Partie A (8 points)

Questions de cours sur 5pts

1. Donner les structures de tables en RAM pour les méthodes:
   (a) - Index primaire à un niveau
   (b) - Index primaire à 2 niveaux
   (c) - Index secondaire sous forme de listes inversées

Solution :

(0.5 pt) (a) - Structures des tables : Index primaire à un niveau

```
Type T1 = Structure
    Clé    : Typeclé
    Adr    : Typeadresse
Fin
VAR T : Tableau (1..N) de T1
```

(0.75 pt) (b) - Structures des tables : Index primaire à deux niveaux

```
Type T1 = Structure
    Clé    : Typeclé
    Adr1 : entier
Fin
Type T2 = Structure
    Clé    : Typeclé
    Adr2 : Typeadresse
Fin
VAR Tniv1 : Tableau (1..N1) de T1 //Index de niveau 1
VAR Tniv2 : Tableau (1..N2) de T2 //Index de niveau 2
```

(1.5 pts) (c) - Structures des tables : Index secondaire sous forme de listes inversées

```
Type T1 = Structure
    Clésec : Typeclé
    Adr1   : entier
Fin
Type T2 = Structure
    Cléprim : Typeclé
    Adr2      : entier
Fin
Type T3 = Structure
    Cléprim : Typeclé
    Adr       : Typeadresse
Fin
//Tab1 et Tab2 constituent l'index secondaire
VAR Tab1 : Tableau (1..N1) de T1
VAR Tab2 : Tableau (1..N2) de T2
VAR Tab3 : Tableau (1..N3) de T3 //Index primaire
```

(0.25 pt)

```
Type Typeadresse = Structure
    Num_Bloc : entier //Numéro de bloc
    Depl         : entier //Déplacement dans le bloc
Fin
```

2. Dire comment se fait une suppression d'article de clé primaire donnée dans la méthode d'accès multicritères ?

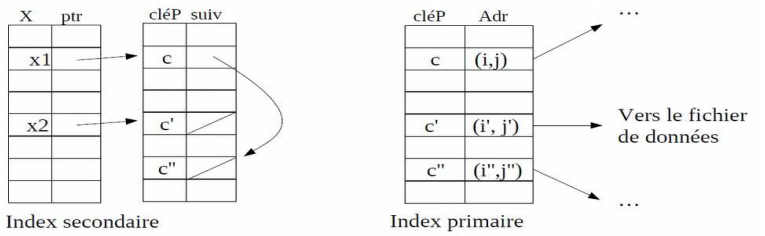
(1 pt) Solution :

Une suppression d'article de clé primaire donnée dans la méthode d'accès multicritères se fait ***uniquement au niveau de la table d'index primaire***.

3. C'est quoi la méthode « Listes inversées » ?

(1 pt) Solution :

La méthode de Listes inversées est quand on recherche les enregistrements suivant une clé secondaire (par exemple X=a), on utilise l'index secondaire sur ce champ pour récupérer la ou les clés primaires associées à la valeur cherchée (a). Pour chaque clé primaire trouvée, on utilise l'index primaire pour localiser l'enregistrement sur le fichier de données (numéro de bloc et déplacement).
Ou autrement dit, l'index secondaire référençant des listes chainées de clés primaires.



Exercice 1 sur 3 pts

Que fait cet algorithme ? Quelle est la structure du fichier utilisée ?

Solution :

(1 pt) L'algorithme fait le ***partage équitable des blocs consécutifs I, I+1 et I+2***.

(0.5 pt) La structure de fichier : tableau avec des enregistrements de taille fixe, soit ordonné ou non, autrement soit TOF ou TÕF.

Après exécution :

(0.5 pt) Bloc I

6							
a	b	c	d	e	f	g	h

(0.5 pt) Bloc I+1

6							
g	h	i	j	k	l	o	p

(0.5 pt) Bloc I+2

7							
m	n	o	p	q	r	s	

Hachage : ***Essai linéaire dynamique.***

But de l'exercice : Ecriture de la procedure de recherche d'une valeur, et la procedure d'insertion d'une valeur.

Les données :

- 2 fonctions de hachage hi et hi+1
- deux fichiers :

-Fp : fichier zone primaire du type TobarreF, ses enregistrements sont réduits à un entier donc on peut dire que c'est un fichier de valeur et non un fichier d'enregistrement.

-Fd : fichier zone de debordement associé à Fp du type LObarreF. Si insertion en zone de debordement alors elle sera faite en tete de liste si il y a de la place sinon une nouvelle tete est allouée.

-Les tailles des blocs de Fp et Fd ne sont pas forcement egales.

Travail demandé :

1.En plus des deux procedures citées plus haut, donnez les declarations complétes du programme principale.

Liste non exhaustive de type et var à utiliser :

-Les types :Tblocp, Tblocd, Fp(fichierprimaire), Fd(fichier debordement), entetep, enteted

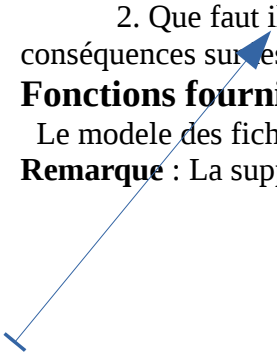
-Les variables : bufp,bufd // si vous avez besoin d'autre buffer alors rajouter un num ex : bufd2.

2. Que faut il faire pour accelerer la recherche en zone primaire et quelles sont les conséquences sur les autre procedures (rech, ins, supp)

Fonctions fournies :

Le modele des fichiers, condition d'éclatement, Eclatement d'un bloc et sa zone de debordement.

Remarque : La suppression qui n'est pas demandée est physique, à la manière des TobarreF.



Réponse : **1 point**

Ordonner le bloc de la zone primaire, pour utiliser la dichotomie, les consequenses : des decalages dans l'ins et la supp.

SOLUTION **2 points**

```
const maxp=P ; maxd=D ; // taille des blocs
type Tblocp : structure T[maxp] : entier ;
                        nb : entier ;
                        lien : entier
              fin ;
type Tblocd : structure T[maxd] : entier ;
                        nb : entier ;
                        lien : entier
              fin ;
 typedef struct Entete p
   {
      int adrDerBloc;
      int nbEnreg;
      int i; // niveau du fichier
      int p ; // bloc courant initialisé au 1er bloc
   } Entetep;
  typedef struct Enteted
   {  int adrDerBloc;
      int nbEnreg;
    } Enteted;
   typedef struct TObF
   {
      FILE *fichierp;
      Entetep entetep;
   } TobF;
   typedef struct LObF
   {
      FILE *fichierd;
      Enteted enteted;
   } TobF;
var bufp:Tblocp ;  bufd : Tblocd ;
```

Procedure recherche(cle : ent, var i,j,ip ent ; var trouv : bool ; var debord : bool) **2 points**

debut

```
ouvrir(Fp,A) ; ouvrir(Fd,A) // on peut supposer que les fichier sont ouverts à partir du pgmPrinc
trouv=faux ; debord=faux ;
 i=hi(clé) ;
si i<entetep(4)  alors i=hi+1(cle)   fsi ;  //entetp(4) : la var p de l'entetep
LirDir(F,i+1 ,bufp) // on met i+1 car la fct h est definit sur [0,n-1] et les blocs sur [1,n]
recherche seq de clé dans  bufp ;
si non trouv alors
   si bufp.lien != -1 alors
         ip=i ; voir explication dans insertion
         debord=vrai ; i=bufp.lien ; LirDir(Fd,i ,bufd) ;
         recherche seq dans bufd et la liste de bloc associé.
   Fsi
fsi
```

Fin

procedure insertion(cle:ent) 3 points

```
var i,j,ip,trouv,debord
bufd2:tblocd
debut
 recherche(cle,i,j,ip,trouv,debord) ;
si  not trouv alors  // inserer la cle dans le bloc i

   si  not debord  alors LirDir(Fp,i,bufp) ;  // ins en zone primaire
                         si bufp.nb<maxp  alors
                               bufp.T[j]=cle ; bufp.nb++ ;
                               ecrireDir(fp,i,bufp)
                         sinon    // bufp est plein donc ins en zone debord  pour la 1ere fois
                              i2=allocbloc(Fd);bufd.T(0)=cle ; bufd.nb=0 ;
                              bufd.lien=-1;bufp.lien=i2 ;
                              EcrireDir(Fp,i,bufp) ;
                              EcrireDir(Fd,i2,bufd) ;
                         fsi
  sinon // debord = vrai  ins en zone de debord
    LirDir(Fd,i,bufd) ;
   si bufd.nb<maxd alors
                          bufd.nb++ ; bufd.T[bufd.nb]=cle ;
                          EcrirDir(Fd,i,bufd)
    sinon
              i2=allocBloc(Fd) ;
              bufd2.T(0)=cle ;
              bufd2.nb=0 ; bufd2.lien=bufd.lien ;
              EcrirDir(Fd,i2,bufd2) ;
              // il reste à mettre à jour le lien de bufp dont on ne connaît pas le numero
             // 2 possiblites : 1-rehacher pour le retrouver
                              2- ajouter un parametre à la recherche qd il y a debord on retourne deux
                                 num de blocs ip et i (ip num blocp et i num blocd) c ce que j'ai choisis
              lirDir(Fp,ip,bufp) ; bufp.lien=i2 ; ecrirDir(Fp,ip,bufp) ;
   fsi

   fsi  // Fin de l'insertion
    si condition_eclatement alors
                           allouer nouveau bloc ds Fp
                          Eclater bloc p et sa zone de débordement
                          si p=2^i*n alors p=0 ; i=i+1 fsi  // remplacer i et p par entetep(3), entetep(4)
                                                           // donc c une maj de entetep
fsi

FIN.
```

Partie C : (4 points)

Exercice 3

Implémentation d'une file d'attente (FIFO) par une structure de fichiers en liste.

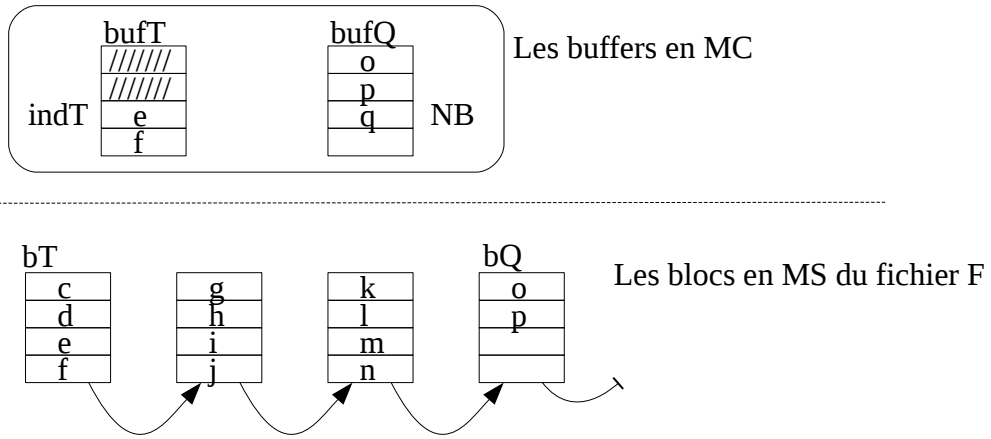
Soit F un fichier de Tbloc buffer bufT, bufQ entete (bT, bQ, NbEnreg, indT, teteLV)

bufT et bufQ représentent les zones tampons par lesquels les éléments de la file transitent.
L'enfilement se fait dans bufQ et le défilement à partir de butT.
Le premier bloc du fichier (le bloc de tête bT) est associé à bufT et le dernier bloc du fichier (le bloc de queue bQ) est associé à bufQ

```
type Tbloc : Structure
        tab:tableau[b] de Tenreg ;
        NB : entier ;
        Suiv : entier ;
     Fin ;
```

Les caractéristiques du fichier (Entête) sont :
- bT, bQ : numéros des blocs de tête et de queue de la file F
- NbEnreg : nombre d'enregistrements dans le fichier // optionnel
- indT : indice (dans bufT) du premier élément de la file (tête de file)
  // l'indice de la queue est représentée par le champ NB de bufQ (le dernier élément de la file)
- teteLV : numéro du bloc de tête de la liste de blocs vides

La figure ci-dessous schématise une file composée des éléments : e,f,g,h,i,j,k,l,m,n,o,p,q



Au niveau de bufT, le défilement se fait par incrémentation de l'indice du premier élément (indT). Cela évite les décalages en MC.
Au niveau de bufQ, l'enfilement se fait en incrémentant le champ NB (insertion à la première position libre dans bufQ).

A l'initialisation, le fichier F est vide (aucun bloc n'est utilisé).
On initialise alors la tête (bT) et la queue (bQ) à -1. L'entête 'indT' peut ne pas être initialisée.

CreerFile(F)

```
bT ← -1 ; bQ ← -1 ; indT ← 0 ;   // Aff_entete(F,1, -1) ; Aff_entete(F,2, -1) ; Aff_Entete(F,4, 0)
NbEnreg = 0 ; teteLV← -1 ;       // Aff_entete(F,3, 0) ; Aff_entete(F,5, -1)
bufT.NB ← 0 ;  bufQ.NB ← 0 ;
```

Pour tester si la file est vide, il suffit de vérifier si le numéro de bloc de tête ou de queue (bT ou bQ) est à -1

FileVide(F) : bool

```
Si (bT = -1)    // Entete(F,1) = -1
        retourner Vrai
Sinon   retourner Faux
Fsi
```

Lors de l'enfilement d'un élément v, on le rajoute à la première position libre (NB+1) dans bufQ (en MC). Si ce dernier est plein, on rajoute alors un bloc en fin de fichier.
Le bloc rajouté est soit un nouveau bloc alloué au fichier (AllocBloc), soit un ancien bloc récupéré depuis la liste des blocs vides (teteLV) si cette dernière n'est pas vide.

Enfiler(F, v)

```
Si ( FileVide(F) )     // cas particulier où la file est vide ...
   // il faut allouer un nouveau bloc ou réutiliser un ancien ...
   Si ( teteLV = -1 )   // Entete(F,5) = - 1
         i ← AllocBloc(F)
   Sinon  i ← teteLV ;   LireDir( F, i, bufT ) ;
         teteLV ← bufT.Suiv      // Aff_Entete(F,5, bufT.Suiv)
   Fsi ;
   bufT.Suiv ← -1 ;   bufT.NB ← 1 ;  indT ← 1 ; bufT.tab[1] ← v ;
   // maintenant la file contient 1 seul bloc, c'est en même temps la tête et la queue ...
   bufQ ← bufT ;        // ou affecter champ par champ …
   bQ ← i ;  bT ← i ;   // Aff_Entete(F,1, i) ; Aff_Entete(F,2, i)

Sinon                  // cas général où la file contient déjà au moins un bloc ….
   // La file n'est pas vide. L'insertion de v se fait donc à la fin (Queue) de la file ...
   Si ( bufQ.NB < b )
      bufQ.NB++ ;   bufQ.tab[ bufQ.NB ] ← v ;
      Si ( bQ = bT ) // si la file ne contenait qu'un seul bloc…
         bufT ← bufQ  // dupliquer les changements dans bufT aussi
      Fsi
   Sinon // bufQ est déjà plein, rajouter un bloc en queue de liste ...
      // soit en allouant un nouveau soit en réutilisant un ancien depuis la liste des blocs vides
      Si ( teteLV = -1 )   // Entete(F,5) = -1
         i ← AllocBloc(F) ;
         ModifLV = 0 ;     // indicateur pour ne pas m-a-j la liste LV ci-dessous
      Sinon
         i ← teteLV ;
         ModifLV = i ;      // indicateur pour m-a-j la liste LV ci-dessous
      Fsi
      bufQ.Suiv ← i ;   EcrireDir(F, bQ, bufQ ) ;
      Si ( ModifLV != 0 )  // m-aj de la liste des blocs vides …
         LireDir( F, i, bufQ ) ;
         teteLV ← bufQ.Suiv ;     // Aff_Entete( F, 5, bufQ.Suiv )
      Fsi ;
      // préparation du nouveau bloc de queue ...
      bufQ.Suiv ← -1 ;   bufQ.NB ← 1 ;    bufQ.tab[ 1 ] ← v ;
   Fsi
Fsi ;
NbEnreg++ ;        // Aff_Entete(F,3, Entete(F,3) + 1 )
```

Lors du défilement (dans v), on récupère l'élément d'indice 'indT' depuis le buffer bufT et on incrémente l'indice.
Si tous les éléments de bufT ont été défilés, on ramène le prochain bloc dans bufT.
L'ancien premier bloc est rajouté en début de la liste des blocs vide (teteLV) pour être réutilisé plus tard.

```
Defiler( F, v )    // v en sortie
  Si ( Non FileVide(F) )
     v ← bufT.tab[ indT ] ;  // v ← bufT.tab[ Entete(F,4) ]
     indT++ ;                // Aff_Entete(F,4, Entete(F,4) + 1 )
     Si (indT > bufT.NB )  // si le bloc de tête devient vide …
        // rajouter le bloc bT en tête de liste teteLV …
        i ← teteLV ;          // Entete(F,5)
        teteLV ← bT ;         // Aff_Entete(F,5, bT)
        bT ← bufT.Suiv ;      // Aff_Entete(F,1, bufT.Suiv)
        bufT.Suiv ← i ;
        EcrireDir( F, teteLV, bufT ) ;
        // Lire le prochain bloc de la file …
        Si ( bT != -1 )
           Si ( bT != bQ )
              LireDir( F, bT, bufT ) ;
           Sinon
              bufT ← bufQ
           Fsi
        Sinon
           // la file est devenue vide …
           bQ ← -1 ;  // Aff_Entete(F,2, -1)
        Fsi
     Fsi ;

    NbEnreg-- ;        // Aff_Entete(F,3, Entete(F,3) - 1 )
  Fsi
```