

ISIMA PREMIÈRE ANNÉE

COMPTE-RENDU DE TP  
STRUCTURES DE DONNÉES

---

## Gestion de news à partir d'une liste chaînée

---

Benjamin BARBESANGE  
Pierre-Loup PISSAVY  
*Groupe G21*

*Enseignant :*  
Michelle CHABROL

février 2015



# 1 | Présentation

Le but de ce TP est de concevoir un ensemble de fonctions permettant de gérer des news sous forme de messages, chacun d'entre eux ayant une date de début et de fin de validité. On doit faire usage d'une liste chaînée. Les news sont ordonnées dans la liste chaînée selon l'ordre décroissant de la date de début (de la plus récente à la plus ancienne).

Les messages et informations satellites sont enregistrés dans un fichier, à raison d'une ligne par message. Ce fichier est supposé correct.

Les opérations suivantes sont permises :

- Charger une liste depuis un fichier,
- Sauvegarder une liste dans un fichier,
- Afficher les messages du jour,
- Supprimer les messages obsolètes,
- Modifier une date de début sur tous les messages,
- Afficher tous les messages contenant une chaîne particulière.

## 1.1 Structure de données employée

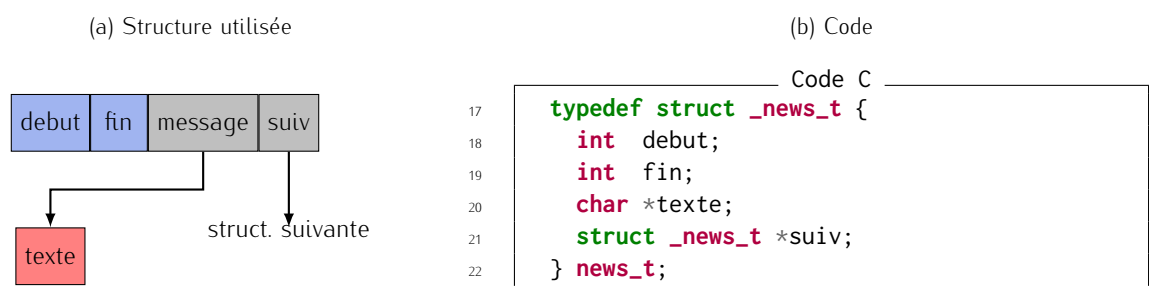


FIGURE 1.1 – Structure et code correspondant

## 1.2 Organisation du code source

Nous avons défini deux modules, le premier est dédié à la gestion des listes chaînées (adjonction, recherche, suppression etc.), et le second à la gestion des news (lecture, sauvegarde, modifications etc.). Enfin, le programme principal utilise conjointement ces deux modules pour réaliser le traitement voulu.

### 1.2.1 Gestion de liste chaînée

- `src/liste_news.h`
- `src/liste_news.c`

### 1.2.2 Gestion de news

- `src/gestion_news.h`
- `src/gestion_news.c`

### 1.2.3 Programme principal

- `src/main.c`

## 2 | Détails du programme

### 2.1 Gestion de liste chaînée

Code C

```
1  /* liste_news.h
2  Header
3
4  -----| GESTION DE NEWS PAR LISTE CHAINEE |-----
5
6  BARBESANGE Benjamin,
7  PISSAVY Pierre-Loup
8
9  ISIMA 1ere Annee, 2014-2015
10 */
11
12 #ifndef __LISTE_NEWS_H__
13 #define __LISTE_NEWS_H__
14
15 #include <string.h>
16
17 typedef struct _news_t {
18     int debut;
19     int fin;
20     char *texte;
21     struct _news_t *suiv;
22 } news_t;
23
24 typedef struct _news_t cell_t;
25
26 cell_t ** rech_prec(cell_t **, int, short int*);
27 void supp_cell(cell_t **);
28 void liberer_liste(cell_t **);
29 void ins_cell(cell_t **, cell_t *);
30 cell_t * creer_cell(int, int, char *);
31
32 #endif
```

```

1  /* liste_news.c
2  Fonctions de gestion de la liste chaine
3
4  -----| GESTION DE NEWS PAR LISTE CHAINEE |-----
5
6  BARBESANGE Benjamin,
7  PISSAVY Pierre-Loup
8
9  ISIMA 1ere Annee, 2014-2015
10 */
11
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include "liste_news.h"
15
16 /* void adj_cell(cell_t **prec, cell_t *elt)
17 Ajoute une cellule apres un element partir d'un pointeur sur l'element
18 et d'un pointeur sur le pointeur de l'element apres lequel ajouter
19
20 Entrees :
21     cell_t **prec : pointeur sur le pointeur de l'element apres lequel ajouter
22     cell_t *elt : pointeur sur l'element a ajouter a la liste chaine
23
24 Sortie :
25     Aucune
26 */
27 void adj_cell(cell_t **prec, cell_t *elt) {
28     elt->suiv = (*prec);
29     (*prec) = elt;
30 }
31
32 /* cell_t ** rech_prec(cell_t **liste, int debut, short int *existe)
33 Recherche le precedent d'un element dans la liste chaine a partir de la
34 date de debut de message
35
36 Entrees :
37     cell_t **liste : pointeur sur le pointeur du premier element de la liste chaine
38     int debut : date de debut de message que l'on souhaite inserer, sous la forme AAAAMMJJ
39                 on cherchera donc les messages dont la date est immediatement superieure
40     short int *existe : variable en entre/sortie indiquant si on a ou pas de message ayant
41     ↳ la date de debut
42         0 : il n'y a pas de message avec cette date de debut
43         1 : il y a au moins un message
44
45 Sortie :
46     cell_t ** : pointeur sur le pointeur de l'element precedent
47 */
48 cell_t ** rech_prec(cell_t **liste, int debut, short int *existe) {
49     cell_t **prec = liste;
50     while ((*prec) && (*prec)->debut > debut) {
51         prec = &((*prec)->suiv);
52     }
53     /* Booleen de presence */
54     /* 1 : present */
55     /* 0 : absent */

```

```

55     *existe = (*prec && (*prec)->debut == debut)?1:0;
56     return prec;
57 }
58
59 /* void supp_cell(cell_t **prec)
60    Permet de supprimer un element dans la liste chaine a partir
61    de son precedent
62
63    Entrees :
64        cell_t **prec : pointeur sur le pointeur de l'element precedent l'element a supprimer
65
66    Sortie :
67        Aucune
68 */
69 void supp_cell(cell_t **prec) {
70     cell_t *elt = *prec;
71     *prec = elt->suiv;
72     free(elt->texte);
73     free(elt);
74 }
75
76 /* void liberer_liste(cell_t **liste)
77    Libere les allocations memoires de la liste
78
79    Entrees :
80        cell_t **liste : pointeur sur le pointeur du premier element de la liste chaine
81
82    Sortie :
83        Aucune
84 */
85 void liberer_liste(cell_t **liste) {
86     while (*liste) {
87         supp_cell(liste);
88     }
89     *liste = NULL;
90 }
91
92 /* void ins_cell(cell_t **liste, cell_t *elt)
93    Permet d'insérer une cellule a la bonne place dans la liste chaine
94    Les messages sont tries par ordre decroissant des date de debut
95
96    Entrees :
97        cell_t **liste : pointeur sur le pointeur du premier element de la liste chaine
98        cell_t *elt : pointeur sur l'element a inserer dans la liste chaine
99
100    Sortie :
101        Aucune
102 */
103 void ins_cell(cell_t **liste, cell_t *elt) {
104     short int existe;
105     cell_t **prec = rech_prec(liste, elt->debut, &existe);
106     adj_cell(prec, elt);
107 }
108
109 /* news_t * creer_cell(int debut, int fin, char *message)
110    Permet de creer un element de la liste chaine a partir de

```

```

111 la date de debut et de fin de validite ainsi que le texte du message
112
113 Entrees :
114     int debut : date de debut de validite de message sous la forme AAAAMMJJ
115     int fin : date de fin de validite de message sous la forme AAAAMMJJ
116     char *message : texte du message
117
118 Sortie :
119     news_t* : pointeur sur l'element cree
120 */
121 news_t * creer_cell(int debut, int fin, char *message) {
122     news_t *elt = (news_t*) malloc(sizeof(news_t));
123     if (elt) {
124         elt->debut = debut;
125         elt->fin = fin;
126         elt->texte = (char*) malloc((strlen(message)+1)*sizeof(char));
127         strcpy(elt->texte,message);
128     }
129     return elt;
130 }

```

## 2.2 Gestion de news

```

Code C
1  /* gestion_news.h
2  Header
3
4  -----| GESTION DE NEWS PAR LISTE CHAINEE |-----
5
6  BARBESANGE Benjamin,
7  PISSAVY Pierre-Loup
8
9  ISIMA 1ere Annee, 2014-2015
10 */
11
12 #ifndef __GESTION_NEWS_H__
13 #define __GESTION_NEWS_H__
14
15     #include "liste_news.h"
16
17     int charger(cell_t **, char *);
18     int sauver(cell_t *, char *);
19     int getDate();
20     void afficher_messages_date(cell_t *, int);
21     void afficher_message(cell_t *);
22     void afficher_liste(cell_t *);
23     void afficher_messages_jour(cell_t *);
24     void afficher_messages_motif(cell_t *, char *);
25     void supprimer_obsoletes(cell_t **);
26     void remplacer_date(cell_t **, int, int);
27
28 #endif

```

```

1  /* gestion_news.c
2  Fonctions de gestion des news
3
4  -----| GESTION DE NEWS PAR LISTE CHAINEE |-----
5
6  BARBESANGE Benjamin,
7  PISSAVY Pierre-Loup
8
9  ISIMA 1ere Annee, 2014-2015
10 */
11
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <time.h>
15 #include <string.h>
16 #include "gestion_news.h"
17
18 /* taille maximale de la chaine de caractere du message */
19 #define SIZE_BUF 118
20
21 char buf[SIZE_BUF+1];
22
23 /* int charger(cell_t **liste, char *nom_fichier)
24 Permet de charger les donnees d'un fichier passe en parametre
25 dans une liste dont l'adresse du pointeur de tete est
26 egalement passe en parametre
27
28 Entrees :
29 cell_t **liste : adresse du pointeur de tete de liste dans
30 char *nom_fichier : chemin et nom du fichier a partir duquel charger la liste
31
32 Sortie :
33 Entier indiquant un code d'erreur
34 0 si aucune erreur
35 1 si le fichier ne s'est pas charge
36 2 si probleme d'allocation d'un element dans la liste
37 */
38 int charger(cell_t **liste, char *nom_fichier) {
39     int ret = 1;
40     FILE *fichier = fopen(nom_fichier, "r");
41     int debut, fin;
42     char *scan;
43     cell_t *tmp;
44     if (fichier) {
45         ret = 0;
46         while (!feof(fichier) && ret == 0 && fgetc(buf, SIZE_BUF+1, fichier)) {
47             /* suppression du caractere \n residuel si le texte fait moins de 100 caracteres */
48             scan = strchr(buf, '\n');
49             if (scan) {
50                 *scan = '\0';
51             }
52             sscanf(buf, "%d %d", &debut, &fin);
53             tmp = creer_cell(debut, fin, &buf[18]);
54             if (tmp) {
55                 ins_cell(liste, tmp);

```



```

56     } else {
57         /* Code erreur: allocation de cellule */
58         ret = 2;
59     }
60 }
61 fclose(fichier);
62 }
63 return ret;
64 }
65
66 /* int sauver(cell_t *liste, char *nom_fichier)
67 Sauvegarde les donnees de la liste chainee dans un fichier passe
68 en parametre
69
70 Entrees :
71     cell_t *liste : pointeur sur le premier element de la liste chainee
72     char *nom_fichier : chemin et nom du fichier dans lequel sauvegarder
73                         ce fichier sera cree et effacera tout autre fichier
74                         du meme nom dans le repertoire specifie s'il y en a
75
76 Sortie :
77     Entier renvoyant un code d'erreur
78     0 si aucune erreur
79     1 si probleme de creation du fichier
80 */
81 int sauver(cell_t *liste, char *nom_fichier) {
82     int ret = 1;
83     cell_t *cour = liste;
84     FILE *fichier = fopen(nom_fichier, "w+");
85     if (fichier) {
86         while (cour) {
87             fprintf(fichier, "%d %d %s\n", cour->debut, cour->fin, cour->texte);
88             cour = cour->suiv;
89         }
90         fclose(fichier);
91         ret = 0;
92     }
93     return ret;
94 }
95
96 /* int getDate()
97 Retourne la date du jour au format AAAAMMJJ
98
99 Entrees :
100     Aucune
101
102 Sortie :
103     Entier representant la date du jour de la forme AAAAMMJJ
104 */
105 int getDate() {
106     time_t now = time(NULL);
107     int datejour;
108     struct tm t = *localtime(&now);
109     datejour = (t.tm_year+1900)*10000+(t.tm_mon+1)*100+t.tm_mday;
110     return datejour;
111 }

```

```

112
113 /* void afficher_message(cell_t *m)
114 Affiche le message d'un element de la liste chainee sous la forme
115 "Debut : *debut*, Fin: *fin*, Message: *texte*"
116
117 Entrees :
118     cell_t *m : pointeur sur un element de la liste chainee
119
120 Sortie :
121     Aucune
122 */
123 void afficher_message(cell_t *m) {
124     printf("Debut: %d, Fin: %d, Message: %s\n", m->debut, m->fin, m->texte);
125 }
126
127 /* void afficher_liste(cell_t *l)
128 Affiche le contenu integral de la liste suivant le schema de la fonction
129 afficher_message(cell_t *m)
130
131 Entrees :
132     cell_t *l : pointeur sur le premier element de la liste
133
134 Sortie :
135     Aucune
136 */
137 void afficher_liste(cell_t *l) {
138     cell_t *cour = l;
139     while (cour) {
140         afficher_message(cour);
141         cour = cour->suiv;
142     }
143 }
144
145 /* void afficher_messages_date(cell_t *liste, int date)
146 Affiche les messages de la liste correspondant a la date passee en parametre
147
148 Entrees :
149     cell_t *liste : pointeur sur le premier element de la liste chainee
150     int date : date de debut des elements a traiter, de la forme AAAAMMJJ
151
152 Sortie :
153     Aucune
154 */
155 void afficher_messages_date(cell_t *liste, int date) {
156     cell_t *cour = liste;
157     while (cour && cour->debut > date) {
158         cour = cour->suiv;
159     }
160     while (cour) {
161         while (cour && cour->fin >= date) {
162             afficher_message(cour);
163             cour = cour->suiv;
164         }
165         if (cour) {
166             cour = cour->suiv;
167         }
168     }

```

```

168     }
169 }
170
171 /* void afficher_messages_jour(cell_t *liste)
172 Affiche les messages de la liste correspondant a la date du jour
173
174 Entrees :
175     cell_t *liste : pointeur sur le premier element de la liste chaine
176
177 Sortie :
178     Aucune
179 */
180 void afficher_messages_jour(cell_t *liste) {
181     afficher_messages_date(liste, getDate());
182 }
183
184 /* void afficher_messages_motif(cell_t *liste, char *motif)
185 Affiche les messages de la liste chaine correspondant a un motif
186 passe en parametre
187
188 Entrees :
189     cell_t *liste : pointeur sur le premier element de la liste chaine
190     char *motif : chaine de caractere representant le motif a chercher dans
191                 les messages de la liste
192
193 Sortie :
194     Aucune
195 */
196 void afficher_messages_motif(cell_t *liste, char *motif) {
197     cell_t *cour = liste;
198     while (cour) {
199         if (strstr(cour->texte, motif)) {
200             afficher_message(cour);
201         }
202         cour = cour->suiv;
203     }
204 }
205
206 /* void supprimer_obsoletes(cell_t **liste)
207 Supprime les messages devenus obsoletes dans la liste chaine
208 Les messages sont obsoletes si leur date de fin est anterieure a la date du jour
209
210 Entrees :
211     cell_t **liste : adresse du pointeur sur le premier element de la liste chaine
212
213 Sortie :
214     Aucune
215 */
216 void supprimer_obsoletes(cell_t **liste) {
217     int date = getDate();
218     cell_t **prec = liste;
219     while (*prec != NULL) {
220         if ((*prec)->fin < date) {
221             supp_cell(prec);
222         } else {
223             prec = &((*prec)->suiv);

```

```

224     }
225 }
226 }
227
228 /* void remplacer_date(cell_t **liste, int date, int nvdate)
229    Remplace la date de debut des messages ayant une date passee en parametre par une
230    autre date aussi passee en parametre
231    Cette fonction s'assure egalement que la liste reste trie
232
233    Entrees :
234        cell_t **liste : adresse du pointeur sur le premier element de la liste chaine
235        int date : date de debut des elements a traiter, de la forme AAAAMMJJ
236        int nvdate : nouvelle date de debut a assigner aux elements
237
238    Sortie :
239        Aucune
240 */
241 void remplacer_date(cell_t **liste, int date, int nvdate) {
242     cell_t *cour = *liste;
243     while (cour && cour->debut > date) {
244         cour = cour->suiv;
245     }
246     while (cour && cour->debut == date) {
247         if (cour->fin >= nvdate) {
248             cour->debut = nvdate;
249         }
250         cour = cour->suiv;
251     }
252     sauver(*liste, "/tmp/tmp_tp1_sdd.list");
253     liberer_liste(liste);
254     charger(liste, "/tmp/tmp_tp1_sdd.list");
255 }

```

## 2.3 Programme principal

Code C

```
1  /* main.c
2   Fichier principal permettant les tests
3
4   -----| GESTION DE NEWS PAR LISTE CHAINEE |-----
5
6   BARBESANGE Benjamin,
7   PISSAVY Pierre-Loup
8
9   ISIMA 1ere Annee, 2014-2015
10 */
11
12 #include <stdio.h>
13 #include "gestion_news.h"
14
15 int main(int argc, char *argv[]) {
16     int old_deb = 20150215;
17     int new_deb = 20150326;
18     cell_t *liste = NULL;
19     if (argc > 1) {
20         charger(&liste,argv[1]);
21     }
22     if (liste) {
23         printf("Affichage de la liste apres recuperation\n");
24         afficher_liste(liste);
25
26         printf("\nAffichage des messages du jour\n");
27         afficher_messages_jour(liste);
28
29         printf("\nSuppression des messages obsoletes\n");
30         supprimer_obsoletes(&liste);
31         afficher_liste(liste);
32
33         printf("\nModification des dates de debut: %d -> %d\n",old_deb,new_deb);
34         remplacer_date(&liste,old_deb,new_deb);
35         afficher_liste(liste);
36
37         liberer_liste(&liste);
38     }
39     return 0;
40 }
```

## 3 | Principes et lexiques des fonctions

Dans cette partie, sont décrits les algorithmes de principe associés aux fonctions écrites en langage C, ainsi qu'un lexique concernant les variables intermédiaires des fonctions.

Le lexique des variable d'entrée, sortie et entrée/sortie sont disponibles dans le code source directement.

### 3.1 Gestion des news

La gestion des news s'effectue grace aux fichiers `gestion_news.c` et `gestion_news.h`.

#### 3.1.1 charger

Algorithme Charger (Principe)

Début

On ouvre en lecture seule le fichier dont le nom est passé en paramètre;

On initialise le code de retour à 1; *[ erreur d'ouverture de fichier ]*

Si on a pu l'ouvrir Alors

Le code de retour passe à 0; *[ pas de problème d'ouverture de fichier ]*

TantQue l'on n'arrive pas à la fin du fichier Faire

On lit une ligne;

On stocke les date de début et de fin de message dans des variables;

On stocke le message dans une chaîne de caractère;

On crée une nouvelle cellule à partir des variables ci-dessus récupérées;

Si on a pu créer la cellule Alors

On l'insère dans la liste chaînée;

Sinon

Le code de retour passe a 2; *[ erreur allocation de cellule ]*

FinSi;

FinTantQue;

On ferme le fichier;

FinSi;

Retourner le code d'erreur;

Fin

#### Lexique :

- ret: entier qui retourne un code d'erreur
  - 0 si aucune erreur
  - 1 si problème lors de l'ouverture du fichier
  - 2 si problème lors de la création de la cellule
- \*fichier: descripteur de fichier, ouvert à partir du nom passé en paramètres
- debut, fin: entiers servant à stocker les dates de début et de fin de message lors du traitement du fichier
- \*scan: chaîne de caractère permettant de stocker le message lors du traitement du fichier
- \*tmp: pointeur de cellule qui servira de cellule temporaire à chaîner dans la liste

### 3.1.2 sauver

#### Algorithme Sauver (Principe)

##### Début

On crée un nouveau fichier à partir du nom donné en paramètre;  
On initialise le code de retour à 1; [ erreur de création de fichier ]  
**Si** l'on a pu créer le fichier **Alors**  
    **TantQue** l'on n'est pas à la fin de la liste **Faire**  
        On écrit les éléments de la cellule en cours de traitement dans le fichier;  
        On passe à la cellule suivante;  
    **FinTantQue**;  
On ferme le fichier;  
On passe le code de retour à 0; [ pas d'erreur ]  
**FinSi**;  
**Retourner** le code d'erreur;

##### Fin

#### Lexique :

- ret: entier représentant le code d'erreur de la fonction
  - 0 aucune erreur
  - 1 erreur de création ou d'ouverture du fichier
- \*cour: pointeur sur la cellule en cours de traitement dans la liste chaînée
- \*fichier: descripteur de fichier créé à partir du nom passé en paramètre

### 3.1.3 getDate

Cette fonction est fournie pour le TP, nous ne la détaillerons donc pas.

### 3.1.4 afficher\_message

#### Algorithme afficher\_message (Principe)

##### Début

On écrit les données d'une cellule en récupérant ses attributs;

##### Fin

### 3.1.5 afficher\_liste

Algorithme afficher\_liste (Principe)

Début

On initialise l'élément courant au début de la liste;  
TantQue l'élément courant n'est pas NULL Faire  
| On affiche les élément de la cellule avec *afficher\_message*;  
| On passe à la cellule suivante;  
FinTantQue;

Fin

Lexique :

| \*cour:pointeur sur la cellule en cours de traitement dans la liste chaînée

### 3.1.6 afficher\_messages\_date

Algorithme afficher\_messages\_date (Principe)

Début

On initialise l'élément courant au début de la liste;  
TantQue la date de début de l'élément courant est supérieure à la date à traiter Faire  
| On passe à la cellule suivante;  
FinTantQue;  
TantQue la liste n'est pas finie Faire  
| TantQue la date de fin de l'élément courant est ultérieure à la date recherchée Faire  
| | On affiche le message de l'élément courant;  
| | On passe à la cellule suivante;  
| FinTantQue;  
| Si la liste n'est pas finie Alors  
| | On passe à la cellule suivante;  
| FinSi;  
FinTantQue;

Fin

Lexique :

| \*cour:pointeur sur la cellule en cours de traitement dans la liste chaînée



### 3.1.7 afficher\_messages\_jour

**Algorithme** afficher\_messages\_jour (Principe)

**Début**

    On appelle la fonction *afficher\_messages\_date* avec en paramètre la date du jour obtenue avec *getDate*;

**Fin**

**Lexique :**

    Aucune variable intermédiaire n'est utilisée dans cette fonction:

### 3.1.8 afficher\_messages\_motif

**Algorithme** afficher\_messages\_motif (Principe)

**Début**

    On initialise l'élément courant au début de la liste;

**TantQue** l'élément courant n'est pas NULL **Faire**

**Si** le message contenu dans la cellule courant contient le motif **Alors**  
            On affiche ce message;

**FinSi**;

        On passe à l'élément suivant;

**FinTantQue**;

**Fin**

**Lexique :**

    \*cour: *Pointeur sur la cellule en cours de traitement dans la liste chaînée*

### 3.1.9 supprimer\_obsoletes

**Algorithme** supprimer\_obsoletes (Principe)

**Début**

    On récupère la date du jour dans un entier;

    On initialise un pointeur double \*\*prec au début de la liste (ceci afin d'économiser l'utilisation d'une variable supplémentaire);

**TantQue** \*prec n'est pas NULL **Faire**

**Si** la date de fin de message de \*prec est inférieure à la date du jour **Alors**  
            On supprime cette cellule à partir du précédent prec;

**Sinon**

            On passe à l'élément suivant;

**FinSi**;

**FinTantQue**;

**Fin**

Lexique :

| date: *Date du jour récupérée avec la fonction getDate*  
| \*\*prec: *Pointeur double servant à parcourir la liste en ayant toujours accès à l'élément précédent d'une cellule*

### 3.1.10 remplacer\_date

Algorithme remplacer\_date (Principe)

Début

| On initialise l'élément courant que début de la liste;  
| TantQue l'élément courant n'est pas NULL Et Alors La date de début de message de l'élément courant est supérieure à la date à modifier Faire  
| | On passe à l'élément suivant;  
| FinTantQue;  
| TantQue l'élément courant n'est pas NULL Et Alors La date de début de message de l'élément courant est égale à la date à modifier Faire  
| | Si la date de fin de l'élément courant est supérieure ou égale à la nouvelle date Alors  
| | | On modifie la date de début de message de l'élément courant à la nouvelle date passée en paramètre;  
| | FinSi;  
| | On passe à l'élément suivant;  
| FinTantQue;  
| On sauvegarde la liste courante dans un fichier temporaire;  
| On libère la liste;  
| On recharge la liste à partir du fichier temporaire, ce qui va recréer la liste triée;  
Fin

Lexique :

| \*cour: *Pointeur sur la cellule en cours de traitement dans la liste chaînée*

## 3.2 Gestion de la liste chaînée

### 3.2.1 adj\_cell

Algorithme adj\_cell (Principe)

Début

| On a \*\*prec un pointeur sur l'adresse de l'élément suivant à l'élément après lequel insérer;  
| L'élément suivant de l'élément à ajouter est \*prec, l'élément suivant de son précédent;  
| L'élément suivant du précédent devient l'élément à ajouter;

Fin

Lexique :

| Aucune variable intermédiaire n'est utilisée dans cette fonction:

### 3.2.2 rech\_prec

Algorithme rech\_prec (Principe)

Début

| On initialise \*\*prec, un pointeur sur l'adresse du début de la liste;

| TantQue \*prec n'est pas NULL Et Alors la date de début de \*prec est supérieure à la date après laquelle

insérer Faire

| | prec prend l'adresse de l'adresse de l'élément suivant dans la liste;

| FinTantQue;

| La variable d'entrée/sortie existe prend la valeur 1 si la date cherchée est présente, 0 sinon;

| Retourner prec;

Fin

Lexique :

| Aucune variable intermédiaire n'est utilisée dans cette fonction:

### 3.2.3 supp\_cell

Algorithme supp\_cell (Principe)

Début

| On sauvegarde l'adresse de l'élément à supprimer dans un pointeur temporaire;

| L'élément suivant du précédent devient l'élément suivant de l'élément à supprimer;

| On libère le texte de la cellule à supprimer (car celui-ci est alloué dynamiquement);

| On libère la cellule à supprimer;

Fin

Lexique :

| \*elt:pointeur temporaire sur la cellule à supprimer

### 3.2.4 liberer\_liste

Algorithme liberer\_liste (Principe)

Début

| TantQue la liste n'est pas vide Faire

| | On supprime les cellules une à une;

| FinTantQue;

| On place le pointeur de tête de la liste à NULL;

Fin

Lexique :

| Aucune variable intermédiaire n'est utilisée dans cette fonction:

### 3.2.5 ins\_cell

Algorithme ins\_cell (Principe)

Début

| On récupère un pointeur sur l'adresse de l'élément avant lequel insérer avec la fonction *rech\_prec*;  
| On ajoute l'élément dans la liste avec la fonction *adj\_cell* en utilisant les variables *prec* et le pointeur sur l'élément à ajouter;

Fin

Lexique :

| existe: *booléen de présence de l'élément recherché, modifié dans rech\_prec*  
| \*\*prec: *pointeur sur l'adresse de l'élément après lequel insérer*

### 3.2.6 creer\_cell

Algorithme (Principe)

Début

| On alloue un élément de la liste chaînée;  
| Si l'allocation a réussi Alors  
| | On place les dates de début et de fin de message dans les champs appropriés;  
| | On alloue l'espace nécessaire pour le message dans le champ texte;  
| | On recopie le texte passé en entrée dans le champ texte de la cellule;  
| FinSi;  
| Retourner le nouvel élément créé;

Fin

Lexique :

| \*elt: *pointeur sur le nouvel élément créé, ce sera également le retour de la fonction. Si celui-ci est à NULL, cela signifie qu'il y a eu un problème lors de son allocation*

## 4 | Compte rendu d'exécution

### 4.1 Makefile

```
1 #Compilateur et options de compilation
2 CC=gcc
3 CFLAGS=-Wall -ansi -pedantic -Wextra -g
4
5 #Fichiers du projet
6 SOURCES=main.c gestion_news.c liste_news.c
7 OBJECTS=$(SOURCES:.c=.o)
8
9 #Nom du programme
10 EXEC=programme
11
12 all: $(OBJECTS)
13     $(CC) $(CFLAGS) $^ -o $(EXEC)
14
15 .c.o:
16     $(CC) -c $(CFLAGS) $.c
17
18 clean:
19     rm $(OBJECTS) $(EXEC)
```

### 4.2 Jeux de tests

Exécution du programme avec le fichier suivant :

```
20150215 20150416 Ajout liste vide
20150220 20150227 Ajout en tete
20150217 20150228 Ajout en milieu de liste
20150212 20150222 Ajout apres le dernier element
20150215 20150316 Evenement lambda
```

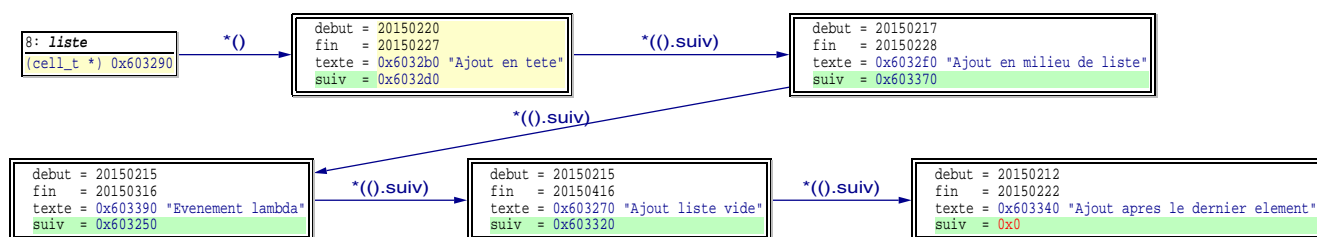


FIGURE 4.1 – Après lecture du fichier, contenu de liste

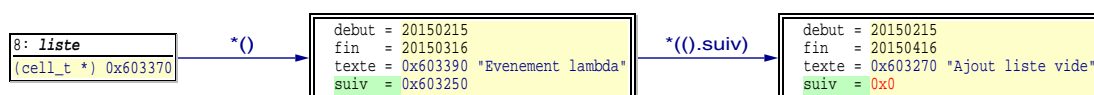


FIGURE 4.2 – Après suppression des messages obsolètes, contenu de liste

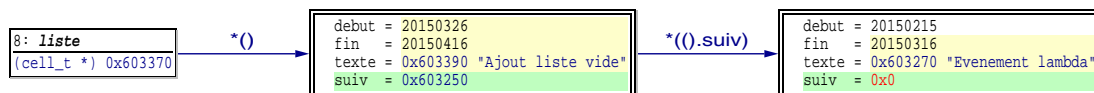


FIGURE 4.3 – Après modification de date de début, contenu de liste

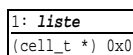


FIGURE 4.4 – Après suppression de la liste, contenu de liste

```

Terminal
Affichage de la liste apres recuperation
Debut: 20150220, Fin: 20150227, Message: Ajout en tete
Debut: 20150217, Fin: 20150228, Message: Ajout en milieu de liste
Debut: 20150215, Fin: 20150316, Message: Evenement lambda
Debut: 20150215, Fin: 20150416, Message: Ajout liste vide
Debut: 20150212, Fin: 20150222, Message: Ajout apres le dernier element

Affichage des messages du jour
Debut: 20150215, Fin: 20150316, Message: Evenement lambda
Debut: 20150215, Fin: 20150416, Message: Ajout liste vide

Suppression des messages obsoletes
Debut: 20150215, Fin: 20150316, Message: Evenement lambda
Debut: 20150215, Fin: 20150416, Message: Ajout liste vide

Modification des dates de debut: 20150215 -> 20150326
Debut: 20150326, Fin: 20150416, Message: Ajout liste vide
Debut: 20150215, Fin: 20150316, Message: Evenement lambda
  
```

FIGURE 4.5 – Exécution du programme sur la sortie standard

```

Terminal
==8377== Memcheck, a memory error detector
==8377== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==8377== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for copyright info
==8377== Command: ./programme test_insert
==8377==
Affichage de la liste apres recuperation
Debut: 20150220, Fin: 20150227, Message: Ajout en tete
Debut: 20150217, Fin: 20150228, Message: Ajout en milieu de liste
Debut: 20150215, Fin: 20150316, Message: Evenement lambda
Debut: 20150215, Fin: 20150416, Message: Ajout liste vide
Debut: 20150212, Fin: 20150222, Message: Ajout apres le dernier element

Affichage des messages du jour
Debut: 20150215, Fin: 20150316, Message: Evenement lambda
Debut: 20150215, Fin: 20150416, Message: Ajout liste vide

Suppression des messages obsoletes
Debut: 20150215, Fin: 20150316, Message: Evenement lambda
Debut: 20150215, Fin: 20150416, Message: Ajout liste vide

Modification des dates de debut: 20150215 -> 20150326
Debut: 20150326, Fin: 20150416, Message: Ajout liste vide
Debut: 20150215, Fin: 20150316, Message: Evenement lambda
==8377==
==8377== HEAP SUMMARY:
==8377==      in use at exit: 0 bytes in 0 blocks
==8377==    total heap usage: 28 allocs, 28 frees, 4,674 bytes allocated
==8377==
==8377== All heap blocks were freed -- no leaks are possible
==8377==
==8377== For counts of detected and suppressed errors, rerun with: -v
==8377== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

FIGURE 4.6 – Exécution avec valgrind

Voici un petit exemple de la recherche de motif sur le même fichier d'entrée. On recherche dans ce cas la chaîne "liste", immédiatement après avoir chargé le fichier dans la liste.

```

Terminal
Messages contenant "liste":
Debut: 20150217, Fin: 20150228, Message: Ajout en milieu de liste
Debut: 20150215, Fin: 20150416, Message: Ajout liste vide

```

FIGURE 4.7 – Exécution de la recherche de motif