.tab[k] := buf.tab[j]
                                k := k + 1
                                j := j + 1
                        finPour
                        articles.nb := articles.nb + 1

                Sinon
                         j := j + nombre( longeur)
                        si ( j > b ) alors
                                j := 1
                                i := i + 1
                        finSI
```

```
                finSi
        finTQ

fermer(f)
Fin
```