

Compte rendu du TP1 de Structure de Données

LAURENT Valentin, MALRIN Vincent

30 mars 2013

Table des matières

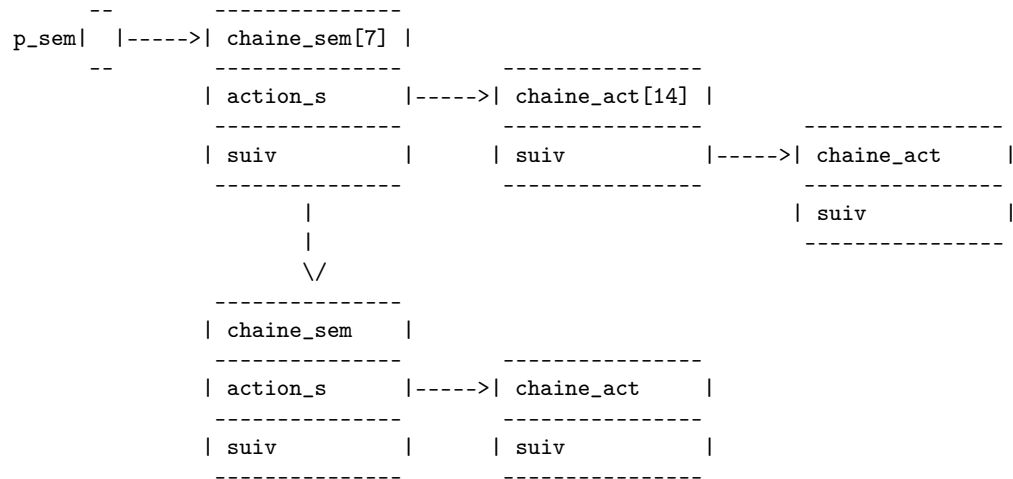
1.1	Objet du TP	2
1.2	Structure de données	2
1.3	Organisation du code source	3
1.4	Codes Sources	4
1.4.1	EN TÊTE : agenda.h	4
1.4.2	MAIN : agenda.c	9
1.4.3	INSERTION : insertion.c	12
1.4.4	AFFICHAGE : afficher.c	14
1.4.5	RECHERCHE : recherche.c	16
1.4.6	SUPPRESSION : suppression.c	18
1.4.7	RECHERCHE DE MOTIF : motif.c	19
1.4.8	ECRITURE : ecriture.c	21
1.4.9	BILATÈRE : bilatere.c	23
1.4.10	MAKEFILE	26
1.5	Jeux de tests	27
1.5.1	Cas à tester	27
1.5.2	Fichiers en entrée	28
1.5.3	Execution	29

1.1 Objet du TP

Création d'un agenda grâce à une liste chaînée à deux niveaux.

1.2 Structure de données

Chaque bloc du premier niveau contient une année, le numéro de la semaine, un pointeur vers la liste des actions de la semaine et un pointeur vers la semaine suivante. Les blocs du second niveau (liste chaînée des actions) contiennent le jour de la semaine, l'heure, le nom de l'action et le pointeur vers l'action suivante.



1.3 Organisation du code source

agenda.h

C'est le fichier d'entête.

insertion.c

Ce fichier contient les fonctions :

- InsererSem
- InsererAct
- Liberation

recherche.c

Ce fichier contient les fonctions :

- RechercheSem
- RechercheAct

suppression.c

Ce fichier contient les fonctions :

- SuppressionAct
- SuppressionSem

afficher.c

Ce fichier contient les fonctions :

- Affichage
- Menu

bilatere.c

Ce fichier contient les fonctions :

- TransfoBilatere
- IsertionBil
- LiberationBil

ecriture.c

Ce fichier contient les fonctions :

- Ecriture
- Lecture

recherche_motif.c

Ce fichier contient les fonctions :

- RechercheMotif
- AffichageMotif

1.4 Codes Sources

1.4.1 EN TÊTE : agenda.h

```
1  /* Malrin Vincent
2   * Laurent Valentin
3   * 25/03/2013
4   * EN TETE : agenda.h
5   */
6
7  /*GARDIEN*/
8  #ifndef AGENDA_H
9  #define AGENDA_H
10
11 /*DEFINE*/
12 #define TLIBELLE 10
13 #define TACT (TLIBELLE + 3)
14 #define TMAX 100
15
16 /* INCLUDE */
17 #include <stdio.h>
18 #include <stdlib.h>
19 #include <string.h>
20
21 /*STRUCTURES*/
22 typedef struct struct2
23 {
24     char          chaine_act[3 + TLIBELLE + 1];
25     struct struct2 *  suiv;
26 } action_t;
27
28 typedef struct struct1
29 {
30     char          chaine_sem[7];
31     action_t *    action_s;
32     struct struct1 *  suiv;
33 } semaine_t;
34
35 typedef struct struct3
36 {
37     char          motif[13];
38 } motif_t;
39
40 typedef struct struct4
41 {
42     char          chaine_sem[7];
43     action_t *    action_s;
44     struct struct4 *  prec;
45     struct struct4 *  suiv;
46 } bilatere_t;
47
48 /*-----*/
49 /*          INSERTION (insertion.c)          */
50 /*-----*/
51
52
53 /* FONCTION : InserirSem
54  * Insertion d'une nouvelle semaine dans la liste chaine.
55  *
```

```

56  * entree : le pointeur de recherche (semaine_t ** prec)
57  *      les informations à inserer (char * chaine_sem)
58  * varloc : le nouveau pointeur à inserer (semaine_t * new)
59  * sortie : le nouveau pointeur (semaine_t * new)
60  */
61
62 semaine_t * InserirSem(semaine_t ** prec, char * chaine_sem);
63
64 /* FONCTION : InserirAct
65  * Insertion d'une nouvelle action dans la liste chainee.
66  *
67  * entree : le pointeur de recherche (action_t ** prec)
68  *      les informations à inserer (char * chaine_act)
69  * varloc : le nouveau pointeur à inserer (action_t * new)
70  * sortie : le nouveau pointeur (action_t * new)
71  */
72
73 action_t * InserirAct(action_t ** prec, char * chaine_act);
74
75 /* FONCTION : Liberation
76  * Libere toute les données allouees.
77  *
78  * entree : le pointeur de debut de la liste chainee (semaine_t * p_sem)
79  * varloc : les pointeurs courants (action_t * cour_act et semaine_t * cour_sem)
80  *      les pointeurs temporaires (action_t * tmp_act et semaine_t * tmp_sem)
81  * sortie : rien
82  */
83
84 void Liberation(semaine_t * p_sem);
85
86 /*-----*/
87 /*      RECHERCHE (recherche.c)      */
88 /*-----*/
89
90 /* FONCTION : RechercheSem
91  * Recherche de la semaine dans la liste chainee.
92  *
93  * entree : le pointeur de debut de la liste chainee (semaine_t ** p_sem)
94  *      les informations de à rechercher (char * chaine_sem)
95  * varloc : le pointeur precedent (semaine_t ** prec)
96  *      le pointeur courant (semaine_t * cour)
97  * sortie : le pointeur correspondant à la recherche (semaine_t ** prec)
98  */
99
100 semaine_t ** RechercheSem(semaine_t ** p_sem, char * chaine_sem);
101
102 /* FONCTION : RechercheAct
103  * Recherche de l'action dans la liste chainee.
104  *
105  * entree : le pointeur de debut de la liste chainee (action_t ** p_act)
106  *      les informations de à rechercher (char * chaine_act)
107  * varloc : le pointeur precedent (action_t ** prec)
108  *      le pointeur courant (action_t * cour)
109  * sortie : le pointeur correspondant à la recherche (action_t ** prec)
110  */
111
112 action_t ** RechercheAct(action_t ** p_act, char * chaine_act);
113
114 /*-----*/

```

```

115  /*          SUPPRESSION (suppression.c)          */
116  /*-----*/
117
118  /* FONCTION : SuppressionAct
119   * Suppression de l'action dans la liste chainée.
120   *
121   * entree : le pointeur de debut de la liste chainee (action_t ** p_act)
122   *          les informations à rechercher (char * jour)
123   * varloc : le pointeur precedent (action_t ** prec)
124   *          le pointeur courant (action_t * cour)
125   */
126
127  void SuppressionAct(action_t ** prec);
128
129  /* FONCTION : SuppressionSem
130   * Suppression de la semaine dans la liste chainee.
131   *
132   * entree : le pointeur de debut de la liste chaiee (semaine_t ** p_sem)
133   *          les informations à rechercher (char * jour)
134   * varloc : le pointeur precedent (semaine_t ** prec)
135   *          le pointeur courant (semaine_t * cour)
136   */
137
138  void SuppressionSem(semaine_t ** p_sem);
139
140  /*-----*/
141  /*          AFFICHER (afficher.c)          */
142  /*-----*/
143
144  /* FONCTION : Affichage
145   * Affiche les informations des listes chainees (semaine_t).
146   *
147   * entree : le pointeur de debut de la liste chainee (semaine_t * p_sem)
148   * varloc : le pointeur courant (action_t * cour)
149   * sortie : rien
150   */
151
152  void Affichage(semaine_t * p_sem);
153
154  /* FONCTION : Menu
155   * Affichage du menu.
156   * entree : code de saisie (int saisie)
157   *          le ponteur vers le choix (int * choix)
158   * sortie : rien
159   */
160
161  void Menu(int saisie, int * choix);
162
163  /*-----*/
164  /*          BILATERE (bilatere.c)          */
165  /*-----*/
166
167  /* FONCTION : Transformation
168   * Transforme la liste des semaines en liste bilatere.
169   *
170   * entree : le pointeur de debut de la liste chainee (semaine_t ** p_sem)
171   *          le pointeur de debut de la liste bilatere (semaine_t * p_bil)
172   * varloc : le pointeurs precedents (semaine_t ** prec, bilatere_t ** prec_bil)
173   *          le pointeur courant de la liste bilatere (bilatere_t * nouv)

```

```

174  * sortie : le pointeur de debut de la liste bilatere (semaine_t * p_bil)
175  */
176
177 bilatere_t * TransfoBilatere (semaine_t ** p_sem, bilatere_t * p_bil);
178
179 /* FONCTION : InsertionBil
180  * Insère une cellule.
181  *
182  * entree : l'adresse du pointeur où il faut inserer (bilatere_t ** b)
183  *          le pointeur de la cellule à inserer (bilatere_t * elt)
184  * sortie : l'adresse de l'élément inséré (semaine_t ** adr_bil)
185  */
186
187 bilatere_t ** InsertionBil(bilatere_t ** b, bilatere_t * elt);
188
189 /* FONCTION : LiberationBil
190  * Libere toute les données allouées dans la structure bilatere.
191  *
192  * entree : le pointeur de debut de la liste bilatere (bilatere_t * p_bil)
193  * varloc : les pointeurs courants (action_t * cour_act et bilatere_t * cour_bil)
194  *          les pointeurs temporaires (action_t * tmp_act et bilatere_t * tmp_bil)
195  * sortie : rien
196  */
197
198 void LiberationBil(bilatere_t * p_bil);
199
200 /*-----*/
201 /*          ECRITURE (ecriture.c)          */
202 /*-----*/
203
204 /* FONCTION : Ecriture
205  * Ecriture de la liste dans un fichier texte.
206  *
207  * entree : le pointeur de debut de liste (semaine_t * p_sem)
208  *          le nom du fichier à ouvrir (char * nom)
209  * varloc : le nouveau pointeur à inserer (semaine_t * new)
210  * sortie : code erreur fourni en parametre de sortie (int code)
211  */
212
213 int Ecriture(semaine_t * p_sem, char * nom);
214
215 /* FONCTION : Lecture
216  * Lecture dans un fichier texte et creation de la liste.
217  *
218  * entree : le pointeur de la structure (semaine_t ** p_sem)
219  *          le nom du fichier à ouvrir (char * nom_f)
220  * varloc : pointeurs de parcours, les informations à copier
221  * sortie : code erreur fourni en parametre de sortie (int erreur)
222  */
223
224 int Lecture(semaine_t ** p_sem, char * nom_f);
225
226 /*-----*/
227 /* RECHERCHE DE MOTIF (recherche_motif.c)    */
228 /*-----*/
229
230 /* FONCTION : RechercheMotif
231  * Crée une liste contigue des jours ou une action contenant un motif donné est présente.
232  *

```



```

233  * entree : pointeur sur la liste chainee des semaines (semaine_t * p_sem).
234  *      pointeur sur la liste contigue des jours (motif_t * p_motif).
235  *      chaine du motif à rechercher (char * motif_rech).
236  * varloc : les pointeurs courants (semaine_t * cour_sem, action_t * cour_act).
237  *      compteur de la liste contigue (int compteur).
238  *      chaine de copie (char chaine_motif[10], char libelle[11], char dechet[TMAX]).
239  * sortie : l'adresse de la dernière cellule de recherche.
240  */
241
242  int RechercheMotif(semaine_t * p_sem, motif_t * p_motif, char * motif_rech);
243
244  /* FONCTION : AffichageMotif
245  * Crée une liste contigue
246  *
247  * entree : pointeur sur la liste chainee des semaines (semaine_t * p_sem).
248  *      pointeur sur la liste contigue des jours (motif_t * p_motif).
249  *      le nombre de cellules completes (int compteur).
250  * varloc : les pointeurs courants (semaine_t * cour_sem, action_t * cour_act).
251  *      compteur de la liste contigue (int compteur).
252  *      chaine de copie (char chaine_motif[10], char libelle[11], char dechet[TMAX]).
253  * sortie : l'adresse de la dernière cellule de recherche.
254  */
255
256  void AffichageMotif(motif_t * p_motif, int compteur);
257
258  #endif

```

1.4.2 MAIN : agenda.c

```
1  /* Malrin Vincent
2   * Laurent Valentin
3   * 25/03/2013
4   * MAIN : agenda.c
5   * Creer une structure de données en mémoire à partir d'un fichier texte.
6   *
7   * EXEMPLE : 201321110TP de SDD
8   *          chaine_sem[6]chaine_act[3 + 10]
9   *
10  * entree : choix du menu (int)
11  *          nom du fichier texte (char *)
12  *          arguments : date (char *) ou nom de fichier d'écriture (char *)
13  *
14  * USAGE : agenda <choix menu> <fichier de lecture> <argument>
15  */
16
17 /*INCLUDE*/
18 #include <stdio.h>
19 #include <stdlib.h>
20 #include <string.h>
21 #include "agenda.h"
22
23 int main (int argc, char ** argv)
24 {
25     /*variables locales*/
26     semaine_t *      p_sem = NULL;
27
28     semaine_t **      rech_sem = NULL;
29     action_t **      rech_act = NULL;
30
31     bilatere_t *      p_bil = NULL;
32
33     motif_t *         p_motif = NULL; /*liste de motifs*/
34     int               compteur_motif = 0;
35
36     char              chaine_sem[7]; /*copies des informations*/
37     char              chaine_act[4];
38     char              dechet[100];
39     char              libelle[11];
40
41     int               choix = 1; /*choix par défaut*/
42     int               erreur = 0;
43
44     if (argc < 3) /*vérification arguments*/
45     {
46         printf("USAGE : %s <choix menu> <fichier de lecture> <argument>\n", argv[0]);
47         /*affichage du menu et saisie du choix*/
48         Menu(0,&choix);
49     }
50     else
51     {
52         /*initialisation du choix*/
53         choix = atoi(argv[1]);
54
55         /*lecture du fichier*/
56         erreur = Lecture(&p_sem,argv[2]);
57     }
```

```

58      /*initialisation des chaines*/
59      chaine_sem[6] = '\0';
60      chaine_act[3] = '\0';
61
62      while (choix != 0)
63      {
64
65          switch(choix)
66          {
67              case 0 :
68                  break;
69
70              case 1 :
71                  /*affichage*/
72                  Affichage(p_sem);
73                  break;
74
75              case 2 :
76                  /*recherche par date*/
77                  if ((argc >= 4) && (strlen(argv[3]) == 9)) /*vérification arguments*/
78                  {
79                      /*copie de la date à rechercher*/
80                      sscanf(argv[3], "%6c%3c%*c", chaine_sem, chaine_act);
81
82                      /*recherche dans l'agenda*/
83                      rech_sem = RechercheSem(&p_sem, chaine_sem);
84
85                      if (*rech_sem != NULL)
86                      {
87                          rech_act = RechercheAct(&(*rech_sem).action_s, chaine_act);
88                      }
89
90                      if ((*rech_sem != NULL) && (*rech_act != NULL))
91                      {
92                          sscanf((*rech_act).chaine_act, "%3c%10c", dechet, libelle);
93                          libelle[10] = '\0';
94                          printf("LIBELLE RECHERCHEE : %s\n", libelle);
95                      }
96                      else if ((*rech_sem == NULL) || (*rech_act == NULL)) /*affichage de l'
97                          erreur*/
98                      {
99                          printf("Aucune page n'est trouvee\n");
100                      }
101                      else
102                      {
103                          printf("Erreur : date incomplete\n");
104                      }
105                      break;
106
107              case 3 :
108                  /*suppression par date*/
109                  if ((argc >= 4) && (strlen(argv[3]) == 9)) /*vérification arguments*/
110                  {
111                      /*copie de la date à supprimer*/
112                      sscanf(argv[3], "%6c%3c%*c", chaine_sem, chaine_act);
113
114                      /*recherche dans l'agenda*/
115                      rech_sem = RechercheSem(&p_sem, chaine_sem);

```

```

116
117         if (*rech_sem != NULL)
118         {
119             rech_act = RechercheAct(&(**rech_sem).action_s,chaîne_act);
120         }
121
122         if ((*rech_sem != NULL) && (*rech_act != NULL))
123         {
124             sscanf(**rech_act).chaîne_act,"%3c%10c",dechet,libelle);
125             libelle[10] = '\0';
126             printf("LIBELLE SUPPRIMEE : %s\n\n",libelle);
127
128             SuppressionAct(rech_act);
129
130             if (**rech_sem).action_s == NULL)
131             {
132                 SuppressionSem(rech_sem);
133             }
134
135             Affichage(p_sem);
136         }
137         else if ((*rech_sem == NULL) || (*rech_act == NULL)) /*affichage de l'
138             erreur*/
139         {
140             printf("Aucune page n'est supprimée\n");
141         }
142     else
143     {
144         printf("Erreur : date incomplète\n");
145     }
146     break;
147
148 case 4 :
149     /*écriture*/
150     if (argc >= 4) /*vérification arguments*/
151     {
152         Ecriture(p_sem,argv[3]);
153     }
154     break;
155
156 case 5 :
157     /*transformation en liste bilatère*/
158     p_bil = (bilatere_t *) malloc(sizeof(bilatere_t));
159     strcpy((*p_bil).chaîne_sem,"\0");
160     (*p_bil).action_s = NULL;
161     (*p_bil).suiv = p_bil;
162     (*p_bil).prec = p_bil;
163
164     if (p_bil) /*vérification de l'allocation*/
165     {
166         TransfoBilatere(&p_sem,p_bil);
167         LiberationBil(p_bil);
168         free(p_bil);
169     }
170     break;
171
172 case 6 :
173     /*recherche de motif*/

```

```

174         p_motif = (motif_t *) malloc(sizeof(motif_t)*TMAX);
175
176         if ((argc >= 4) && (strlen(argv[3]) < 11) && p_motif) /*vérification arguments
177             */
178         {
179             /*copie du motif à rechercher*/
180             sscanf(argv[3], "%s%c", libelle);
181
182             compteur_motif = RechercheMotif(p_sem, p_motif, libelle);
183             AffichageMotif(p_motif, compteur_motif);
184             free(p_motif);
185         }
186         else
187         {
188             printf("Erreur : motif incorrect\n");
189         }
190         break;
191
192     default :
193         printf("Choix non identifié !\n");
194         break;
195     }
196
197     /*reinitialisation des pointeurs*/
198     rech_sem = NULL;
199     rech_act = NULL;
200
201     /*reinitialisation du choix (sinon boucle infinie)*/
202     /*Menu(1, &choix);*/
203     choix = 0;
204 }
205
206 /*libération des ressources*/
207 Liberation(p_sem);
208
209 printf("Fin du programme\n");
210 }
211
212 return 0;
213 }

```

1.4.3 INSERTION : insertion.c

```

1  /* Malrin Vincent
2   * Laurent Valentin
3   * 25/03/2013
4   * INSERTION : insertion.c
5   */
6
7  /*INCLUDE*/
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11 #include "agenda.h"
12
13 /* FONCTION : InsérerSem
14  * Insertion d'une nouvelle semaine dans la liste chaînée.
15  *
16  * entree : le pointeur de recherche (semaine_t ** prec)

```

```

17  *           les informations à inserer (char * chaine_sem)
18  * varloc : le nouveau pointeur à inserer (semaine_t * new)
19  * sortie : le nouveau pointeur (semaine_t * new)
20  */
21
22 semaine_t * InserirSem(semaine_t ** prec, char * chaine_sem)
23 {
24     /*variables locales*/
25     semaine_t *      new = (semaine_t *) malloc (sizeof(semaine_t));
26
27     if(new)
28     {
29         /*copie des informations*/
30         strcpy((*new).chaine_sem, chaine_sem);
31         (*new).action_s = NULL;
32         (*new).suiv = NULL;
33
34         /*insertion du pointeur new*/
35         (*new).suiv = *prec;
36         *prec = new;
37     }
38     return new;
39 }
40
41 /* FONCTION : InserirAct
42  * Insertion d'une nouvelle action dans la liste chainee.
43  *
44  * entree : le pointeur de recherche (action_t ** prec)
45  *           les informations à inserer (char * chaine_act)
46  * varloc : le nouveau pointeur à inserer (action_t * new)
47  * sortie : le nouveau pointeur (action_t * new)
48  */
49
50 action_t * InserirAct(action_t ** prec, char * chaine_act)
51 {
52     /*variables locales*/
53     action_t *      new = (action_t *) malloc (sizeof(action_t));
54
55     if(new)
56     {
57         /*copie des informations*/
58         strcpy((*new).chaine_act, chaine_act);
59
60         /*insertion du pointeur new*/
61         (*new).suiv = *prec;
62         *prec = new;
63     }
64     return new;
65 }
66
67 /* FONCTION : Liberation
68  * Libere toute les données allouees.
69  *
70  * entree : le pointeur de debut de la liste chainee (semaine_t * p_sem)
71  * varloc : les pointeurs courants (action_t * cour_act et semaine_t * cour_sem)
72  *           les pointeurs temporaires (action_t * tmp_act et semaine_t * tmp_sem)
73  * sortie : rien
74  */
75

```

```

76 void Liberation(semaine_t * p_sem)
77 {
78     /*variables locales*/
79     semaine_t *    cour_sem = p_sem;
80     action_t *     cour_act;
81     action_t *     tmp_act;
82     semaine_t *    tmp_sem;
83
84     /*parcours*/
85     while (cour_sem)
86     {
87
88         /*on initialise le pointeur d'action*/
89         cour_act = (*cour_sem).action_s;
90
91         while (cour_act)
92         {
93             tmp_act = cour_act;
94
95             /*on avance les pointeurs de action_t*/
96             cour_act = (*cour_act).suiv;
97
98             /*libération de la structure action_t*/
99             free(tmp_act);
100         }
101
102         tmp_sem = cour_sem;
103
104         /*on avance le pointeur de semaine_t*/
105         cour_sem = (*cour_sem).suiv;
106
107         /*libération de la structure semaine_t*/
108         free(tmp_sem);
109     }
110
111     p_sem = NULL;
112 }
113

```

1.4.4 AFFICHAGE : afficher.c

```

1  /* Malrin Vincent
2   * Laurent Valentin
3   * 25/03/2013
4   * AFFICHAGE : afficher.c
5   */
6
7  /*INCLUDE*/
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11 #include "agenda.h"
12
13 /* FONCTION : Affichage
14  * Affiche les informations des listes chaînées (semaine_t).
15  *
16  * entree : le pointeur de debut de la liste chaînée (semaine_t * p_sem)
17  * varloc : les pointeurs courants (action_t * cour_act et semaine_t * cour_sem)
18  *
19  * les informations à afficher (char)

```

```

19  * sortie : rien
20  */
21
22 void Affichage(semaine_t * p_sem)
23 {
24     /*variables locales*/
25     semaine_t *    cour_sem = p_sem;
26     action_t *     cour_act;
27     char           annee[5];
28     char           semaine[3];
29     char           jour[2];
30     char           heure[3];
31     char           libelle[11];
32
33     /*parcours*/
34     while (cour_sem)
35     {
36         /*initialisation des chaines*/
37         jour[1] = '\0';
38         annee[4] = '\0';
39         semaine[2] = '\0';
40         heure[2] = '\0';
41         libelle[10] = '\0';
42
43
44         /*on initialise le pointeur d'action*/
45         cour_act = (*cour_sem).action_s;
46
47         /*mise à jour des informations à afficher*/
48         /*chaîne_sem,semaine*/
49         sscanf(cour_sem->chaîne_sem,"%4c%2c",annee,semaine);
50
51         while (cour_act)
52         {
53             /*jour,heure,libelle*/
54             sscanf(cour_act->chaîne_act,"%1c%2c%10c",jour,heure,libelle);
55
56             /*affichage des informations*/
57             printf("ANNEE : %s\n",annee);
58             printf("SEMAINE : %s\n",semaine);
59             printf("JOUR : %s\n",jour);
60             printf("HEURE : %s\n",heure);
61             printf("LIBELLE : %s\n",libelle);
62             printf("\n");
63
64             /*on avance les pointeurs de action_t*/
65             cour_act = (*cour_act).suiv;
66         }
67
68         /*on avance le pointeur de semaine_t*/
69         cour_sem = (*cour_sem).suiv;
70     }
71
72     printf("-----\n");
73 }
74
75
76 /* FONCTION : Menu
77  * Affichage du menu.

```



```

78  * entree : code de saisie (int saisie)
79  *         le ponteur vers le choix (int * choix)
80  * sortie : rien
81  */
82
83 void Menu(int saisie, int * choix)
84 {
85     /*saisie controle activation du scanf*/
86     printf("\n<choix menu> :\n");
87     printf("0) Quitter\n");
88     printf("1) Affichage\n");
89     printf("2) Recherche par date\n");
90     printf("3) Suppression\n");
91     printf("4) Ecriture\n");
92     printf("5) Transformation en liste bilatere\n");
93     printf("6) Recherche de motifs\n");
94
95     printf("\n<argument> :\n<menu 1,2> -> date\n<menu 5> -> fichier d'ecriture\n<menu 6> -> motif\n");
96
97     if (saisie == 1)
98     {
99         printf("Veuillez choisir votre choix ? ");
100        fflush(stdout);
101        scanf("%d%c", choix);
102        printf("\n");
103    }
104 }

```

1.4.5 RECHERCHE : recherche.c

```

1  /* Malrin Vincent
2  * Laurent Valentin
3  * 25/03/2013
4  * RECHERCHE : recherche.c
5  */
6
7  /*INCLUDE*/
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11 #include "agenda.h"
12
13 /* FONCTION : RechercheSem
14  * Recherche de la semaine dans la liste chainee.
15  *
16  * entree : le pointeur de debut de la liste chainee (semaine_t ** p_sem)
17  *         les informations de à rechercher (char * chaine_sem)
18  * varloc : le pointeur precedent (semaine_t ** prec)
19  *         le pointeur courant (semaine_t * cour)
20  * sortie : le pointeur correspondant à la recherche (semaine_t ** prec)
21  */
22
23 semaine_t ** RechercheSem(semaine_t ** p_sem, char * chaine_sem)
24 {
25     /*variables locales*/
26     char annee[5];
27     char cour_annee[5];
28     char semaine[3];

```

```

29     char                cour_semaine[3];
30     semaine_t **        prec = p_sem;
31     semaine_t *          cour = *p_sem;
32
33     if (cour != NULL) /*si la liste des semaines est nulle*/
34     {
35         /*initialisation des chaines*/
36         sscanf(chaine_sem,"%4c%2c",annee,semaine);
37         sscanf(cour->chaine_sem,"%4c%2c",cour_annee,cour_semaine);
38         annee[4] = '\0';
39         cour_annee[4] = '\0';
40         semaine[2] = '\0';
41         cour_semaine[2] = '\0';
42     }
43
44     /*parcours*/
45     while ((cour != NULL) && ((strcmp(cour_annee,annee) < 0) || ((strcmp(cour_annee,annee) == 0)
46         && (strcmp(cour_semaine,semaine) < 0))))
47     {
48         /*on avance les pointeurs*/
49         prec = &(*cour).suiv;
50         cour = (*cour).suiv;
51
52         /*si le pointeur suivant est vide, la copie est impossible*/
53         if (cour != NULL)
54         {
55             sscanf(cour->chaine_sem,"%4c%2c",cour_annee,cour_semaine);
56         }
57         if ((cour != NULL) && ((strcmp(cour_annee,annee) > 0) || (strcmp(cour_semaine,semaine) > 0)))
58         {
59             *prec = NULL; /*la date est anterieure aux dates de listes*/
60         }
61
62         return prec;
63     }
64
65     /* FONCTION : RechercheAct
66     * Recherche de l'action dans la liste chainee.
67     *
68     * entree : le pointeur de debut de la liste chainee (action_t ** p_act)
69     *           les informations de à rechercher (char * chaine_act)
70     * varloc : le pointeur precedent (action_t ** prec)
71     *           le pointeur courant (action_t * cour)
72     * sortie : le pointeur correspondant à la recherche (action_t ** prec)
73     */
74
75     action_t ** RechercheAct(action_t ** p_act, char * chaine_act)
76     {
77         /*variables locales*/
78         char                jour[2];
79         char                cour_jour[2];
80         char                heure[3];
81         char                cour_heure[3];
82         action_t **        prec = p_act;
83         action_t *          cour = *p_act;
84
85         if (cour != NULL) /*si la liste des actions est nulle*/
86         {

```

```

87      /*initialisation des chaines*/
88      sscanf(chaine_act,"%1c%2c",jour,heure);
89      sscanf(cour->chaine_act,"%1c%2c",cour_jour,cour_heure);
90      jour[1] = '\0';
91      cour_jour[1] = '\0';
92      heure[2] = '\0';
93      cour_heure[2] = '\0';
94  }
95
96  /*parcours*/
97  while ((cour != NULL) && ((strcmp(cour_jour,jour) < 0) || ((strcmp(cour_jour,jour) == 0) && (
    strcmp(cour_heure,heure) < 0))))
98  {
99      /*on avance les pointeurs*/
100     prec = &(*cour).suiv;
101     cour = (*cour).suiv;
102
103     /*si le pointeur suivant est vide, la copie est impossible*/
104     if (cour != NULL)
105     {
106         sscanf(cour->chaine_act,"%1c%2c",cour_jour,cour_heure);
107     }
108 }
109
110 if ((cour != NULL) && ((strcmp(cour_jour,jour) > 0) || (strcmp(cour_heure,heure) > 0)))
111 {
112     *prec = NULL; /*la date est anterieure aux dates de listes*/
113 }
114 return prec;
115 }

```

1.4.6 SUPPRESSION : suppression.c

```

1  /* Malrin Vincent
2   * Laurent Valentin
3   * 25/03/2013
4   * SUPPRESSION : suppression.c
5   */
6
7  /*INCLUDE*/
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11 #include "agenda.h"
12
13
14 /* FONCTION : SuppressionAct
15  * Suppression de la semaine dans la liste chainée.
16  *
17  * entree : l'adresse de la structure à supprimer (action_t ** prec)
18  * varloc : le pointeur temporaire (action_t * tmp)
19  */
20
21 void SuppressionAct(action_t ** prec)
22 {
23     /*variables locales*/
24     action_t *      tmp = *prec;
25
26     /*le pointeur à supprimer est NULL*/

```

```

27     if (*prec == NULL)
28     {
29         printf("Erreur\n");
30     }
31     else
32     {
33         *prec = (**prec).suiv;
34         free(tmp);
35     }
36 }
37
38 /* FONCTION : SuppressionSem
39  * Suppression de la semaine dans la liste chaînée.
40  *
41  * entree : l'adresse de la structure à supprimer (semaine_t ** prec)
42  * varloc : le pointeur temporaire (semaine_t * tmp)
43  */
44
45 void SuppressionSem(semaine_t ** prec)
46 {
47     /*variables locales*/
48     semaine_t *      tmp = *prec;
49
50     /*le pointeur à supprimer est NULL*/
51     if (*prec == NULL)
52     {
53         printf("Erreur\n");
54     }
55     else
56     {
57         *prec = (**prec).suiv;
58         free(tmp);
59     }
60 }

```

1.4.7 RECHERCHE DE MOTIF : motif.c

```

1  /* Malrin Vincent
2   * Laurent Valentin
3   * 25/03/2013
4   * RECHERCHE DE MOTIF : motif.c
5   */
6
7  /*INCLUDE*/
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11 #include "agenda.h"
12
13 /* FONCTION : RechercheMotif
14  * Crée une liste contigue des jours ou une action contenant un motif donné est présente.
15  *
16  * entrée : pointeur sur la liste chaînée des semaines (semaine_t * p_sem).
17  *          pointeur sur la liste contigue des jours (motif_t * p_motif).
18  *          chaine du motif à rechercher (char * motif_rech).
19  * varloc : les pointeurs courants (semaine_t * cour_sem, action_t * cour_act).
20  *          compteur de la liste contigue (int compteur).
21  *          chaine de copie (char chaine_motif[10], char libelle[11], char dechet[TMAX]).
22  * sortie : l'adresse de la dernière cellule de recherche.

```

```

23  */
24
25  int RechercheMotif(semaine_t * p_sem, motif_t * p_motif, char * motif_rech)
26  {
27      /*variables locales*/
28      semaine_t *      cour_sem = p_sem;
29      action_t *      cour_act = (*cour_sem).action_s; /*on initialise le pointeur d'action*/
30      int              compteur = 0;
31
32      char              dechet[TMAX];
33      char              libelle[11];
34
35      /*compteur*/
36      while (cour_sem)
37      {
38          /*initialisation des chaines*/
39          libelle[11] = '\0';
40
41          /*on initialise le pointeur d'action*/
42          cour_act = (*cour_sem).action_s;
43
44          while (cour_act)
45          {
46              sscanf(cour_act->chaine_act, "%3c%10c", dechet, libelle);
47              if (strstr(libelle, motif_rech) != NULL)
48              {
49                  /*copie de la date du motif*/
50                  strcpy(p_motif[compteur].motif, cour_sem->chaine_sem);
51                  strncat(p_motif[compteur].motif, cour_act->chaine_act, 3);
52                  p_motif[compteur].motif[10] = '\0';
53                  compteur++;
54              }
55
56              /*on avance les pointeurs de action_t*/
57              cour_act = (*cour_act).suiv;
58          }
59
60          /*on avance le pointeur de semaine_t*/
61          cour_sem = (*cour_sem).suiv;
62      }
63      return compteur;
64  }
65
66  /* FONCTION : AffichageMotif
67   * Crée une liste contigue
68   *
69   * entrée : pointeur sur la liste chaînée des semaines (semaine_t * p_sem).
70   *          pointeur sur la liste contigue des jours (motif_t * p_motif).
71   *          le nombre de cellules complètes (int compteur).
72   * varloc : les pointeurs courants (semaine_t * cour_sem, action_t * cour_act).
73   *          compteur de la liste contigue (int compteur).
74   *          chaine de copie (char chaine_motif[10], char libelle[11], char dechet[TMAX]).
75   * sortie : l'adresse de la dernière cellule de recherche.
76   */
77
78  void AffichageMotif(motif_t * p_motif, int compteur)
79  {
80      /*variables locales*/
81      int              compteur_aff = 0;

```

```

82
83     if (compteur == 0) /*cas d'une liste vide*/
84     {
85         printf("Aucune date correspondante\n");
86     }
87     else
88     {
89         /*parcours*/
90         while (compteur_aff < compteur)
91         {
92             printf("%s\n",p_motif[compteur_aff].motif);
93             compteur_aff++;
94         }
95     }
96 }

```

1.4.8 ECRITURE : ecriture.c

```

1  /* Malrin Vincent
2   * Laurent Valentin
3   * 25/03/2013
4   * ECRITURE : ecriture.c
5   */
6
7  /*INCLUDE*/
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11 #include "agenda.h"
12
13 /* FONCTION : Ecriture
14  * Ecriture de la liste dans un fichier texte.
15  *
16  * entree : le pointeur de debut de liste (semaine_t * p_sem)
17  *          le nom du fichier à ouvrir (char * nom)
18  * varloc : le nouveau pointeur à inserer (semaine_t * new)
19  * sortie : code erreur fourni en parametre de sortie (int code)
20  */
21
22 int Ecriture(semaine_t * p_sem, char * nom)
23 {
24     /*variables locales*/
25     FILE *          f_ecriture = fopen(nom, "w");
26     semaine_t *     cour_sem = p_sem;
27     action_t *      cour_act;
28     char            chaine_sem[7];
29     char            chaine_act[3 + TLIBELLE + 1];
30     int             code = 0;
31
32     /*parcours*/
33     if (!f_ecriture)
34     {
35         printf("Ecriture impossible !");
36         code = 1;
37     }
38     else
39     {
40         while (cour_sem)
41         {

```

```

42      /*initialisation des chaines*/
43      chaine_sem[6] = '\0';
44      chaine_act[TACT] = '\0';
45
46      /*on initialise le pointeur d'action*/
47      cour_act = (*cour_sem).action_s;
48
49      /*mise à jour des informations à afficher*/
50      /*annee,semaine*/
51      strcpy(chaine_sem,cour_sem->chaine_sem);
52
53      while (cour_act)
54      {
55          /*jour,heure,libelle*/
56          strcpy(chaine_act,cour_act->chaine_act);
57
58          /*affichage des informations*/
59          fputs(chaine_sem,f_ecriture);
60          fputs(chaine_act,f_ecriture);
61          fputc('\n',f_ecriture);
62
63          /*on avance les pointeurs de action_t*/
64          cour_act = (*cour_act).suiv;
65      }
66
67      /*on avance le pointeur de semaine_t*/
68      cour_sem = (*cour_sem).suiv;
69  }
70
71      /*fermeture du fichier d'ecriture*/
72      fclose(f_ecriture);
73  }
74
75  return code;
76  }
77
78  /* FONCTION : Lecture
79   * Lecture dans un fichier texte et creation de la liste.
80   *
81   * entree : le pointeur de la structure (semaine_t ** p_sem)
82   *          le nom du fichier à ouvrir (char * nom_f)
83   * varloc : pointeurs de parcours, les informations à copier
84   * sortie : code erreur fourni en parametre de sortie (int erreur)
85   */
86
87  int Lecture(semaine_t ** p_sem, char * nom_f)
88  {
89      /*variable locales*/
90      FILE *      f_lecture = fopen(nom_f, "r");
91      int         taille = TMAX + 10;
92      char *      ligne = (char *) calloc(1,sizeof(char)*taille);
93
94      semaine_t * new_sem = NULL; /*pointeurs de parcours des listes*/
95      semaine_t ** prec_sem = NULL;
96      action_t *   new_act = NULL;
97      action_t **  prec_act = NULL;
98
99      int          erreur = 0;
100

```

```

101 |     char          chaine_sem[7]; /*copies des informations*/
102 |     char          chaine_act[TACT + 1];
103 |     char          dechet[100];
104 |
105 |     if (f_lecture && ligne)
106 |     {
107 |         /*initialisation*/
108 |         prec_sem = p_sem;
109 |
110 |         /*creation de la structure*/
111 |         while ((fgets(ligne,taille,f_lecture) != NULL))
112 |         {
113 |             sscanf(ligne,"%6c%13c%100c",chaine_sem,chaine_act,dechet);
114 |
115 |             /*initialisation des chaines*/
116 |             chaine_sem[6] = '\0';
117 |             chaine_act[strlen(chaine_act)] = '\0';
118 |
119 |             if ((*p_sem == NULL) || (strcmp(new_sem->chaine_sem,chaine_sem) != 0))
120 |             {
121 |                 /*insertion de semaine_t*/
122 |                 new_sem = InsérerSem(prec_sem,chaine_sem);
123 |                 prec_sem = &((*new_sem).suiv);
124 |                 prec_act = &((*new_sem).action_s);
125 |
126 |                 /*insertion de action_t*/
127 |                 new_act = InsérerAct(prec_act,chaine_act);
128 |                 prec_act = &((*new_act).suiv);
129 |             }
130 |             else
131 |             {
132 |                 /*insertion de action_t*/
133 |                 new_act = InsérerAct(prec_act,chaine_act);
134 |                 prec_act = &((*new_act).suiv);
135 |
136 |             }
137 |         }
138 |
139 |         fclose(f_lecture);
140 |         free(ligne);
141 |     }
142 |     else
143 |     {
144 |         printf("Ouverture du fichier de lecture impossible !\n");
145 |         erreur = 1;
146 |     }
147 |
148 |     return erreur;
149 | }

```

1.4.9 BILATÈRE : bilatere.c

```

1 | /* Malrin Vincent
2 |  * Laurent Valentin
3 |  * 25/03/2013
4 |  * BILATERE : bilatere.c
5 |  */
6 |
7 | /*INCLUDE*/

```



```

8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11 #include "agenda.h"
12
13 /* FONCTION : Transformation
14  * Transforme la liste des semaines en liste bilatère.
15  *
16  * entrée : le pointeur de debut de la liste chaînée (semaine_t * p_sem)
17  *          le pointeur de debut de la liste bilatère (semaine_t * p_bil)
18  * varloc : le pointeurs precedents (semaine_t ** prec, bilatere_t ** prec_bil)
19  *          le pointeur courant de la liste bilatère (bilatere_t * nouv)
20  * sortie : le pointeur de debut de la liste bilatère (semaine_t * p_bil)
21  */
22
23 bilatere_t * TransfoBilatere (semaine_t ** p_sem, bilatere_t * p_bil)
24 {
25     /*variables locales*/
26     semaine_t **    prec = p_sem;
27     bilatere_t **    prec_bil = &p_bil;
28     bilatere_t *     nouv = NULL;
29
30     /*boucle*/
31     while (*prec)
32     {
33         nouv = (bilatere_t *) calloc (1,sizeof(bilatere_t)); /*création de la cellule bilatère*/
34
35         strcpy((*nouv).chaine_sem,(**prec).chaine_sem); /*copie des données*/
36         (*nouv).action_s = (**prec).action_s;
37
38         prec_bil = InsertionBil(prec_bil,nouv); /*insertion*/
39
40         prec = &(**prec).suiv; /*avance les prec*/
41     }
42     return p_bil;
43 }
44
45 /* FONCTION : InsertionBil
46  * Insère une cellule .
47  *
48  * entrée : l'adresse du pointeur où il faut inserer (bilatere_t ** b)
49  *          le pointeur de la cellule à inserer (bilatere_t * elt)
50  * sortie : l'adresse de l'élément inséré (semaine_t ** adr_bil)
51  */
52
53 bilatere_t ** InsertionBil(bilatere_t ** b, bilatere_t * elt)
54 {
55     /*insertion*/
56     elt->suiv = (**b).suiv;
57     elt->prec = *b;
58     (*b)->suiv->prec = elt;
59     (*b)->suiv = elt;
60
61     return &(**b).suiv;
62 }
63
64 /* FONCTION : LiberationBil
65  * Libere toute les données allouées dans la structure bilatere.
66  *

```

```

67  * entree : le pointeur de debut de la liste bilatere (bilatere_t * p_bil)
68  * varloc : les pointeurs courants (action_t * cour_act et bilatere_t * cour_bil)
69  *          les pointeurs temporaires (action_t * tmp_act et bilatere_t * tmp_bil)
70  * sortie : rien
71  */
72
73 void LiberationBil(bilatere_t * p_bil)
74 {
75     /*variables locales*/
76     bilatere_t *    cour_bil = (*p_bil).suiv;
77     bilatere_t *    tmp_bil = NULL;
78     int             verif = 0;
79
80     /*parcours*/
81     while ((cour_bil != NULL) && ((*cour_bil).action_s != NULL))
82     {
83
84         /* ATTENTION : les pointeurs (action_t *) ne sont pas libérés !
85          * Ils seront libérés dans la fonction Liberation.
86          */
87
88         tmp_bil = cour_bil;
89
90         /*on avance le pointeur de semaine_t*/
91         cour_bil = (*cour_bil).suiv;
92
93         /*libération de la structure semaine_t*/
94         (*tmp_bil).action_s = NULL;
95         (*tmp_bil).prec = NULL;
96         (*tmp_bil).suiv = NULL;
97         free(tmp_bil);
98
99     }
100     (*p_bil).suiv = NULL;
101     (*p_bil).prec = NULL;
102 }

```

1.4.10 MAKEFILE

```
1  # compilateur
2  CC = gcc
3
4  # options de compilation
5  CFLAGS = -Wall -ansi -pedantic -Wextra -g -O2
6
7  # options de l'edition des liens
8  LDFLAGS = -g
9
10 # nom de l'executable a generer
11 EXEC = agenda
12
13 # liste des fichiers objets
14 OBJ = agenda.o afficher.o bilatere.o insertion.o recherche.o suppression.o ecriture.o motif.o
15
16 all : $(EXEC)
17
18 # regle de production finale
19 agenda : $(OBJ)
20     $(CC) $(OBJ) -o $@ $(LDFLAGS)
21
22 # regle de production pour chaque fichier
23 agenda.o : agenda.c agenda.h
24     $(CC) -c $< $(CFLAGS)
25
26 afficher.o : afficher.c agenda.h
27     $(CC) -c $< $(CFLAGS)
28
29 bilatere.o : bilatere.c agenda.h
30     $(CC) -c $< $(CFLAGS)
31
32 ecriture.o : ecriture.c agenda.h
33     $(CC) -c $< $(CFLAGS)
34
35 insertion.o : insertion.c agenda.h
36     $(CC) -c $< $(CFLAGS)
37
38 recherche.o : recherche.c agenda.h
39     $(CC) -c $< $(CFLAGS)
40
41 suppression.o : suppression.c agenda.h
42     $(CC) -c $< $(CFLAGS)
43
44 motif.o : motif.c agenda.h
45     $(CC) -c $< $(CFLAGS)
46
47 # regle de suppression des fichiers
48 clean :
49     rm $(OBJ)
```

1.5 Jeux de tests

1.5.1 Cas à tester

Lecture et Insertion (Question 1)

Cas à tester :

- Cas de fichier non valide (pour la lecture)
- Cas liste vide
- Cas cellule unique
- Cas général (10 cellules)

Recherche et Suppression (Question 3)

Cas à tester :

- Cas liste vide
- Cas cellule unique
- Cas général (10 cellules)
 - Recherche en début
 - Recherche au milieu
 - Recherche en fin
- Tests sur des dates non valides

Recherche de Motif (Question 2)

Cas à tester :

- Cas liste vide
- Cas général (10 cellules)
 - Recherche en début
 - Recherche au milieu
 - Recherche en fin
- Test sur un motif non valide
- Test sur un motif multiple

Ecriture (Question 2)

Cas à tester :

- Cas liste vide
- Cas général (10 cellules)

Liste Bilatère (Question 4)

Cas à tester :

- Cas liste vide
- Cas cellule unique
- Cas général (3 cellules, pour facilité l’affichage sur DDD)

1.5.2 Fichiers en entrée

Test de listes vides : `donnee0.txt`

Test de cellule unique : `donnee1.txt`

```
1 || 201305401fin de la premiere question
```

Test du cas général (10 cellules) : `donnee10.txt`

```
1 || 201342108Calcul
2 || 201342110Automates
3 || 201342113Proba
4 || 201343208TP de SDD
5 || 201343210Systemes
6 || 201344308TP de BDD
7 || 201344313Anglais
8 || 201345408Circuits
9 || 201403515TP de physique
10 || 201403517Liberte
```

Test du cas général (4 cellules) : `donnee4.txt`

```
1 || 201305401fin de la premiere question
2 || 201305402joue a la balancoire
3 || 201321110TP de SDD
4 || 201371110TP de BDD
```

Fichier d'écriture : `donnee_e.txt`

1.5.3 Execution

Lecture et Insertion (Question 1)

- Cas Fichier invalide : /agenda 1 fichier_innexistent.txt

```
1 | Ouverture du fichier de lecture impossible !
2 | -----
3 | Fin du programme
```

- Cas liste vide : /agenda 1 donnee0.txt

```
1 | -----
2 | Fin du programme
```

- Cas liste unique : /agenda 1 donnee1.txt

```
1 | ANNEE : 2013
2 | SEMAINE : 05
3 | JOUR : 4
4 | HEURE : 01
5 | LIBELLE : fin de la
6 |
7 | -----
8 | Fin du programme
```

- Cas général : /agenda 1 donnee10.txt

```
1 | ANNEE : 2013
2 | SEMAINE : 42
3 | JOUR : 1
4 | HEURE : 08
5 | LIBELLE : Calcul
6 |
7 | ANNEE : 2013
8 | SEMAINE : 42
9 | JOUR : 1
10 | HEURE : 10
11 | LIBELLE : Automates
12 |
13 | ANNEE : 2013
14 | SEMAINE : 42
15 | JOUR : 1
16 | HEURE : 13
17 | LIBELLE : Proba
18 |
19 | ANNEE : 2013
20 | SEMAINE : 43
21 | JOUR : 2
22 | HEURE : 08
23 | LIBELLE : TP de SDD
24 |
25 | ANNEE : 2013
```

26 SEMAINE : 43
27 JOUR : 2
28 HEURE : 10
29 LIBELLE : Systemes
30
31 ANNEE : 2013
32 SEMAINE : 44
33 JOUR : 3
34 HEURE : 08
35 LIBELLE : TP de BDD
36
37 ANNEE : 2013
38 SEMAINE : 44
39 JOUR : 3
40 HEURE : 13
41 LIBELLE : Anglais
42
43 ANNEE : 2013
44 SEMAINE : 45
45 JOUR : 4
46 HEURE : 08
47 LIBELLE : Circuits
48
49 ANNEE : 2014
50 SEMAINE : 03
51 JOUR : 5
52 HEURE : 15
53 LIBELLE : TP de phys
54
55 ANNEE : 2014
56 SEMAINE : 03
57 JOUR : 5
58 HEURE : 17
59 LIBELLE : Liberte
60
61 -----
62 Fin du programme

Recherche (Question 3)

- Recherche -> Cas liste vide : /agenda 2 donnee0.txt 201342108

```
1 || Aucune page n'est trouvee
2 || Fin du programme
```

- Recherche -> Cas cellule unique : /agenda 2 donnee1.txt 201342108

```
1 || LIBELLE RECHERCHEE : fin de la
2 || Fin du programme
```

- Cas général (recherche au début) : /agenda 2 donnee10.txt 201342108 (libelle : Calcul)

```
1 || LIBELLE RECHERCHEE : Calcul
2 || Fin du programme
```

- Cas général (recherche au milieu) : /agenda 2 donnee10.txt 201343210 (libelle : Systeme)

```
1 || LIBELLE RECHERCHEE : Systemes
2 || Fin du programme
```

- Cas général (recherche au fin) : /agenda 2 donnee10.txt 201403517 (libelle : Liberte)

```
1 || LIBELLE RECHERCHEE : Liberte
2 || Fin du programme
```

- Cas général (date incorrecte) : /agenda 2 donnee10.txt 201343209

```
1 || Aucune page n'est trouvee
2 || Fin du programme
```


Suppression (Question 3)

- Suppression -> Cas liste vide : /agenda 3 donnee0.txt 201342108

```
1 | Aucune page n'est supprimee
2 | Fin du programme
```

- Suppression -> Cas cellule unique : /agenda 3 donnee1.txt 201342108

```
1 | LIBELLE SUPPRIMEE : fin de la
2 |
3 | -----
4 | Fin du programme
```

- Cas général (Suppression au début) : /agenda 3 donnee10.txt 201343108 (libelle : Calcul)

```
1 | LIBELLE SUPPRIMEE : Calcul
2 |
3 | ANNEE : 2013
4 | SEMAINE : 42
5 | JOUR : 1
6 | HEURE : 10
7 | LIBELLE : Automates
8 |
9 | ANNEE : 2013
10 | SEMAINE : 42
11 | JOUR : 1
12 | HEURE : 13
13 | LIBELLE : Proba
14 |
15 | ANNEE : 2013
16 | SEMAINE : 43
17 | JOUR : 2
18 | HEURE : 08
19 | LIBELLE : TP de SDD
20 |
21 | ANNEE : 2013
22 | SEMAINE : 43
23 | JOUR : 2
24 | HEURE : 10
25 | LIBELLE : Systemes
26 |
27 | ANNEE : 2013
28 | SEMAINE : 44
29 | JOUR : 3
30 | HEURE : 08
31 | LIBELLE : TP de BDD
32 |
33 | ANNEE : 2013
34 | SEMAINE : 44
35 | JOUR : 3
36 | HEURE : 13
37 | LIBELLE : Anglais
38 |
39 | ANNEE : 2013
```

```

40 SEMAINE : 45
41 JOUR : 4
42 HEURE : 08
43 LIBELLE : Circuits
44
45 ANNEE : 2014
46 SEMAINE : 03
47 JOUR : 5
48 HEURE : 15
49 LIBELLE : TP de phys
50
51 ANNEE : 2014
52 SEMAINE : 03
53 JOUR : 5
54 HEURE : 17
55 LIBELLE : Liberte
56
57 -----
58 Fin du programme

```

– Cas général (Suppression au milieu) : /agenda 3 donnee10.txt 201343208 (libelle : TP de SDD)

```

1 LIBELLE SUPPRIMEE : TP de SDD
2
3 ANNEE : 2013
4 SEMAINE : 42
5 JOUR : 1
6 HEURE : 08
7 LIBELLE : Calcul
8
9 ANNEE : 2013
10 SEMAINE : 42
11 JOUR : 1
12 HEURE : 10
13 LIBELLE : Automates
14
15 ANNEE : 2013
16 SEMAINE : 42
17 JOUR : 1
18 HEURE : 13
19 LIBELLE : Proba
20
21 ANNEE : 2013
22 SEMAINE : 43
23 JOUR : 2
24 HEURE : 10
25 LIBELLE : Systemes
26
27 ANNEE : 2013
28 SEMAINE : 44
29 JOUR : 3
30 HEURE : 08
31 LIBELLE : TP de BDD
32
33 ANNEE : 2013
34 SEMAINE : 44
35 JOUR : 3

```

```

36 | HEURE : 13
37 | LIBELLE : Anglais
38 |
39 | ANNEE : 2013
40 | SEMAINE : 45
41 | JOUR : 4
42 | HEURE : 08
43 | LIBELLE : Circuits
44 |
45 | ANNEE : 2014
46 | SEMAINE : 03
47 | JOUR : 5
48 | HEURE : 15
49 | LIBELLE : TP de phys
50 |
51 | ANNEE : 2014
52 | SEMAINE : 03
53 | JOUR : 5
54 | HEURE : 17
55 | LIBELLE : Liberte
56 |
57 | -----
58 | Fin du programme

```

– Cas général (Suppression au fin) : /agenda 3 donnee10.txt 201403517 (libelle : Liberte)

```

1 | LIBELLE SUPPRIMEE : Liberte
2 |
3 | ANNEE : 2013
4 | SEMAINE : 42
5 | JOUR : 1
6 | HEURE : 08
7 | LIBELLE : Calcul
8 |
9 | ANNEE : 2013
10 | SEMAINE : 42
11 | JOUR : 1
12 | HEURE : 10
13 | LIBELLE : Automates
14 |
15 | ANNEE : 2013
16 | SEMAINE : 42
17 | JOUR : 1
18 | HEURE : 13
19 | LIBELLE : Proba
20 |
21 | ANNEE : 2013
22 | SEMAINE : 43
23 | JOUR : 2
24 | HEURE : 08
25 | LIBELLE : TP de SDD
26 |
27 | ANNEE : 2013
28 | SEMAINE : 43
29 | JOUR : 2
30 | HEURE : 10
31 | LIBELLE : Systemes

```

```
32 |
33 | ANNEE : 2013
34 | SEMAINE : 44
35 | JOUR : 3
36 | HEURE : 08
37 | LIBELLE : TP de BDD
```

```
38 |
39 | ANNEE : 2013
40 | SEMAINE : 44
41 | JOUR : 3
42 | HEURE : 13
43 | LIBELLE : Anglais
```

```
44 |
45 | ANNEE : 2013
46 | SEMAINE : 45
47 | JOUR : 4
48 | HEURE : 08
49 | LIBELLE : Circuits
```

```
50 |
51 | ANNEE : 2014
52 | SEMAINE : 03
53 | JOUR : 5
54 | HEURE : 15
55 | LIBELLE : TP de phys
```

```
56 |
57 | -----
58 | Fin du programme
```

– Cas général (date incorrecte) : /agenda 3 donnee10.txt 201343209

```
1 | Aucune page n'est supprimee
2 | Fin du programme
```

Ecriture (Question 2)

- Cas liste vide : /agenda 4 donnee0.txt donnee_e0.txt

```
1 || Fin du programme
```

Affichage de fichier donnee_e0.txt

- Cas général : /agenda 4 donnee10.txt donnee_e1.txt

```
1 || Fin du programme
```

Affichage du fichier donnee_e1.txt :

```
1 | 201342108Calcul -  
2 | 201342110Automates -  
3 | 201342113Proba -  
4 | 201343208TP de SDD -  
5 | 201343210Systemes -  
6 | 201344308TP de BDD -  
7 | 201344313Anglais -  
8 | 201345408Circuits -  
9 | 201403515TP de phys-  
10 | 201403517Liberte -
```

Bilatere (Question 4)

- Cas liste vide : /agenda 5 donnee0.txt

1 || Fin du programme

Après l'exécution de la fonction TransfoBilatere on a sur DDD :

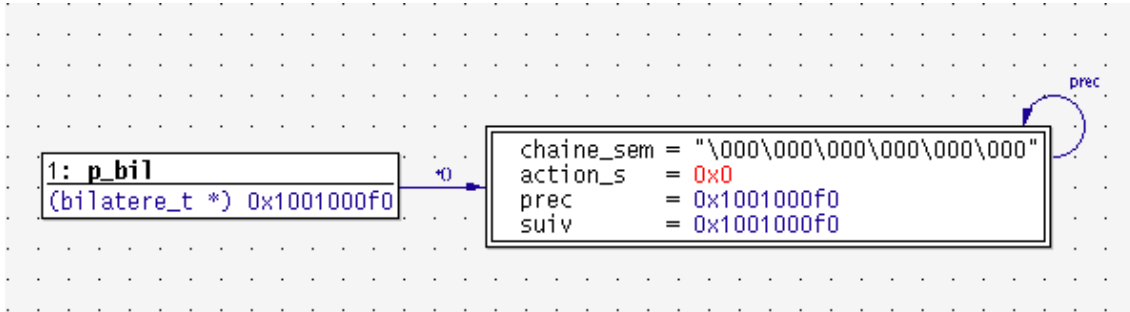


FIGURE 1.1 – Cas liste vide

- Cas cellule unique : /agenda 5 donnee1.txt

1 || Fin du programme

Après l'exécution de la fonction TransfoBilatere on a sur DDD :

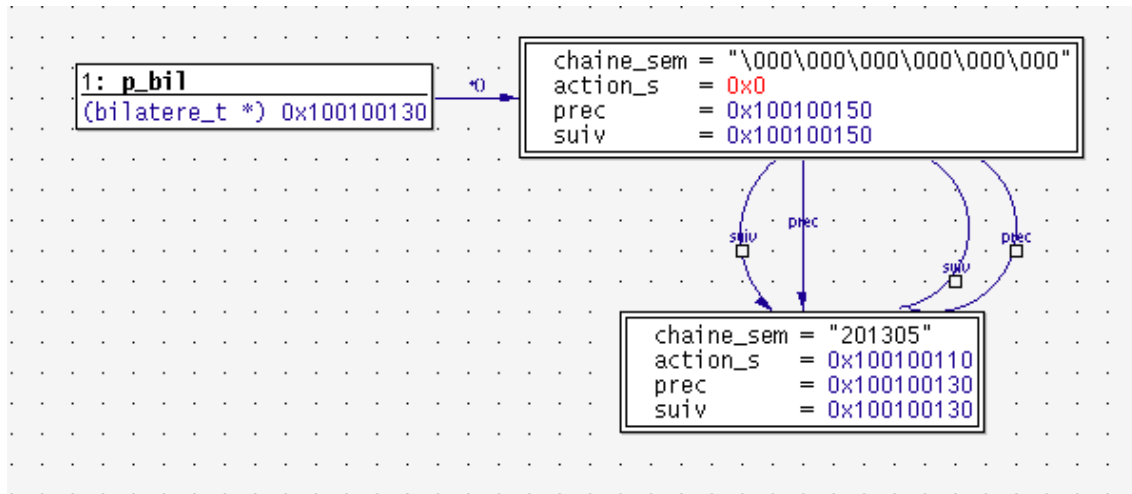


FIGURE 1.2 – Cas liste vide

Num	Expression	State	Scope	Address
1:	p_bil	enabled	TransfoBilatere	0x7fff5fbff748
2:	*p_bil	enabled	TransfoBilatere	0x100100130
3:	*p_bil->prec	enabled	TransfoBilatere	0x100100150
4:	*p_bil->suiv	alias of 3	TransfoBilatere	0x100100150
5:	*p_bil->prec->suiv	alias of 2	TransfoBilatere	0x100100130
6:	*p_bil->prec->prec	alias of 2	TransfoBilatere	0x100100130

– Cas général (4 cellules) : /agenda 5 donnee4.txt

1 || Fin du programme

Après l'exécution de la fonction TransfoBilaterale on a sur DDD :

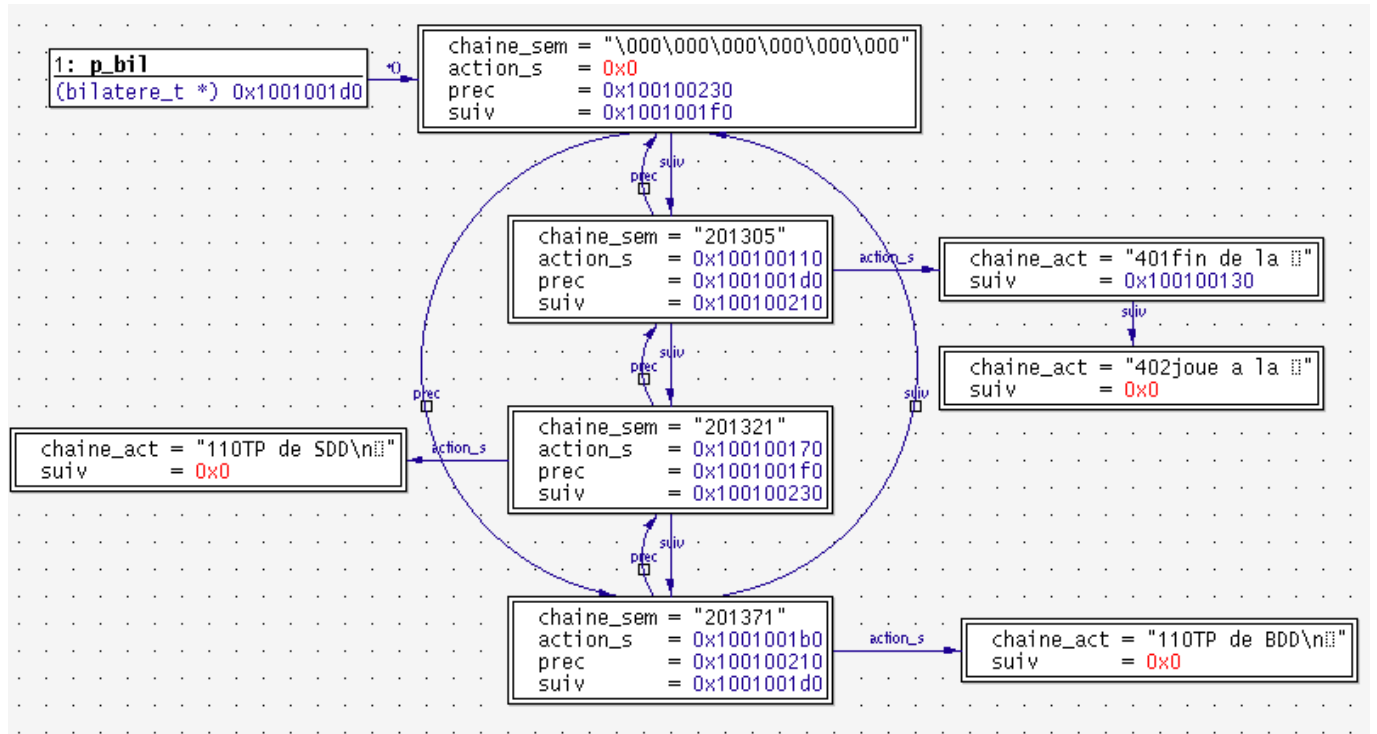


FIGURE 1.3 – Cas liste vide

Num	Expression	State	Scope	Address
1:	p_bil	enabled	TransfoBilaterere	0x7fff5fbff728
2:	*p_bil	enabled	TransfoBilaterere	0x1001001d0
8:	*p_bil->suiv->prec	alias of 2	TransfoBilaterere	0x1001001d0
9:	*p_bil->suiv	enabled	TransfoBilaterere	0x1001001f0
10:	*p_bil->suiv->suiv	enabled	TransfoBilaterere	0x100100210
11:	*p_bil->suiv->suiv->suiv	enabled	TransfoBilaterere	0x100100230
14:	*p_bil->suiv->prec	alias of 2	TransfoBilaterere	0x1001001d0
17:	*p_bil->suiv->suiv->prec	alias of 9	TransfoBilaterere	0x1001001f0
18:	*p_bil->suiv->suiv->suiv->prec	alias of 10	TransfoBilaterere	0x100100210
19:	*p_bil->suiv->suiv->suiv->suiv	alias of 2	TransfoBilaterere	0x1001001d0
20:	*p_bil->prec	alias of 11	TransfoBilaterere	0x100100230

Recherche Motif (Question 2)

- Cas liste vide : /agenda 6 donnee0.txt SDD

```
1 ||| Aucune date correspondante
2 ||| Fin du programme
```

- Cas général (recherche en début) : /agenda 6 donnee10.txt Calcul

```
1 ||| 201342108
2 ||| Fin du programme
```

- Cas général (recherche au milieu) : /agenda 6 donnee10.txt SDD

```
1 ||| 201343208
2 ||| Fin du programme
```

- Cas général (recherche en fin) : /agenda 6 donnee10.txt Liberte

```
1 ||| 201403517
2 ||| Fin du programme
```

- Cas général et libelle innexistant : /agenda 6 donnee10.txt truc

```
1 ||| Aucune date correspondante
2 ||| Fin du programme
```

- Cas général et libelle multiple : /agenda 6 donnee10.txt TP

```
1 ||| 201343208
2 ||| 201344308
3 ||| 201403515
4 ||| Fin du programme
```