```
-* principal
langueLue
                                 // y prévoir les textes à imprimer dans les 2 langues
           ↓ langue
                                // (le fichier langues n'étant pas encore chargé ☺)
                 ↓ langue
                               // chargement des messages du fichier langues dans une LCS
chargerLexique
                 ↓ codeErreur, pLexique
- if (codeErreur ≠ PAS_D_ERREUR)
                                          // fichier non trouvé ou pas assez de mémoire
                         ↓ codeErreur
 imprimerMessageErreur
                                // prévoir les textes à imprimer dans les 2 langues
else
                          // initialisation des shifts dans la LCS via le fichier adéquat
 initialiserShifts
                     ↓ codeErreur, pDebShifts
 - if (codeErreur ≠ PAS_D_ERREUR)
                                           // impossible d'avoir la LC des shifts
                          ↓ codeErreur
  imprimerMessageErreur
  else
                           // initialisation des membres dans un tableau membres
  chargementMembres
                      ↓ codeErreur, membres, nbMembres
  if (codeErreur ≠ PAS D ERREUR)
                                           // sans membre, pas d'exécution adéquate
                            ↓ codeErreur
   imprimerMessageErreur
   else
               ↓ pLexique, pDebShifts, membres, nbMembres
   dialogues
                              // Le lexique peut avoir été chargé en partie ou totalement
if (pLexique ≠ NIL)
                 ↓ pLexique
libérerLexique
```

```
/***** LEXIQUE *****/
   -* chargerLexique
                      ı↓ langue
    nomFichierLangage
                      ↓ nomFichier
 pLexique = NIL
                                        // adresse début de la liste chaînée des messages
 ouvrir le fichier en mode lecture
   - if (erreur ouverture)
  codeErreur = ERREUR_OUVERTURE_FICHIER
   - else
  codeErreur = PAS_D_ERREUR
     messageLu
               」↓ num, texte
    = do while (NOT fin de fichier AND codeErreur = PAS D ERREUR)
   pMessage = allocation de mémoire
     - if (pMessage = NIL)
    codeErreur = ERREUR MEMOIRE
      - else
     pMessage→num = num
     pMessage→texte = texte
     pMessage→pSuiv = NIL
       - if (pLexique = NIL)
      pLexique = pMessage
       - else
      pLexiquePrec→pSuiv = pMessage
     pLexiquePrec = pMessage
       messageLu
                  ↓ num, texte
  fermer le fichier
```

Les modules suivants concernent le lexique et seront appelés par d'autres modules que chargerLexique.

```
* nomFichierLangage

if (langue = FRANCAIS)
fichierLexique = "Francais.txt"

else
fichierLexique = "English.txt"

* messageTrouvé
pMessage = pLexique

do while (pMessage ≠ NIL AND numMessage > pMessage→num)
pMessage= pMessage→pSuiv
```

```
plexique, numMessage

plexique, numMessage

pmessageTrouvé

pmessage

pmess
```

-* afficherMessage

```
/***** INTERFACE *****/
   * dialogues
                 ↓ pLexique, debMenu = MENU_PRINCIPAL¹
   choixObtenu
                 ↓ choix
   = do while (choix ≠ QUITTER)
    - if (choix = AJOUTER_INSCRIPTION)
                           ↓ pLexique, pDebShifts, membres, nbMembres
     ajouterInscription
                          <sup>l</sup>↓ codeErreur
    - if (choix = SUPPRIMER_INSCRIPTION)
                             ↓ pLexique, pDebShifts, membres, nbMembres
     supprimerInscription
                             ↓ codeErreur
     if (choix = IMPRIMER_SHIFTS)
                             ↓ pLexique, pDebShifts, membres, nbMembres
      imprimerInscriptions
                            <sup>l</sup>↓ codeErreur
    - if (codeErreur ≠ PAS_D_ERREUR)
   numMessage = codeErreur + PREMIERE_ERREUR
                       ↓ pLexique, numMessage
      afficherMessage
    - if (codeErreur ≠ ERREUR_MEMOIRE and codeErreur ≠ ERREUR_OUVERTURE_FICHIER)
                   ↓ pLexique, debMenu = MENU_PRINCIPAL
      choixObtenu
                   ↓ choix
                  ↓ pDebShifts
   sauverShifts
                  ↓ codeErreur
                   ↓ pDebShifts
   libérerShifts
```

¹ Numéro de référence de ce texte dans le fichier de langues correspondant

```
−* choixObtenu
 = do
                 ↓ pLexique, debMenu
   afficherMenu
                 ↓ maxChoix
                     ↓ pLexique, numMessage = OBTENTION_CHOIX
   afficherMessage
 obtenir choix
 choixValide = choix >= PREMIER_CHOIX AND choix <= maxChoix</pre>
    // PREMIER CHOIX est une constante dont la valeur correspond au premier élément du menu
   if (NOT choixValide)
                     ↓ pLexique, numMessage = PREMIERE_ERREUR + MAUVAIS_CHOIX
    afficherMessage
  while (NOT choixValide)
 -* afficherMenu
                 ↓ pLexique, numMessage = debMenu
  messageTrouvé
                 ↓ pDebMenu
 sortir pDebMenu→texte
                 ↓ pDebMenu->pSuiv, debMenu
  nbChoixMenu
                 ↓ maxChoix
numChoix = 1
pMessage= pDebMenu→pSuiv
 = do while (pMessage ≠ NIL AND numChoix <= maxChoix)</pre>
 sortir numChoix, ". ", pMessage→texte
 numChoix++
 pMessage= pMessage→pSuiv
```

Exemple :

1001MENU PRINCIPAL 1002AJOUTER UNE INSCRIPTION 1003SUPPRIMER UNE INSCRIPTION 1004IMPRIMER 1005QUITTER

Donnera :

MENU PRINCIPAL

- 1. Ajouter une inscription
- 2. Supprimer une inscription...

```
-* shiftObtenu <sup>2</sup>
codeErreur = PAS_D_ERREUR
 — do
                     ↓ pLexique, numMessage = OBTENTION_DATE
   afficherMessage
 obtenir date
                    <sub>1</sub>↓ pLexique, numMessage = = OBTENTION_HEURE
   afficherMessage
 obtenir heure
  // module qui permet de détecter si le shift existe et renvoie aussi son adresse mémoire
                 ↓ pDebShifts, date, heure
   shiftTrouvé
                 ↓ shiftExiste, pShift
   - if (NOT shiftExiste)
                      ↓ pLexique, numMessage = PREMIERE_ERREUR + SHIFT_INCONNU
    afficherMessage
   while (NOT shiftExiste)
 —* matriculeObtenu
codeErreur = PAS_D_ERREUR
                 ↓ membres, nbMembres
  listerMembre
 = do
                     ↓ pLexique, numMessage = OBT_MATRICULE
   afficherMessage
 obtenir matricule
                  ↓ membres, nbMembres, matricule
   membreTrouvé
                  ↓ membreExiste, indMembre
   - if (NOT membreExiste)
                      ↓ pLexique, numMessage= PREMIERE_ERREUR + MEMBRE_INCONNU
    afficherMessage
 while (NOT membreExiste)
moy = membres[indMembre].moyenne
```

² Les deux modules de cette page attendent que l'utilisateur entre des données cohérentes. A vous d'adapter votre programme si vous souhaitez quitter les modules à un moment donné.

```
-* ajouterInscription<sup>3</sup>
codeErreur = PAS_D_ERREUR
                               ↓ pDebShifts
 afficherShiftsIncomplets
                               ↓ codeErreur
  if (codeErreur # PLUS_DE_PLACE_SHIFTS)
                                         // ce module crée un chaînon doublette vide
  nouvelleDoublette
                       <sup>」</sup>↓ codeErreur, pNouvDoublette
   - if (codeErreur ≠ ERREUR_MEMOIRE)
                  ↓ pLexique, pDebShifts
   shiftObtenu
                  ↓ pShift
                       ↓ pLexique, pDebShifts, membres, nbMembres
   matriculeObtenu
                      <sup>l</sup>↓ matricule1, moy1
                        <sub>l</sub>↓ pDebShifts, matricule1
   membreDéjàInscrit
                        <sup>l</sup>↓ déjàInscrit
    - if (NOT déjàInscrit)
                       ↓ pLexique, pDebShifts, membres, nbMembres
    matriculeObtenu
                       <sup>」</sup>↓ matricule2, moy2
                          ↓ pDebShifts, matricule2
    membreDéjàInscrit
                         <sup>J</sup>↓ déjàInscrit
      if (NOT déjàInscrit)
                    ↓ moy1, moy2
       categorie
                    ↓ categorie
                                  <sub>l</sub>↓ pShift, matricule1, matricule2, categorie,
        ajouterDoubletteShift
                                                                 pNouvDoublette
                          ↓ pShift, PLUS
        majNbDoublette
```

³ Ce DA est à ajuster si vous ajustez les modules shiftObtenu et matriculeObtenu !

```
-* supprimerInscription<sup>4</sup>
                  ↓ pLexique, membres, nbMembres
matriculeObtenu
                  ↓ matricule1, moy
                  ↓ pLexique, membres, nbMembres
matriculeObtenu
                  ↓ matricule2, moy
                  ↓ pDebShifts, matricule1, matricule2
doubletteTrouvée
                  ↓ estDoublette, pShift, pDoublette, pDoubletteSauv
  if (estDoublette)
                       ↓ pShift, pDoublette, pDoubletteSauv
  supprimerDoublette
                   ↓ pShift, MOINS
  majNbDoublette
-* imprimerInscriptions<sup>5</sup>
                            ↓ pLexique, pDebShifts, membres, nbMembres
afficherShiftsDoublettes
```

 $^{^4}$ Même remarque : à adapter si vous adaptez le module matriculeObtenu ; de plus, nous n'avons prévu un message si la doublette n'existe pas !

⁵ Adaptez vos messages quand les listes sont vides

```
/***** FICHIER ******/
   -* detectionFichierShifts
    codeErreur = PAS D ERREUR
    ouvrir Inscriptions en lecture
    if (erreur d'ouverture)
    premiereFois = true
    ouvrir DatesOrgShifts en lecture
       - if (erreur d'ouverture)
         codeErreur = ERREUR_OUVERTURE_FICHIER
         fermer DatesOrgShifts
    else
    premiereFois = false
    fermer Inscriptions.dat
   -* initialiserShifts
  detectionFichierShifts
                           ↓ premiereFois, codeErreur
   - if (codeErreur = PAS_D_ERREUR)
     - if (premiereFois)
      chargementDatesOrgShifts
                                 ↓ codeErreur, pDebShifts
     – else
      chargementInscriptions
                               ↓ codeErreur, pDebShifts
```

```
—* chargementDatesOrgShifts
ouvrir DatesOrgShifts.dat en mode lecture
codeErreur = PAS_D_ERREUR
pDebShifts = NIL
pShiftSauv = NIL
  shiftLu
         — ↓ date, heure
  = do while (NOT fin de fichier and codeErreur = PAS_D_ERREUR)
   nouveauShift
                 <sup>l</sup>↓ codeErreur, pShiftNouv
   - if (codeErreur = PAS_D_ERREUR)
                  ↓ pDebShifts, date, heure, pShiftNouv, pShiftSauv
    ajouterShift
                  ↓ pDebShifts, pShiftSauv
    shiftLu
             ↓ date, heure
fermer le fichier
```

```
-* chargementInscriptions
ouvrir Inscriptions.dat en mode lecture
codeErreur = PAS_D_ERREUR
pDebShifts = NIL
pShiftSauv = NIL
  shiftLu
          ↓ date, heure
  = do while (NOT fin de fichier and codeErreur = PAS_D_ERREUR)
   nouveauShift
                 ↓ codeErreur, pNouvShift
    if (codeErreur = PAS_D_ERREUR)
                  ↓ pDebShifts, date, heure, pShiftNouv, pShiftSauv
    ajouterShift
                  ↓ pDebShifts, pShiftSauv
// gestion des doublettes à ajouter
    shiftLu
             ↓ date, heure
fermer le fichier
```

```
-* sauverShifts
ouvrir Inscriptions.dat en écriture
 — if (erreur ouverture)
 codeErreur = ERREUR_OUVERTURE_FICHIER
  - else
 codeErreur = PAS_D_ERREUR
 pShift = pDebShifts
  = do while (pShift ≠ NIL)
  shift.date = pShift → date
  shift.heure = pShift → heure
  shift.nbDoublettes = pShift → nbDoublettes
                 ↓ shift
    ecrireShift
  pDoublette = pShift → pDebDoublettes
   = do while (pDoublette ≠ NIL)
   inscription.matricule1 = pDoublette → matricule1
   inscription.matricule2 = pDoublette → matricule2
   inscription.catégorie = pDoublette → catégorie
                        <sub>1</sub>↓ inscription
     ecrireInscription
   pDoublette = pDoublette→pSuiv
  pShift = pShift→pSuiv
 fermer Inscriptions.dat
```