Exercice 1 :

Fichier avec Essai-lineaire combinant suppressions logique et physique
a)
La case vide est representee par un bloc non plein
Pour être sur qu'il existe toujours au moins une case vide dans le fichier, on peut utiliser (comme caracteristique) un compteur d'insertions, celui-ci doit toujours être < N*b
b) Les caracteristiques du fichier ainsi que la declaration d'un buffer
- Les caracteristiques du fichiers peuvent inclure :
le nombre d'insertions, le nombre de suppressions logiques, …
- Declaration d'un buffer ;
buf:Tbloc ;

Type Tbloc = struct
NB : entier // nombre de donnees dans le bloc
tab : tableau[b] d'entiers // les donnees du bloc
eff : tableau[b] de booleens // les indicateurs d'effacement logiques
fin.
c) Algorithmes de suppression et d'insertion

sup(x)

```
Rech( x, trouv, i, j ) ;
SI ( trouv )
    LireDir( F, i, buf ) ;
    SI ( buf.NB < b )  // bloc non plein, donc suppression physique …
          buf.tab[ j ] ← buf.tab[ buf.NB ] ;
          buf.NB-- ;
          EcrireDir( F, i, buf ) ;
          Aff_Entete( F, 1, Entete(F,1) - 1 ) ;
   SINON  // bloc plein, donc suppression logique …
          buf.eff[j] ← VRAI ;
          EcrireDir( F, i, buf )
   FSI
FSI
```

ins(x)

```
Rech( x, trouv, i, j ) ;
SI ( Non trouv )
    LireDir( F, i, buf ) ;
    SI ( j ≤ buf.NB ) // donc remplacement d'une donnée effacée logiquement
          buf.tab[ j ] ← x ;
          buf.eff[ j ] ← FAUX ;
          EcrireDir( F, i, buf )
    SINON    // insertion dans une case vide (si ce n'est pas la dernière) …
     SI ( Entete(F,1) < N*b - 1 )
           buf.NB++ ;
           buf.tab[ buf.NB ] ← x ;
```

```
            buf.eff[ buf.NB ] ← FAUX ;
            EcrireDir( F, i, buf ) ;
            Aff_Entete( F, 1, Entete(F,1) + 1 ) ;
        SINON
            ecrire(≪ Pas de place pour inserer une nouvelle donnee ≫)
        FSI
    FSI
FSI // (Non Trouv)
```

Exercice 2 :

Nobre total de blocs et le numéro du dernier bloc dans la zone de débordement

```
type
Tbloc = structure
tab : TABLEAU[1..b]de Tenreg;
NB : entier
Fin;

var

F : FICHIER de Tbloc BUFFER buf entete( entier, entier );

// entete(F,1) : désigne N (change à chaque réorganisation)
// entete(F,2) : désigne le dernier bloc en zone de débordement (initialement N+1)

Recherche( c : entier; var trouv : Booleen; var e:Tenreg )
var
i, j, N, DernierBloc : entier;

DEBUT // Recherche
ouvrir( F, « donnees.dat », « A » );
N := entete(F,1);
DernierBloc := entete(F,2);// on commence la recherche par un accès direct au bloc h(c)
i := h(c);
lireBloc(F,i,buf);
j := 1;
trouv := FAUX;
TQ ( Non trouv ET j <= buf.NB )
        SI (buf.tab[j].cle = c)
            trouv := VRAI;
            e := buf.tab[j]
        SINON
          j := j + 1
    FSI
FTQ;

// si la clé c n'existe pas dans le bloc i et celui-ci est plein, il faudra continuer la recherche

// dans la zone de débordement (de manière séquentielle)
```

```
SI (Non trouv et buf.NB = b)
        i := N+1;
        TQ (Non trouv ET i <= DernierBloc) // parcours séquentiel de la zone de débordement
            lireBloc(F,i,buf);
            j := 1;
           TQ ( Non trouv ET j <= buf.NB )
                SI (buf.tab[j].cle = c)
                        trouv := VRAI;
                        e := buf.tab[j]
                SINON
                        j := j + 1
                FSI;
            FTQ;

            SI (Non trouv)
                i := i + 1
            FSI
FTQ // (Non trouv ET i <= DernierBloc)
FSI; // Non trouv et buf.NB = b
Fermer( F );
FIN // Recherche
```