```
Algorithme TP ALSDD

// Déclaration des structures
Structure Logement //structure de logement
    id_logement : Entier
    type : Chaîne[10]  // "studio", "f2", "f3", "f4"
    superficie : Réel
    quartier : Chaîne[50]
    distance : Réel // distance au quartier
    loyer : Réel
    etat : Booléen  // occupé ou non (true si occupé)
    suivant : Pointeur vers Logement
Fin Structure

Structure Locataire  //structure locataire
    id_locataire : Entier
    nom : Chaîne[20]
    prenom : Chaîne[20]
    telephone : Chaîne[15]
    suivant : Pointeur vers Locataire
Fin Structure

Structure Location  //STRUCTURE LOCATION
    id_location : Entier
    id_locataire : Entier
    id_logement : Entier
    date_debut : Entier  // AAAAMMJJ
    date_fin : Entier    // AAAAMMJJ
    suivant : Pointeur vers Location
Fin Structure

// Fonction pour calculer le loyer
Fonction calculer_loyer(type : Chaîne, superficie : Réel) : Réel
var
    LB, SM : Réel
debut
    Si type = "studio" Alors
        LB =15000
        SM = 20
    Sinon Si type = "f2" Alors
        LB = 20000
        SM= 45
    Sinon Si type = "f3" Alors
        LB = 30000
        SM = 65
    Sinon Si type = "f4" Alors
        LB = 45000
        SM = 85
```

```
// le calcul suive le tableau de tp par type de logement
    Sinon
        calculer_loyer=-1 // en cas d'erreur il s'affiche -1
    Fin Si
    calculer_loyer= LB + (superficie - SM) * 800 // loi donne dans le tp
Fin Fonction

// procedure pour ajouter un logement
procedure ajouter_logement(var liste : pointeur vers Logement)
var
   id : Entier
   id_existe : Booléen // pour voir si id est deja utilise
   temp : poniteur vers Logement
debut
   allouer_logement(nouveau)
   repeat
        id_existe = Faux // initialisation que il n'existe pas
        Afficher ("Entrez l'id du logement : ")
        Lire (id)
        temp : = *liste
        Tant que (temp <> NUL) Faire
            Si (id_log_val(temp)= id) Alors
                id_existe = Vrai // en cas de trouver que id est utilise
                Afficher ("id ", id, " est déjà utilisé, veuillez entrer un autre id")
            Fin Si
            temp = next(temp)
        Fin Tant que
      until ( id_existe<>true) //un nouveau id
// lire les informations :
       aff_val_id(nouveau,id)
       Afficher ("Entrez le type de logement (studio, f2, f3, f4) : ")
       Lire (*nouveau.type)
       Afficher ("Entrez la superficie en m2 : ")
       Lire (*nouveau.superficie)
       Afficher ("Entrez le quartier : ")
       Lire (*nouveau.quartier)
       Afficher ("Entrez la distance jusqu'à la commune : ")
       Lire (*nouveau.distance)
       aff_val_etat(nouveau, Faux) // logement n'est pas alloue
       aff_loyer_val(nouveau , calculer_loyer(type_val(nouveau),super_val (nouveau)))
// loyer utilisant la fonction calculer_loyer
       next(nouveau= *liste)
       *liste = nouveau

      Afficher ("Logement ajouté avec succès, Loyer = ",loyer_val( nouveau), " DA")
      logement_fichier(*liste)//ajouter au fichier
```

```
Fin


//procedure pour supprimer un logement
procedure  supprimer_logement(var liste : Pointeur vers Logement, var archive : Pointeur
vers Logement, id : Entier)
var
   temp, prev : Pointeur vers Logement
    temp = *liste
debut
      prev = NUL
    Tant que(temp <> NUL ET id_lod_val(temp) <>id) Faire // cherche du logement a suprimer
        prev = temp
        temp = next(temp)
     Fin Tant que
    Si (temp = NUL) Alors //cas non trouve
        Afficher ("Logement non trouvé")
   Sinon
    Si (prev = NUL) Alors//cas debut de liste
        *liste =next(temp)
    Sinon//cas au melieu
        next(prev) = next(temp)
    Fin Si
    aff_adr(temp , *archive) // deplacement vers l'archive
    *archive = temp
    Afficher ("Logement id : ", id, " supprimé et archivé")
    logement_archive(*archive)//ajouter au fichier
    suprimer_logement_fichier(*liste) // supprimer du fichier
Fin si
Fin

// procedure pour afficher un logement
procedure afficher_logement(liste : Pointeur vers Logement)
    Afficher( "ID: ", id_log_val(liste), " | ",type_val( liste), " | ",superficie_val( liste), " m2 | ",
*liste.quartier, " | ",dis_val( liste), " km | ",loyer_val( liste), " DA | ")
    Si (etat_val(liste) = Vrai) Alors // afficher si occupe ou non selon l'etat
        Afficher ("Le logement est occupé")
    Sinon
        Afficher ("Le logement est libre")
    Fin Si
Fin

// procedure pour afficher les logements
procedure afficher_logements(liste : Pointeur vers Logement)
 debut
    Afficher ("Liste des logements :")
    Tant que (liste <> NUL) Faire // jusqu'a la fin de la LLC
        afficher_logement(liste)//fonction affiche un logement
```

```
        liste = next(liste) //passer au suivant
    Fin Tant que
Fin

// procedure pour supprimer un type de logement
procedure supprimer_type_logement(var liste :Pointeur vers Logement,var archive :Pointeur
vers Logement, type : Chaîne)
var
    temp, prev : Pointeur vers Logement
    found : Entier
    temp = *liste
    prev = NUL
    a_supprimer : Pointeur vers Logement
debut
   found=false
    Tant que( temp <> NUL) Faire
        Si (type_val(temp) = type) Alors
            a_supprimer= temp // verifier l'egalite pour la suppression
            found =true
            Si (prev = NUL) Alors // si dans la tete
                *liste =next( temp)
            Sinon//si au melieu
                aff_adr(prev, next(temp))
             Fin Si
            temp = next(temp)
            aff_adr( a_supprimer,*archive)
            *archive = a_supprimer
            logement_archive(*archive) //ajouter au fichier
            Afficher( "Logement de type ", type, " supprimé et archivé")
        Sinon
            prev =temp
            temp = next(temp)
        Fin Si
    Fin Tant que
    Si (found =false) Alors
        Afficher ("Aucun logement du type ", type, " trouvé")
    Fin Si
    suprimer_logement_fichier(*liste)//suprimer du fichier
Fin

// procedure pour ajouter un locataire
procedure ajouter_locataire(var liste : pointeur vers Locataire)
  var
      nouveau : pointeur vers Locataire
      id : Entier
      id_existe : Booléen
  debut
    allouer(nouveau)
```

```
      repeat
       id_existe = Faux //si l'id existe deja
       Afficher ("Entrez l'id du locataire : ")
       Lire (id)
       temp : *Locataire = *liste
       Tant que (temp <> NUL) Faire //si l'id existe deja
          Si (id_val(temp) = id) Alors
             id_existe = Vrai
             Afficher ("id ", id, " est déjà utilisé, veuillez entrer un autre id")
          Fin Si
          temp = next(temp)
       Fin Tant que
   unitl(id_existe<>true)
//lire les informations
  aff_val_id( nouveau) = id
   Afficher ("Entrez le nom : ")
   Lire (*nouveau.nom)
   Afficher ("Entrez le prénom : ")
   Lire (*nouveau.prenom)
   Afficher ("Entrez le téléphone : ")
   Lire (*nouveau.telephone)
  aff_adr( nouveau) = *liste
   *liste = nouveau //ajouter a la list

   Afficher ("Locataire ajouté avec succès")
   locataire_fichier(*liste)//ajouter au fichier
Fin


// procedure pour supprimer un locataire
procedure supprimer_locataire(var liste : Pointeur vers Locataire,var archive : Pointeur vers
Locataire, id : Entier)
var
   temp, prev : Pointeur vers Locataire
   temp = *liste
   prev = NUL
debut
   Tant que (temp <> NUL ET id_val( temp)_locataire <> id )Faire //chercher le locataire a
suprimier
       prev = temp
       temp = next(temp)
   Fin Tant que
   Si (temp = NUL) Alors
      Afficher ("Locataire non trouvé")
   Fin Si
   Si (prev = NUL) Alors  //si a la tete
      *liste = next(temp)
   Sinon //si au melieu
     aff_adr( prev , next(temp))
```

```
    Fin Si
    aff_adr(temp , *archive)
    *archive = temp //metre dans la liste d'archive
    Afficher ("Locataire id : ", id, " supprimé et archivé")
    locataire_archive(*archive)
    suprimer_locataire_file(*liste)
Fin

//procedure pour afficher un locataire
procedure afficher_locataire(locataire : Pointeur vers Locataire)
    Afficher ("ID Locataire: ",id_loc_val( locataire), " | Nom: ", nom_val(locataire), " | Prénom:
",prenom_val( locatair)->prenom, " | Téléphone: ", tel_val(locataire))
Fin

//procedure pour afficher les locataires
procedure afficher_locataires(liste : Pointeur vers Locataire)
    Tant que (liste <> NUL) Faire
        afficher_locataire(liste)//fonction pour afficher un locataire
        liste = next(liste)
    Fin Tant que
Fin

// procedure pour changer l'état d'un logement
procedure changer_letat(liste : Pointeur vers Logement, id : Entier, state : Booléen)
var
    temp : Pointeur vers Logement = liste
debut
    Tant que (temp <> NUL) Faire
        Si (id_val(temp) = id) Alors
            aff_val_etat(temp)= state //changer l'etat
        Sinon
            temp =next( temp)
        Fin Si
    Fin Tant que
Fin

// procedure pour ajouter une location
procedure ajouter_location(var liste : Pointeur vers Location, logements : Pointeur vers
Logement)
var
    nouvelle : Pointeur vers Location
debut
   allouer(nouvelle)
//lire les informations
    Afficher( "Entrez l'ID de la location : ")
    Lire (*nouvelle.id_location)
    Afficher ("Entrez l'ID du locataire : ")
    Lire (*nouvelle.id_locataire)
```

```
    Afficher ("Entrez l'ID du logement : ")
    Lire (*nouvelle.id_logement)
    Afficher ("Entrez la date de début (AAAAMMJJ) : ")
    Lire (*nouvelle.date_debut)
    Afficher ("Entrez la date de fin (AAAAMMJJ) : ")
    Lire (*nouvelle.date_fin)
    changer_letat(logements, id_log_val(nouvelle), Vrai) //changer l'etat du logement
    aff_adr(nouvelle, *liste)
    *liste = nouvelle
    Afficher ("Location ajoutée avec succès")
    location_fichier(*liste) //ajouter au fichier
    suprimer_logement_fichier(logements) /modifier le fichier des logements
Fin

// Fonction pour supprimer une location
Fonction supprimer_location(var liste : Pointeur vers Location,var archive : Pointeur vers
Location, id : Entier, logements : Pointeur vers Logement)
var
    temp, prev : Pointeur vers Location
debut
    temp = *liste
    prev = NUL
    Tant que (temp <> NUL ET id_log_val(temp) <> id) Faire //chercher la location
        prev = temp
        temp = next( temp)
    Fin Tant que
    Si (temp = NUL) Alors
        Afficher ("Location non trouvée")
   Si non Si (prev = NUL) Alors //si a la tete
        *liste =next( temp)
    Sinon
        aff_adr(prev,next( temp)) //si au melieu
    Fin Si
    Fin si
    aff_adr(temp ,*archive)
    *archive = temp
    changer_letat(logements, *temp.id_logement, Faux) //changer l'etat
    Afficher ("Location id : ", id, " supprimée et archivée")
    location_archive(*archive)  //ajouter au fichier archive
    suprimer_location_file(*liste) //suprimer du fichier
    suprimer_logement_fichier(logements)//modifier le fichier des logements
Fin

// procedurepour afficher les logements occupés
procedure afficher_occupe(liste : Pointeur vers Logement)
var
    temp : Pointeur vers Logement
    cpt : Entier
```

```
debut
 temp = liste
    cpt = 0
    Tant que (temp <> NUL) Faire
        Si (etat_val(temp)= Vrai) Alors
            afficher_logement(temp) //fonction pour afficher
            cpt = cpt + 1 //incrementer le compteur
        Fin Si
        temp =next( temp)
    Fin Tant que
    Si (cpt = 0) Alors
        Afficher ("Il n'y a pas de logements occupés")
    Fin Si
Fin

// procedure pour afficher les logements libres à une date donnée
procedure afficher_libre(liste : Pointeur vers Location, logements : Pointeur vers Logement)
var
    date : Entier
     save : Pointeur vers Logement
debut
    Afficher ("Entrez la date (AAAAMMJJ) : ")
    Lire (date)
    save = logements
    Tant que (logements <> NUL) Faire //afficher les logements libre
        Si(etat_val( logements)= Faux) Alors
            afficher_logement(logements)
        Fin Si
        logements =next(logements)
    Fin Tant que
    Tant que (liste <> NUL )Faire //afficher les logements qui seront libres
        Si(date_fin_val( liste)< date) Alors
            afficher_par_id(save, id_log_val(liste)) //afficher
        Fin Si
        liste =next( liste)
    Fin Tant que
Fin

// procedure pour afficher les locataires occupant un logement avec superficie supérieure à
la moyenne
procedure afficher_locataires_superficie(locations : Pointeur vers Location, logements :
Pointeur vers Logement, locataires : Pointeur vers Locataire)
  var
    loc : Pointeur vers Location
   log : Pointeur vers Logement
   locataire : Pointeur vers Locataire
debut
  Afficher( "Locataires occupant un logement avec une superficie supérieure à la moyenne :")
```

```
    loc = locations
    Tant que( loc <>NUL) Faire
        log = logements
        Tant que (log <> NUL) Faire
            Si (id_val(log) = id_val(loc)) Alors
                moyenne : Réel
                Si (type_val(log) = "studio" )Alors
                    moyenne =20
                Sinon Si (type_val(log) = "f2") Alors
                    moyenne =45
                Sinon Si(type_val( log) = "f3") Alors
                    moyenne = 65
                Sinon Si (type_val(log) = "f4") Alors
                    moyenne = 85
                Fin Si
                Si (superficie_val(log) > moyenne) Alors
                    locataire = locataires
                    Tant que (locataire <>NUL) Faire
                        Si (id_val(locataire) = id_val(loc) Alors
                            afficher_locataire(locataire)
                        Fin Si
                        locataire =next( locataire)
                    Fin Tant que
                Fin Si
            Fin Si
            log =next( log)
        Fin Tant que
        loc =next( loc)
    Fin Tant que
Fin

// fonction pour déterminer le type d'un logement
fonction the_type(log : Pointeur vers Logement, id : Entier) : Entier
var
    result : Entier
debut
  result = 0
    Tant que( log <> NUL) Faire //rendre 1pour studio 2 =f2 3 =f3 4=f4
        Si(id_log_val( log) = id )Alors
            Si (type_val(log)= "studio") Alors
                result =1
            Sinon Si( type_val(log))= "f2" Alors
                result = 2
            Sinon Si (type_val(log)= "f3" )Alors
                result = 3
            Sinon Si (type_val(log)= "f4") Alors
                result = 4
```

```
            Fin Si
        Fin Si
        log = next(log)
    Fin Tant que
the_type= result //retourner le resultat
Fin Fonction

// procedure pour séparer les locations par type
procedure location_type(var location : Pointeur vers Location,var kind :Pointeur vers
Location, type : Entier, logements : Pointeur vers Logement)
var
    prev : Pointeur vers Location = NUL
    save : Pointeur vers Location = *location
debut
    Tant que (save <> NUL)Faire
        Si (the_type(logements,id_log_val( save)) = type) Alors
            Si (prev = NUL) Alors //si a la tete de la liste
                *location =next( save)
                aff_adr(save, *kind) // deplacer la location
                *kind = save
                save = *location
            Sinon  //au melieu
                aff_adr(prev ,next( save))
                aff_adr(save, *kind)  //deplacer la location
                *kind = save
                save =next( prev)
            Fin Si
        Sinon //passer a la suivante
            prev =save
            save =next( save)
        Fin Si
    Fin Tant que
Fin

// Fonction pour obtenir le loyer d'un logement
Fonction rent(log : Pointeur vers Logement, id : Entier) : Entier
var
    result : Entier
debut
 result=0
    Tant que (log <> NUL) Faire
        Si(id_val( log) = id )Alors
            result =loyer_val( log)
        Fin Si
        log =next( log)
    Fin Tant que
    rent= result
Fin Fonction
```

```
// procedure pour trouver la location avec le loyer le plus élevé
procedure  highest_rent(list : Pointeur vers Location,var prec : Pointeur vers Location,var
proch : Pointeur vers Location, log : Pointeur vers Logement)
var
   prev : Pointeur vers Location = NUL
   save : Entier = 0
debut
   Tant que( list <> NUL) Faire
      Si( rent(log, id_val(list)) > save) Alors //comparer le loyer
         save = rent(log,id_val( list))
         *prec = prev //save le plus grand
         *proch = list
      Fin Si
      prev = list
      list =next( list)
   Fin Tant que
Fin

// procedure pour trier les locations par loyer
procedure sort_location(var list : Pointeur vers Location, log : Pointeur vers Logement)
var
   prec, proch : Pointeur vers Location
   save : Pointeur vers Location = NUL
debut
   Tant que (*list <> NUL) Faire
      highest_rent(*list, prec, proch, log)
      Si prec = NUL Alors //si a la tete de la liste
         *list =next(proch)
         aff_adr(proch, save) //deplacer la location vers nouvelle liste
         save = proch
      Sinon // au melieu
        aff_adr( prec,next( proch))
         aff_adr(proch, save)
         save = proch //deplacer la location
      Fin Si
   Fin Tant que
   *list = save // affecter la tete de la nouvelle liste
Fin

// procedure pour fusionner deux listes de locations
procedure merge_location(var list : Pointeur vers Location,var other :  Pointeur vers
Location, log : Pointeur vers Logement)
var
   tete, last : Pointeur vers Location //pour la nouvelle liste tete et fin
    save : Pointeur vers Location
debut
   tete=NUL
```

```
    last= NUL
   Tant que (*list <> NUL ET *other <>NUL) Faire
      Si (rent(log, id_log_val((*list)))<rent(log, id_log_val((*other))) Alors //comparer le loyer
         save = *list
         *list =next( (*list))
      Sinon
         save = *other
         *other =next( (*other))
      Fin Si
     //deplacer la location vers une nouvelle liste
      Si (tete = NUL )Alors //si la liste est vide
         tete= save
      Sinon //non vide
         aff_adr(last,save)
      Fin Si
      last= save
   Fin Tant que
   Si (*other=NUL) Alors //si la liste est finie lier l'autre liste
      Si (tete = NUL) Alors //si la nouvelle est vide
         tete= *list
      Sinon
        aff_adr( last, *list)
      Fin Si
  Si (*list= NUL) Alors  //si la liste est finie lier l'autre liste
      Si (tete = NUL) Alors //la liste est vide
         tete= *other
      Sinon
         aff_adr(last, *other)
      Fin Si
   Fin Si
   *list =tete //affecter la tete de la nouvelle liste
Fin

// procedure pour afficher les locations
procedure afficher_location(list : Pointeur vers Location)
debut
   Tant que (list <> NUL) Faire
      Afficher ("Id location : ",id_val( list), " | Id logement : ", id_val(list) " | Id locataire : ",
id_val(list), " | Date de début : ",date_d_val( list) " | Date de fin : ",date_f_val( list))
      list =next( list)
   Fin Tant que
Fin

// procedure pour trouver le logement avec la plus grande distance
procedure highest_distance(log : Pointeur vers Logement,var prec : Pointeur vers Logement,
var proch :  Pointeur vers Logement)
var
   prev : Pointeur vers Logement
```

```
    save : Entier
debut
     prev = NUL
    save = -1
    Tant que (log <> NUL) Faire //chrcher la plus grande
        Si(distance_val( log) > save) Alors
            save =distance_val( log)
            *prec = prev //sauvgarder le logement
            *proch = log
        Fin Si
        prev = log
        log =next(log)
    Fin Tant que
Fin

// procedure pour trier les logements par distance
procedure sort_par_distance(var list : Pointeur vers Logement)
var
    prev, proch : Pointeur vers Logement
    save : Pointeur vers Logement //pour une nouvelle liste
debut
    save = NUL
    Tant que( *list <>NUL) Faire
        highest_distance(*list, prev, proch) //chercher la plus grande et la deplacer
        Si (prev = NUL) Alors //a la tete
            *list =next( proch)
           aff_adr( proch, save)
            save =proch
        Sinon //au melieu
            aff_adr(prev,next( proch)
            aff_adr(proch, save)
            save = proch
        Fin Si
    Fin Tant que
    *list =save //affecter la tete de la nouvelle liste
Fin

// procedure pour afficher les logements les plus proches d'une commune
procedure proche_commune(var log : Pointeur vers Logement)
var
    cpt : Entier
    save : Pointeur vers Logement
debut
    Afficher ("Entrez le nombre de logements à afficher : ")
    Lire (cpt)
    save = *log
    sort_par_distance(save) //trier la liste
    *log = save
```

```
    Tant que (save <>NUL ET cpt > 0) Faire //afficher
        afficher_logement(save)
        cpt = cpt - 1
        save =next( save)
    Fin Tant que
Fin

// procedure pour trouver le logement avec le loyer le plus élevé
procedure highest_loyer(log : Pointeur vers Logement,var prec : Pointeur vers Logement,
var proch : Pointeur vers Logement)
var
    prev : Pointeur vers Logement
    save : Entier
debut
 prev = NUL
    save = -1
    Tant que( log <> NUL) Faire
        Si (loyer_val(log) > save) Alors //comparer et sauver la plus grande
            save =loyer_val( log)
            *prec =prev
            *proch = log
        Fin Si
        prev = log
        log= next( log)
    Fin Tant que
Fin

// procedure pour trier les logements par loyer
procedure sort_par_loyer(var list :  Pointeur vers Logement)
var
    prev, proch : Pointeur vers Logement
    save : Pointeur vers Logement
debut
    save = NUL
    Tant que (*list <> NUL )Faire
        highest_loyer(*list, prev, proch)  //chercher la plus grande et la deplacer
        Si( prev = NUL )Alors  // a la tete de la liste
            *list =next( proch)
           aff_adr( proch,save)
            save = proch
        Sinon //au melieu
            aff_adr(prev,next( proch))
            aff_adr(proch, save)
            save =proch
        Fin Si
    Fin Tant que
    *list = save //affecter la tete de la nouvelle liste
Fin
```

```
// procedure pour afficher les logements avec le loyer minimal
procedure minimal_loyer(var log : Pointeur vers Logement)
var
   cpt : Entier
   save : Pointeur vers Logement
debut
   Afficher ("Entrez le nombre de logements à afficher : ")
   Lire (cpt)
   save =*log
   sort_par_loyer(save)  //trier la liste
   *log =save
   Tant que (save <> NUL ET cpt > 0) Faire //afficher
      afficher_logement(save)
      cpt = cpt - 1
      save =next( save)
   Fin Tant que
Fin

// Fonction pour extraire l'année d'une date
Fonction extract_year(date : Entier) : Entier
   extract=date div  10000
Fin Fonction

// procedure pour compter les logements loués par quartier pour une année donnée
procedure count_logements_by_quartier_and_year(logements : Pointeur vers Logement,
archive_logements : Pointeur vers Logement, locations : Pointeur vers Location,
archive_locations : Pointeur vers Location, year : Entier)
var
   log : Pointeur vers Logement
   loc : Pointeur vers Location
   count_by_quartier : Tableau[100] de Entier
   quartiers : Tableau[100][50] de Chaîne
   quartier_count : Entier
     temp : Pointeur vers Location
    found : bool
debut
   quartier_count  = 0
   log = logements
   loc= locations
   // Vérifier les logements actifs et les locations
   Tant que (log <> NUL) Faire
      temp = locations
      Tant que (temp <> NUL) Faire
         Si (id_val(temp) = id_val(log) ET extract_year(date_d_val(temp)) = year Alors
            found = false
            Pour i de 0 à quartier_count - 1 Faire
               Si( quartiers[i] = *log.quartier) Alors
```

```
                count_by_quartier[i] = count_by_quartier[i] + 1
                found =true
            Fin Si
        Fin Pour
        Si( found = false) Alors
            quartiers[quartier_count] = *log.quartier
            count_by_quartier[quartier_count] =count_by_quartier[quartier_count] +1
            quartier_count = quartier_count + 1
        Fin Si
    Fin Si
    temp =next( temp)
  Fin Tant que
  log =next( log)
Fin Tant que

// Vérifier les logements archivés et les locations archivées
log =archive_logements
Tant que( log <> NUL) Faire
  temp  = archive_locations
  Tant que (temp <> NUL) Faire
    Si (id_val(temp) = id_val(log) ET extract_year(date_d_val(temp)) = year) Alors
      found : bool=false
      Pour i de 0 à quartier_count - 1 Faire
        Si (quartiers[i] = quartier_val(log)) Alors
          count_by_quartier[i] = count_by_quartier[i] + 1
          found =true
        Fin Si
      Fin Pour
      Si (found =false) Alors
        quartiers[quartier_count] =quarier_val(log)
        count_by_quartier[quartier_count] =count_by_quartier[quartier_count]+ 1
        quartier_count = quartier_count + 1
      Fin Si
    Fin Si
    temp =next( temp)
  Fin Tant que
  log =next( log)
Fin Tant que

// Afficher les résultats
Afficher ("Logements loués en ", year, ", classés par quartier :")
Pour i de 0 à quartier_count - 1 Faire
  Afficher( "Quartier : ", quartiers[i], " | Nombre de logements loués : ",
count_by_quartier[i])
Fin Pour
Fin Fonction

// procedure pour compter les locations par type pour une année donnée
```

```
procedure count_locations_by_type_and_year(locations : Pointeur vers Location,
archive_locations : Pointeur vers Location, logements : Pointeur vers Logement,
archive_logements : Pointeur vers Logement, year : Entier)
var
   loc : Pointeur vers Location
   studio_count, f2_count, f3_count, f4_count : Entier
debut
 loc = locations
   studio_count, f2_count, f3_count, f4_count = 0
   // Vérifier les locations actives et les logements
   Tant que (loc <> NUL) Faire
      Si (extract_year(date_d_val(loc) = year) Alors
         log : Pointeur vers Logement = logements
         Tant que( log <> NUL) Faire
            Si (id_val(log) = id_val(loc)) Alors // si id est trouve alors +1 pour son type
               Si (type_val(log)= "studio") Alors
                  studio_count = studio_count + 1
               Sinon Si (type_val(log)= "f2") Alors
                  f2_count = f2_count + 1
               Sinon Si (type_val(log) = "f3") Alors
                  f3_count = f3_count + 1
               Sinon Si(type_val( log) = "f4") Alors
                  f4_count = f4_count + 1
               Fin Si
            Fin Si
            log =,ext( log)
         Fin Tant que
      Fin Si
      loc =next( loc)
   Fin Tant que
   // Vérifier les locations archivées et les logements archivés
   loc =archive_locations
   Tant que (loc <> NUL )Faire
      Si( extract_yeardate_d_val(loc)) = year) Alors
         log : Pointeur vers Logement = archive_logements
         Tant que( log <> NUL) Faire
            Si (id_val(log) = id_val(loc)) Alors
               Si(tpe_val( log) = "studio") Alors
                  studio_count =studio_count + 1
               Sinon Si (type_val(log) = "f2" )Alors
                  f2_count = f2_count + 1
               Sinon Si (type_val(log)= "f3" )Alors
                  f3_count = f3_count + 1
               Sinon Si (type_val(log)= "f4") Alors
                  f4_count = f4_count + 1
               Fin Si
            Fin Si
            log next( log)
```

```
            Fin Tant que
        Fin Si
        loc =next( loc)
    Fin Tant que
    // Afficher les résultats
    Afficher ("Total des locations par type en ", year, " :")
    Afficher ("Studio : ", studio_count)
    Afficher ("F2 : ", f2_count)
    Afficher ("F3 : ", f3_count)
    Afficher ("F4 : ", f4_count)
Fin Fonction
// procedure pour gérer les logements dans le menu
procedure menu_logements(var logements : Pointeur vers Logement,var archive_logements :
Pointeur vers Logement)
var
    choix, id : Entier
    type_logement : Chaîne[10]
debut
repeat
        Afficher ("=== GESTION DES LOGEMENTS ===")
        Afficher ("1. Ajouter un logement")
        Afficher ("2. Supprimer un logement")
        Afficher ("3. Supprimer un type de logement")
        Afficher ("4. Afficher les logements")
        Afficher ("5. Retour au menu principal")
        Afficher ("Votre choix : ")
        Lire (choix)
       cas de (choix) Faire
            Cas 1:
                ajouter_logement(logements)
            Cas 2:
                Afficher ("Entrez l'ID du logement à supprimer : ")
                Lire (id)
                supprimer_logement(logements, archive_logements, id)
            Cas 3:
                Afficher ("Entrez le type de logement à supprimer (studio, f2, f3, f4) : ")
                Lire (type_logement)
                supprimer_type_logement(logements, archive_logements, type_logement)
            Cas 4:
                afficher_logements(*logements)
            Cas 5:
              sortir
            Défaut:
                Afficher ("Choix invalide. Veuillez réessayer.")
        Fin Selon
         until( choix = 5)
Fin

// procedure pour gérer les locataires dans le menu
```

```
procedure menu_locataires(var locataires : Pointeur vers Locataire,var archive_locataires :
Pointeur vers Locataire)
var
   choix, id : Entier
debut
   repeat
      Afficher( "=== GESTION DES LOCATAIRES ===")
      Afficher ("1. Ajouter un locataire")
      Afficher ("2. Supprimer un locataire")
      Afficher ("3. Afficher les locataires")
      Afficher ("4. Retour au menu principal")
      Afficher ("Votre choix : ")
      Lire (choix)

      cas de ( choix) Faire
         Cas 1:
            ajouter_locataire(locataires)
         Cas 2:
            Afficher( "Entrez l'ID du locataire à supprimer : ")
            Lire( id)
            supprimer_locataire(locataires, archive_locataires, id)
         Cas 3:
            afficher_locataires(*locataires)
         Cas 4:
           sorir
         Défaut:
            Afficher( "Choix invalide. Veuillez réessayer.")
      Fin Seloni
   until (choix = 4)
Fin
// procedure pour gérer les locations dans le menu
procedure menu_locations(var locations :  Pointeur vers Location,var archive_locations :
Pointeur vers Location, logements : Pointeur vers Logement,var locataires : Pointeur vers
Locataire)
var
   choix, id : Entier
debut
   repeat
      Afficher ("=== GESTION DES LOCATIONS ===")
      Afficher( "1. Ajouter une location")
      Afficher( "2. Supprimer une location")
      Afficher ("3. Afficher les logements occupés")
      Afficher ("4. Afficher les logements libres à une date")
      Afficher ("5. Afficher les locataires occupant un logement de même type avec superficie
moyenne")
      Afficher ("6. Retour au menu principal"
      Afficher ("Votre choix : ")
      Lire (choix)
      cas de (choix) Faire
         Cas 1:
```

```
            ajouter_location(locations, logements)
        Cas 2:
            Afficher ("Entrez l'ID de la location à supprimer : ")
            Lire (id)
            supprimer_location(locations, archive_locations, id, logements)
        Cas 3:
            afficher_occupe(logements)
        Cas 4:
            afficher_libre(*locations, logements)
        Cas 5:
            afficher_locataires_superficie(*locations, logements, *locataires)
        Cas 6:
            sortir
        Défaut:
            Afficher( "Choix invalide. Veuillez réessayer.")
    Fin Selon
  until( choix = 6)
Fin Fonction

// procedure pour fusionner et trier les locations dans le menu
procedure fusion_et_tris(var locations :Pointeur vers Location,var studio : Pointeur vers Location,
var f2 : Pointeur vers Location,var f3 :Pointeur vers Location, var f4 :Pointeur vers Location,
logements : Pointeur vers Logement)
var
  choix : Entier
debut
  repeat
    Afficher ("=== FUSION ET TRIS ===")
    Afficher( "1. Séparer les locations par type")
    Afficher ("2. Trier les locations")
    Afficher ("3. Fusionner les locations")
    Afficher ("4. Afficher toutes les locations")
    Afficher ("5. Retour au menu principal")
    Afficher ("Votre choix : ")
    Lire (choix)
    cas de(choix) Faire
      Cas 1:
        location_type(locations, studio, 1, logements)
        location_type(locations, f2, 2, logements)
        location_type(locations, f3, 3, logements)
        location_type(locations, f4, 4, logements)
        Afficher ("Studios :")
        afficher_location(*studio)
        Afficher( "F2 :")
        afficher_location(*f2)
        Afficher( "F3 :")
        afficher_location(*f3)
        Afficher( "F4 :")
        afficher_location(*f4)
      Cas 2:
```

```
            sort_location(studio, logements)
            sort_location(f2, logements)
            sort_location(f3, logements)
            sort_location(f4, logements)
            Afficher( "Studios :")
            afficher_location(*studio)
            Afficher( "F2 :")
            afficher_location(*f2)
            Afficher ("F3 :")
            afficher_location(*f3)
            Afficher ("F4 :")
            afficher_location(*f4)
            Afficher ("Listes triées avec succès.")
        Cas 3:
            merge_location(locations, studio, logements)
            merge_location(locations, f2, logements)
            merge_location(locations, f3, logements)
            merge_location(locations, f4, logements)
            Afficher ("Listes fusionnées avec succès.")
        Cas 4:
            afficher_location(*locations)
        Cas 5:
            sortir
        Défaut:
            Afficher ("Choix invalide. Veuillez réessayer.")
    Fin Selon
  until(choix = 5)
Fin Fonction

// procedure pour la recherche dans le menu
procedure recherche(var logements : Pointeur vers Logement)
var
  choix : Entier
debut
  repeat
    Afficher( "=== RECHERCHE ===")
    Afficher ("1. Rechercher les logements proches d'une commune")
    Afficher ("2. Rechercher les logements avec loyer minimal")
    Afficher ("3. Retour au menu principal")
    Afficher ("Votre choix : ")
    Lire (choix)

    cas de(choix) Faire
        Cas 1:
            proche_commune(logements)
        Cas 2:
            minimal_loyer(logements)
        Cas 3:
            sortir
        Défaut:
```

```
            Afficher ("Choix invalide. Veuillez réessayer.")
        Fin Selon
    until( choix = 3)
Fin
// procedure pour l'historique dans le menu
procedure historique(var logements :Pointeur vers Logement,var archive_logements : Pointeur
vers Logement,var locations : Pointeur vers Location,var archive_locations : Pointeur vers
Location)
var
    choix, year : Entier
debut
    debut
        Afficher( "=== HISTORIQUE ===")
        Afficher ("1. Consulter l'historique des logements par année")
        Afficher( "2. Consulter l'historique des locations par année")
        Afficher( "3. Retour au menu principal")
        Afficher ("Votre choix : ")
        Lire (choix)
        cas de(choix) Faire
            Cas 1:
                Afficher ("Entrez l'année (AAAA) : ")
                Lire (year)
                count_logements_by_quartier_and_year(*logements, *archive_logements, *locations,
*archive_locations, year)
            Cas 2:
                Afficher ("Entrez l'année (AAAA) : ")
                Lire (year)
                count_locations_by_type_and_year(*locations, *archive_locations, *logements,
*archive_logements, year)
            Cas 3:
                sortir
            Défaut:
                Afficher ("Choix invalide. Veuillez réessayer.")
        Fin Selon
    until (choix = 3)
Fin
// procedure principale menu
procedure menu()
var
     choix : Entier
    // Initialisation des listes
    logements, archive_logements : Pointeur vers Logement = NUL
    locataires, archive_locataires : Pointeur vers Locataire = NUL
    locations, archive_locations : Pointeur vers Location =NUL
    studio, f2, f3, f4 : Pointeur vers Location = NUL
debut
    // Chargement des informations depuis les fichiers
    create_file()
    file_logement(logements)
    archive_logement(archive_logements)
```

```
    file_locataire(locataires)
    archive_locataire(archive_locataires)
    file_location(locations)
    archive_location(archive_locations)
    repeat
        Afficher ("----- MENU PRINCIPAL -----")
        Afficher ("1. Gestion des logements")
        Afficher ("2. Gestion des locataires")
        Afficher ("3. Gestion des locations")
        Afficher ("4. Fusion et tris")
        Afficher ("5. Recherche")
        Afficher ("6. Historique")
        Afficher ("7. Quitter")
        Afficher ("Votre choix : ")
       Lire (choix)
        cas de (choix) Faire
            Cas 1:
                menu_logements(logements, archive_logements)
            Cas 2:
                menu_locataires(locataires, archive_locataires)
            Cas 3:
                menu_locations(locations, archive_locations, logements, locataires)
            Cas 4:
                fusion_et_tris(locations, studio, f2, f3, f4, logements)
            Cas 5:
                recherche(logements)
            Cas 6:
                historique(logements, archive_logements, locations, archive_locations)
            Cas 7:
                Afficher ("Au revoir !")
            Défaut:
                Afficher ("Choix invalide. Veuillez réessayer.")
        Fin cas de
            until (choix = 7)
Fin

// Programme principal
Algorithme Principal()
    menu() // appel de fonction de menu
Fin Algorithme
```