CONCOURS d'accès au Cycle Supérieur de l'ESI

Corrigé de l'épreuve : Algorithmique et Programmation
Code : ALGPRO

Partie 1 : Structures statiques

Question 1 :

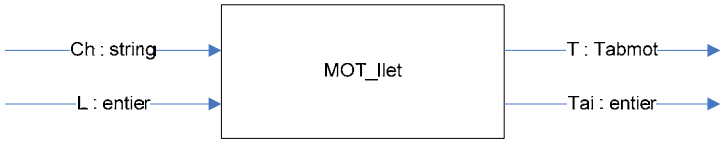
Plusieurs découpages sont possibles, donc tout autre découpage cohérent sera retenu.

Le découpage et sa justification :

- Il faut extraire de la chaîne les mots ayant une certaine longueur (1 lettre, 2 lettres , …) et les mettre dans un tableau à une dimension . ***(Module Mot_iLet)***

- Puis il faut trier les mots contenus dans ce tableau. On a pris le tri par sélection mais peu importe le type de tri choisi, il faut simplement l'adapter aux chaînes de caractères ***(module Tri_SelC)***

- Finalement, on doit ranger un tableau dans une ligne donnée de T2 (à deux dimensions). Il n'est pas nécessaire de faire un module pour cela.



Rôle : Extrait de la chaîne Ch les mots de i lettres et les range dans le tableau T (Tai :nbre éléments de T)

Rôle : Trie le tableau T selon la méthode par séléction

```
Type    Mot = Chaîne[25]
        Tabmot = tableau[1..100] de mot
```

Question 2 :

Analyse de l'algorithme principal :

Comme le mot le plus long de la langue française est 'anticonstitutionnellement' (25 lettes).

- On fait varier *i* de 1 à 25 , et à chaque fois // *pour extraire les mots de 1 lettre , puis 2 lettres, …*
  - On met les mots de *i* lettres dans un tableau à une dimension T1 ***(Module Mot_iLet)***
  - On trie T1 ***(module Tri_SelC)***
  - On met T1 dans la $i^{ième}$ ligne de T2

Algorithme principal

```
Algorithme SUJconc3
Type    Mot = Chaine [25]
        Tabmot = Tableau [ 1..100] de mot
        Tab2M = Tableau [1..25] de tabmot
Variables   p :Chaîne
            Table : tabmot
             taille, i,j:Entier;
            T2:Tab2M;
Debut
Lire (p)
Pour i allant de 1 à 25 faire
          Dpour
          | Mot_Ilet (p, i , table , taille)
          | tri_selc (table , taille)
          | Pour j allant de 1 à taille faire T2[i,j] ← table[j]
          | Fpour
Fin
```

Programme

```
program SUJconc3;
Type    Mot=string[25];
        tabmot=array[1..100] of mot;
        Tab2M = array [1..25] of tabmot;
Var  p :string;
     table:tabmot;
     taille,i,j:integer;
     T2:Tab2M;

BEGIN
Write ('donner votre expression :');
readln(p);
for i := 1 to 25 do
        BEGIN
        Mot_Ilet(p,i,table,taille);
        tri_selc(table,taille);
        for j := 1 to taille do T2[i,j]:=table[j];
        END;
readln;
END.
```

Partie 2 : Structures dynamiques

```
{
Création d'un Arbre de recherche binaire
à partir d'une liste linéaire chaînée
Parcours récursif inordre de l'arb
}

A_r_b est l'abre de recherche binaire à créé ( initialisé à Nil)

//Parcours de la liste
  ACTION Creation_arb ( L ) ;
  SOIT
     L UNE LISTE ;
  DEBUT
     SI L <> NIL
        APPEL Inserer_arb ( VALEUR ( L ) ) ;
        APPEL Creation_arb ( SUIVANT ( L ) ) ;

     FSI
  FIN

//Recherche / Insertion dans l'arb
ACTION Inserer_arb ( Val )
  SOIT
     P , Q DES POINTEURS VERS ARB ;
     Val DES ENTIERS ;

  DEBUT
    //Au départ
     SI A_r_b = NIL  // Arbre à créer
        CREERNOEUD ( P ) ;
        A_r_b := P ;
        AFF_INFO ( P , Val ) ;

     SINON
       //Recherche dans l'arb
        P := A_r_b ;
        Q := P ;
        TQ ( Val <> INFO ( P ) ) ET ( Q <> NIL ) :
           P := Q ;
           SI Val < INFO ( P )
              Q := FG ( P )
           SINON
              Q := FD ( P )
           FSI
        FTQ ;
       //Insertion dans l'arb
        SI Val <> INFO ( P )
           CREERNOEUD ( Q ) ;
           AFF_INFO ( Q , Val ) ;
           SI Val < INFO ( P )
              AFF_FG ( P , Q )
           SINON
              AFF_FD ( P , Q )
           FSI
       FSI
     FSI
```

```
  FIN

//Affichage
 ACTION Inordre ( Q )
  SOIT
     Q UN ARB ;
  DEBUT
     SI Q <> NIL
        APPEL Inordre ( FG ( Q ) ) ;
        ECRIRE ( INFO ( Q ) ) ;
        APPEL Inordre ( FD ( Q ) )
     FSI
  FIN
  {
  Création d'un Arbre de recherche m-aire d'ordre 4
  à partir d'un arbre de recherche binaire
  Parcours récursif inordre de l'arm
  }
```

A_r_m est l'abre de recherche m-aireà créé (initialisé à Nil)

```
  // Parcours de l'arbre de recherche binaire
   ACTION Creation_arm ( A ) ;
   SOIT
      A UN ARB ;

   DEBUT
      SI A <> NIL
         APPEL Creation_arm ( FG ( A ) ) ;
         APPEL Inserer_arm ( INFO ( A ) ) ;
         APPEL Creation_arm ( FD ( A ) ) ;

      FSI
   FIN
```

Recherche / Insertion dans l'arm

```
   ACTION Inserer_arm ( Val )
   SOIT
      P , Q , R DES POINTEURS VERS ARM ( 4 ) ;
      Nt , J , I , Pos , Val , N DES ENTIERS ;
      Trouv1 , Trouv2 DES BOOLEENS ;

   DEBUT
//Au départ
      SI A_r_m = NIL
         CREERNOEUD ( P ) ;
         A_r_m := P ;
         AFF_DEGRE ( P , 1 ) ;
         AFF_INFOR ( P , 1 , Val ) ;

      SINON
//Recherche dans l'arm
         P := A_r_m ;
         Trouv1 := FAUX ;
         TANTQUE NON Trouv1 ET ( P <> NIL ) :
         { Recherche dans le noeud }
            Trouv2 := FAUX ;
            Pos := 1 ;
            TANTQUE NON Trouv2 ET ( Pos <= DEGRE ( P ) ) :
               SI INFOR ( P , Pos ) >= Val
```

```
            Trouv2 := VRAI
          SINON
            Pos := Pos + 1
          FSI
        FTQ ;
        Q := P ;
        SI Trouv2
          SI Val = INFOR ( P , Pos )
            Trouv1 := VRAI
          SINON
            P := FILS ( P , Pos )
          FSI
        SINON
          P := FILS ( P , Pos )
        FSI
      FTQ ;
      //Insertion dans l'arm

      SI NON Trouv1 :
        SI DEGRE ( Q ) < Ordre - 1 :
          Nt := DEGRE ( Q ) ;
          AFF_DEGRE ( Q , Nt + 1 ) ;
          POUR J := Nt , Pos , - 1
            AFF_INFOR ( Q , J + 1 , INFOR ( Q , J ) ) ;

          FINPOUR ;
          AFF_INFOR ( Q , Pos , Val ) ;

        SINON
          CREERNOEUD ( R ) ;
          AFF_DEGRE ( R , 1 ) ;
          AFF_INFOR ( R , 1 , Val ) ;
          AFF_FILS ( Q , Pos , R )
        FSI
      FSI ;

    FSI
  FIN

//Affichage
  ACTION Inordre ( Q )
  SOIT
    Q UN ARM ( 4 ) ;
    I UN ENTIER ;

  DEBUT
    SI Q <> NIL
      POUR I := 1 , DEGRE ( Q )
        APPEL Inordre ( FILS ( Q , I ) ) ;
        ECRIRE ( INFOR ( Q , I ) )
      FINPOUR ;
      APPEL Inordre ( FILS ( Q , DEGRE ( Q ) + 1 ) )
    FSI
  FIN
```

Mesures et Suggestion

	Complexité	Amélioration
Arbre de recherche binaire	$O(Log_2(n))$	Maintien de l'équilibre de l'arbre avec les technique AVL ou RB
Arbre de recherche 4-aire	$O(Log_4(n))$	Construction ascendante de

		l'arbre m-aire

CONCOURS d'accès au Cycle Supérieur de l'ESI

Corrigé de l'épreuve : Algorithmique et Programmation
Code : ALGPRO

Partie 3 : Structures de fichiers

Question 1 :

Chaque enregistrement occupe 270 caractères (255 pour la partie info et 15 caractères pour la clé [8 pour la date, 6 pour l'heure et le caractère '-']).
Donc le nombre maximal d'enregistrements par bloc est : 4096 div 270 = **15**.

La déclaration du fichier pourrait être de la forme :

```
TypeEnreg = structure {
    cle : tableau[15] caractères;
    info : tableau[255] caractères
}

TypeBloc = structure {
        tab :  Tableau[15] de TypeEnreg;
        nb : Entier
}

F : Ficher de TypeBloc Buffer buf Entete (entier);
```

F est la variable fichier logique, buf est un buffer de type TypeBloc et on dispose d'une caractéristique de type entier 'entete(F,1)' indiquant le nombre de blocs utilisés par le fichier.

Question 2 :

Comme le fichier est ordonné, on utilisera alors la recherche dichotomique pour localiser t1 (ou alors la plus petite clé supérieure ou égale à t1) et on continue séquentiellement jusqu'à atteindre (ou dépasser) t2 :

```
Ouvrir(F, 'fichier_exemple.log') ;
enMemoire ← VRAI ;

//*** recherche dichotomique externe pour localiser t1 ...
bi ← 1 ;
bs ← Entete(F,1) ;
stop ← FAUX ;
trouv ← FAUX ;
j ← 1 ;

TQ (Non trouv & Non stop & bi <= bs)
    i ← (bi+bs) div 2
    LireDir(F, i, buf) ; // lecture du bloc i dans la var buf
    SI (t1 >= buf.tab[1].cle & t1 <= buf.tab[buf.nb].cle)
        Recherche_interne(buf, t1, trouv, j)
        stop ← VRAI
```

```
    SINON
        SI (t1< buf.tab[1].cle)
            bs ← i-1
        SINON  //  t1> buf.tab[buf.nb].cle
            bi ← i+1
        FSI
    FSI
FTQ ;
SI ( Non trouv & Non stop )
    i ← bi ;  j ← 1 ;  enMemoire ← FAUX
FSI ;

//*** parcours séquentiel jusqu'à dépasser t2...
stop ← FAUX ;
TQ ( Non stop et i <= Entete(F,1) )
    SI (Non enMemoire)   // le 1er bloc (le bloc num i) est déjà lu
        LireDir(F, i, buf) ;  enMemoire ← VRAI
    FSI ;
    k ←  j ;
    TQ ( k <= buf.nb & Non stop )
        SI ( buf.tab[j].cle <= t2 )
            Ecrire( buf.tab[j]) ;
        SINON
            stop ← VRAI
        FSI ;
        k ← k+1
    FTQ ;
    i ← i+1 ; j ← 1
FTQ ;
Fermer(F)

// La recherche dichotomique interne …
Recherche_interne(buf : TypeBloc,  t1:chaine,  VAR trouv:booleen, VAR j:entier)
jmin ← 1 ;
jmax ← buf.nb ;
trouv ← FAUX ;
TQ ( jmin < jmax & Non trouv )
    j ← (jmin + jmax) div 2 ;
    SI ( t1 < buf[j].cle )
        jmax ← j – 1
    SINON
        SI ( t1 > buf[j].cle )
            jmin ← j + 1
        SINON // égalité...
            trouv ← VRAI
        FSI
    FSI
FTQ ;
SI (Non trouv)
    j ← jmin
FSI
```

Question 3 :

L'idée est de parcourir les n fichiers en même temps en utilisant n buffers en mémoire centrale. On réalise la fusion des enregistrements présents dans les n buffers sur un buffer de sortie (le n+1ème buffer). Quand ce dernier buffer est plein, on le vide sur le fichier de sortie.

```
Tf : tableau[n+1] de fichier de TypeBloc ;
Ti : tableau[n+1] d'entier ;        // les numéros de blocs manipulés
Tj : tableau[n+1] d'entier ;        // les numéros d'enregistrements dans les blocs
Tbuf : tableau[n+1] de TypeBloc ;  // les n+1 buffers en mémoire centrale.

Const maxBloc = 15 ;      // capacité maximale d'un bloc

// *** ouverture des fichiers en entrée …
Pour i = 1 ... n
    Ouvrir(Tf[i], …) ;
    // Lecture des premiers blocs …
    SI ( Entete(Tf[i],1) > 0 )
        LireDir( Tf[i], 1, Tbuf[i] ) ;
        Ti[i] ← 1 ;
        Tj[i] ← 1 ;
    SINON
        Ti[i] ← -1
    FSI
FP ;

// *** ouverture d'un nouveau fichier en sortie …
Ouvrir( Tf[n+1], …) ;
Ti[n+1] ← 1 ;
Tj[n+1] ← 1 ;

// *** boucle principale de la fusion de n fichiers ordonnés …
stop ← FAUX ;
TQ ( Non stop )
    // récupère dans i le num du fichier associé à la plus petite clé dans les n buffers...
    // si tous les enregistrements des n fichiers ont déjà été lus, stop devient VRAI.
    recuperer_plus_petit( i, stop ) ;

    SI (Non stop)
        // ajouter l'enreg au résultat...
        SI ( Tj[n+1] < maxBloc )
            Tbuf[n+1].tab[ Tj[n+1] ] ← Tbuf[i].tab[ Tj[i] ] ;
            Tj[n+1] ← Tj[n+1] + 1 ;
        SINON
            // vider le buf resultat sur le fichier de sortie...
            Tbuf[n+1].nb ← maxBloc ;       // rempli à 100 %
            EcrireDir( Tf[n+1], Ti[n+1], Tbuf[n+1] ) ;
            Ti[n+1] ← Ti[n+1] + 1 ;
            Tbuf[n+1].tab[ 1 ] ← Tbuf[i].tab[ Tj[i] ] ;
            Tj[n+1] ← 2
        FSI ;
```

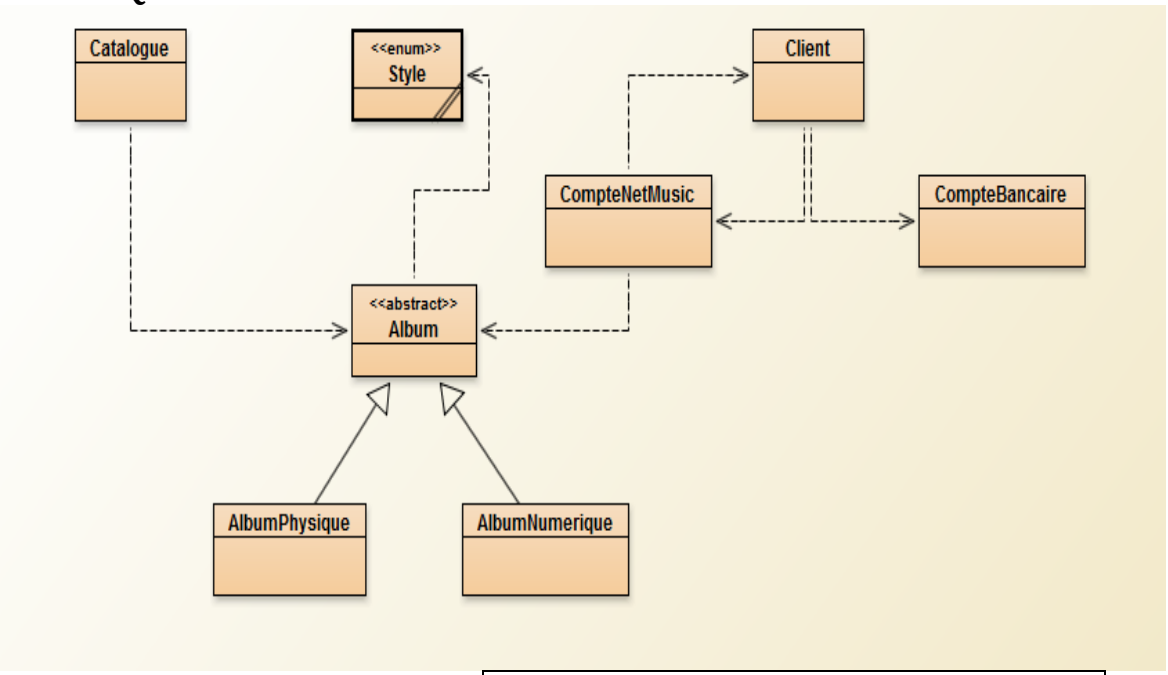
```
            // passer à l'enreg suivant dans le fichier Tf[i] …
            SI ( Tj|i] < Tbuf[i].nb )
                Tj|i] ← Tj|i] + 1
            SINON
                SI ( Ti[i] < Entete(Tf[i], 1) )
                    Ti[i] ← Ti[i] +1 ;
                    LireDir( Tf[i], Ti[i], Tbuf[i] ) ;
                    Tj[i] ← 1
                SINON
                    // marquer que le fichier a été complètement lu...
                    Ti[i] ← -1
                FSI
            FSI

        FSI  // (Non stop)
    FTQ ;

    // dernière écriture sur le fichier de sortie pour vides le buffer n+1 …
    SI ( Tj[n+1] > 1 )
        Tbuf[n+1].nb ← Tj[n+1] – 1 ;
        EcrireDir( Tf[n+1], Ti[n+1], Tbuf[n+1] ) ;
        // sauvegarde du nombre de bloc utilisé dans l'entête du fichier...
        Aff_Entete( Tf[n+1], 1, Ti[n+1] ) ;
    SINON
        // le fichier résultat est vide, aucune écriture n'a été faite...
        Aff_Entete( Tf[n+1], 1, 0 ) ;
    FSI ;

    // Fermeture des fichiers...
    POUR i = 1 … n+1
        Fermer( Tf[i] )
    FP

    recuperer_plus_petit( VAR i:entier, VAR stop:booleen )
    // retourne dans i le numéro du fichier en entrée, associé au buffer contenant la plus
    // petite clé, non encore considérée.
    // si toutes les clés ont déjà été considérées, on retourne dans stop VRAI.
    stop ← VRAI ;
    i ← 1 ;
    TQ ( i <= n & stop )
        SI ( Ti[i] <> -1 )
            stop ← FAUX
        SINON
            i ← i+1
        FSI
    FTQ ;
```

```
    SI ( Non stop )
        ind_min ← i ;
        i ← i + 1 ;
        TQ ( i <= n )
            SI ( Ti[i] <> -1)  // le fichier num i n'a pas été complètement lu...
                SI ( Tbuf[i].tab[ Tj[i] ].cle < Tbuf[ ind_min ].tab[ Tj[ inf_min ].cle )
                    ind_min ← i
                FSI
            FSI ;
            i ← i+1
        FTQ ;
        i ← ind_min
    FSI
```

Partie 4 : Programmation objets

Question 1 : Le concept orienté objet est le polymorphisme.
Explication : Prévoir une méthode abstraite dans la classe Album et la redéfinir au niveau des classes AlbumNumerique et AlbumPhysique

Question 2



Catalogue
Attributs
Private Set<Album> albums
Méthodes

Abstract class Album
Attributs
private String titre; private String auteur; private String MaisonProduction; private int année; private ArrayList <String> listeChansons; private double prix; private Style style;
Méthodes
public abstract double calculerPrix();

AlbumPhysique extends Album
Attributs

AlbumNumerique Extends Album
Attributs
Private String URL
Méthodes
Public double calculerPrix () // redéfinition

Enum Style
Attributs
Classique, Jazz, Rap, Pop, Rock, Rnb
Méthodes

Méthodes
Public double calculerPrix () // redéfinition

CompteNetMusic
Attributs
private String nomUser; private String motdePasse; private Client client; private ArrayList<Album> panier;
Méthodes
public void payer()

Client
Attributs
private String nom; private String adresse; private CompteBancaire compte; private CompteNetMusic compteNM ;
Méthodes

CompteBancaire
Attributs
String nom; String prenom private double solde;
Méthodes
Public void debiter(double somme) Public void accrediter(double somme) Public consulterSolde()

Question 3

```
public void payer()throws SoldeException{

double somme = 0;

for (Album a:panier){
     somme = somme + a.calculerPrix();
}
if (somme>client.getCompte().getSolde()) throw new SoldeException();

else client.getCompte().debiter(somme);
}
class SoldeException extends Exception{
public SoldeException(){
System.out.println(« Erreur : Votre solde est insuffisant pour effectuer ce
paiement ») ;
}
}
```

Remarque: L'étudiant peut utiliser un bloc try catch au lieu de throw et throws.
Il peut également ne pas déclarer la classe SoldeException et se contenter d'un affichage dans le bloc catch