

# SDD TP1 – Compte rendu

DHUMES Felix  
GAYTON Mathis

## Présentation générale

Le but de ce TP est de créer un module de gestion de messages à l'aide d'une liste chaînée. La structure de données comporte plusieurs données telles que les dates de début et de fin de validité du message. A l'aide du module, on pourra afficher seulement les messages valides, chercher un motif dans une chaîne par exemple.

Description et schéma de la structure de données et des fichiers de données utilisés

### Structure d'un message :

struct message : Chaque message est composé de 4 variables

- DebutValidite : entier représentant la date de début de validité du message sous la forme aaaammjj
- FinValidite: entier représentant la date de fin de validité du message aaaammjj
- texte : pointeur sur une chaîne de caractères représentant le message (de taille 100 au maximum)
- suivant : variable de type message \* qui est un pointeur sur le prochain message

```
DebutValidite = 19980712
FinValidite   = 20190901
texte         = 0x6046a0 "message valide 1"
suivant       = 0x604680
```

struct liste :

- premier : variable de type message \* qui est un pointeur sur le premier message de la liste chaînée. C'est la tête réelle de la liste.

```
1: liste
premier = 0x0
```

### Fichiers utilisés :

Les fichiers d'entrées et de sorties possèdent des messages et sont de la forme suivante :

Sur chaque ligne on retrouve un message (sous sa forme de structure de donnée) avec sa date de début de validité, sa date de fin de validité et le message. Chacune de ces données sont séparées par un espace.

```
20110429 20121221 message 1
19980712 20190901 message 2
```

### Organisation du code source :

listeChaine.h : Contient les librairies nécessaires, les deux structures de données (message et liste), les constantes (#define) pour la taille d'un message et la taille d'une date, et les prototypes des fonctions de listeChaine.c

listeChaine.c : Contient l'implémentation des fonctions de listeChaine.h

message.h : Contient les librairies nécessaires et les prototypes de message.c

message.c : Contient l'implémentation des fonctions de message.h

fichier.h : Contient les librairies nécessaires et les prototypes de fichier.c

fichier.c : Contient l'implémentation des fonctions de fichier.h

main.h : Contient les librairies nécessaires.

main.c : Contient les tests des fonctions.

Makefile : Permet la compilation du programme et crée l'exécutable sous le nom de main

### Fonctions contenues dans chaque fichier :

listeChaine.h

```
Void InitLCH(liste_t* liste);  
message_t * CreerCellule(int DebutValidite, int FinValidite, char * texte);  
void InsertionApres(message_t * message, message_t ** prec);  
message_t ** RecherchePrec(liste_t * liste, int valeur);  
void SupprimerCellule(message_t ** prec);  
void ParcoursLCH(liste_t * liste);  
void SupprimerLCH(liste_t * liste);
```

message.h

```
void AfficherMessagesValides(liste_t * liste);  
int RecupererDateSysteme();  
void SupprimerMessagesObsoletes(liste_t * liste);  
void ModifierDateDebut(liste_t * liste, int DateInitiale, int NouvelleDate);  
void MessagesContenantMotif(liste_t * liste, char * motif);  
int ContientMotif(char * s, char * motif);
```

fichier.h

```
void LectureFichier(char * NomFichier, liste_t * liste);
```

```
void suprRN(char* s, int tailleMax);
```

```
void SauvegardeLCH(char * NomFichier, liste_t * liste);
```

Pour lancer le programme, il faudra faire, après avoir entré les fonctions désirées dans le fichier main.c et fait make :

```
User@PC:~$ ./main fichier
```

Si on veut pouvoir enregistrer dans un autre fichier les messages, il faudra préciser un deuxième nom de fichier :

```
User@PC:~$ ./main fichierLecture fichierEcriture
```

## Compte rendu d'exécution :

### Jeux de test :

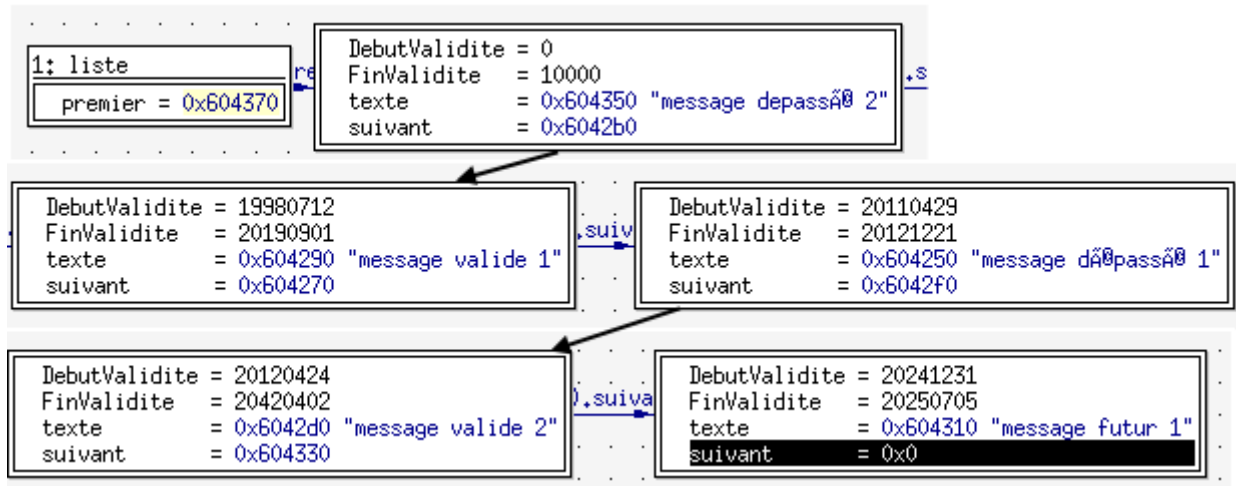
#### Lecture fichier :

Utilisation de la fonction `void LectureFichier(char * NomFichier, liste_t * liste);`

1. En entrée on utilise le fichier suivant :

```
20110429 · 20121221 · message · dépassé · 1  
19980712 · 20190901 · message · valide · 1  
20241231 · 20250705 · message · futur · 1  
00000000 · 00010000 · message · dépassé · 2  
20120424 · 20420402 · message · valide · 2
```

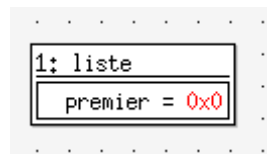
On obtient en sortie le résultat suivant :



On peut voir que les messages sont bien triés selon leur date de début de validité.

## 2. Avec un fichier vide

On obtient le résultat suivant :



Ce qui est le résultat attendu, car le fichier de départ ne possède pas de message.

## Modifier date debut :

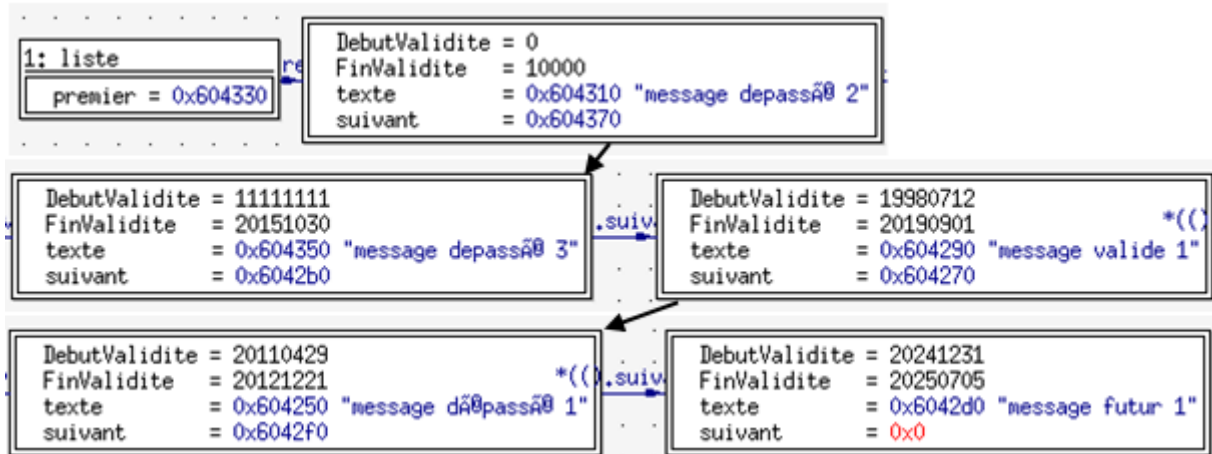
Utilisation de la fonction `void ModifierDateDebut(liste_t * liste, int DateInitiale, int NouvelleDate);`

1. On utilise le fichier suivant en entrée, en modifiant la date du

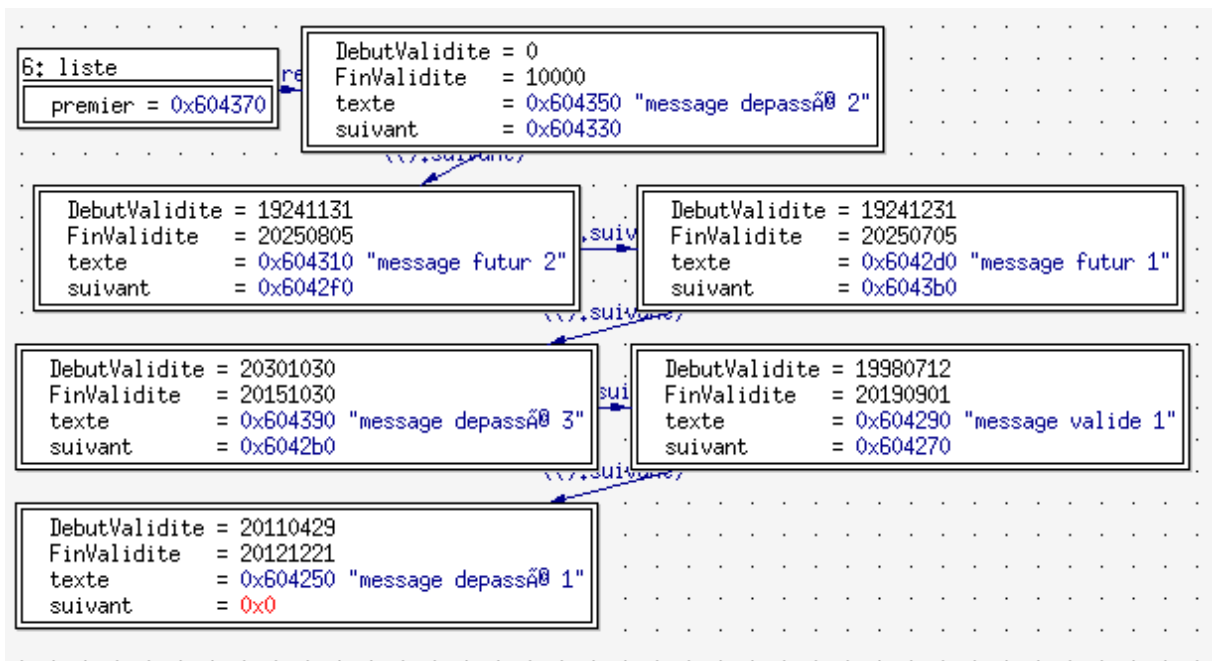
```
20110429 20121221 message dépassé 1
19980712 20190901 message valide 1
20241231 20250705 message futur 1
00000000 00010000 message dépassé 2
19851030 20151030 message dépassé 3
```

On obtient les résultats suivants avec ce fichier d'entrée :

- 1) en changeant la date 19851030 en 11111111 :

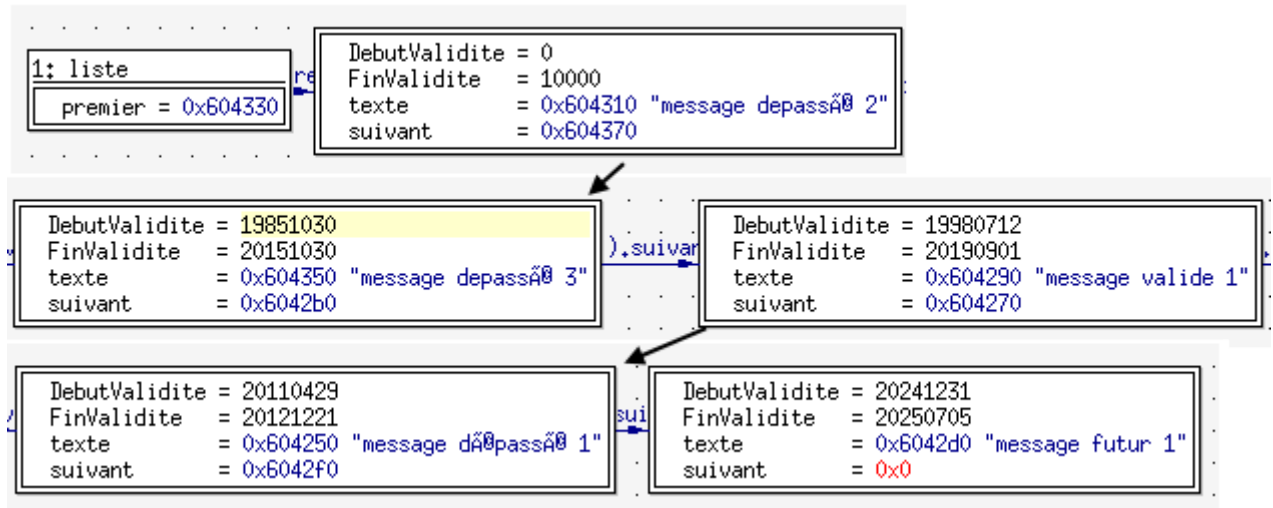


2) En changeant la date du 19851030 en 20301030 :

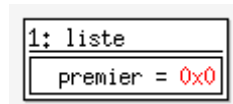


La liste ne se trie pas si la nouvelle date est postérieure à l'ancienne date, il faudra sauvegarder et rouvrir le fichier dans ce cas.

3) En changeant une date qui n'est pas présente dans la liste chaînée



#### 4) Avec un fichier vide



Ce qui est le résultat attendu, car le fichier de départ ne possède pas de message.

### Supprimer Messages Obsoletes :

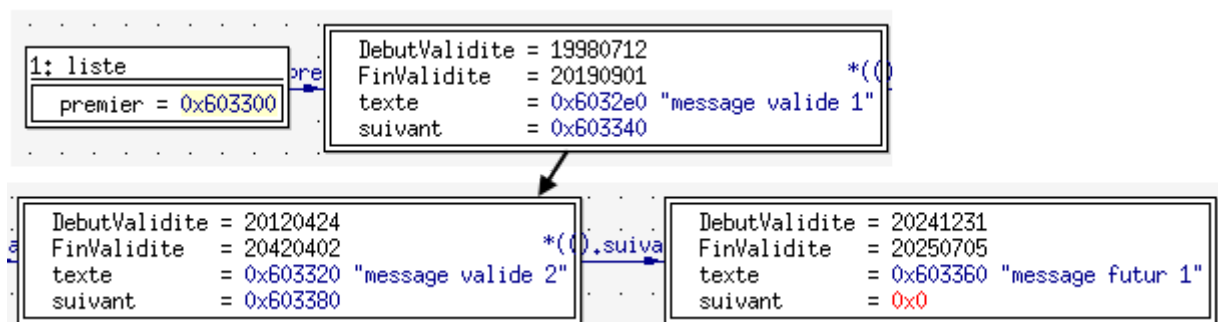
Utilisation de la fonction `void SupprimerMessagesObsoletes(liste_t * liste);`

#### 1. Avec un fichier normal :

```

20110429 20121221 message dépassé 1
19980712 20190901 message valide 1
20241231 20250705 message futur 1
00000000 00010000 message dépassé 2
20120424 20420402 message valide 2
  
```

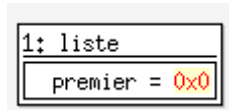
Résultat :



## 2. En mettant en entrée un fichier avec que des messages dépassés

```
00000000 00010000 message depasse 1
00002000 00003000 message depasse 2
19900101 19950101 message depasse 3
19851030 20151030 message depasse 4
```

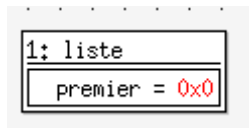
On obtient :



```
1: liste
premier = 0x0
```

Tous les messages du fichier étant dépassés, on les supprime tous, ce qui nous donne le bon résultat.

## 3. Avec un fichier vide



```
1: liste
premier = 0x0
```

Ce qui est le résultat attendu, car le fichier de départ ne possède pas de message.

## Sauvegarde liste :

Utilisation de la fonction `void SauvegardeLCH(char * NomFichier, liste_t * liste);`

### 1. Avec une liste non vide

En mettant en entrée

```
20110429 20121221 message dépassé 1
19980712 20190901 message valide 1
20241231 20250705 message futur 1
00000000 00010000 message dépassé 2
20120424 20420402 message valide 2
```

On obtient en sortie

```
00000000 00010000 message dépassé 2
19980712 20190901 message valide 1
20110429 20121221 message dépassé 1
20120424 20420402 message valide 2
20241231 20250705 message futur 1
```

Toute la liste a bien été sauvegardée, et triée selon la date du jour, le résultat est correct.



## 2. Avec une liste vide

La liste est vide, donc le fichier reste vide.

### Afficher messages Valides :

Utilisation de la fonction `void AfficherMessagesValides(liste_t * liste);`

Les différents résultats selon le fichier utilisé :

fichier vide

fichier normal

19980712 20190901 - message valide 1  
20120424 20420402 - message valide 2

fichier depasses

---

Le fichier normal et le fichier dépasse correspondent aux deux fichiers suivants :

20110429 20121221 - message dépassé 1	00000000 00010000 message depasse 1
19980712 20190901 - message valide 1	00002000 00003000 message depasse 2
20241231 20250705 - message futur 1	19900101 19950101 message depasse 3
00000000 00010000 - message dépassé 2	19851030 20151030 message depasse 4
20120424 20420402 - message valide 2	

Le résultat de l'exécution est correct :

Pour le fichier vide, il n'y a aucun message, donc aucun affichage.

Pour le fichier normal, seul les messages valides sont affichés.

Pour le fichier avec tous les messages dépassés, aucun n'est valide, il n'y a donc aucun affichage.

### Messages contenant motif :

Utilisation de la fonction `void MessagesContenantMotif(liste_t * liste, char * motif);`

Le motif non vide utilisé ici est « message depasse ». Les fichiers utilisés sont les fichiers vides et normaux :

GAYTON Mathis  
DHUMES Félix

```
20110429 20121221 message dépassé 1
19980712 20190901 message valide 1
20241231 20250705 message futur 1
00000000 00010000 message dépassé 2
19851030 20151030 message dépassé 3
```

On obtient :

```
chaîne vide
```

```
motif vide
```

```
00000000 00010000 - message depasse 2
19851030 20151030 - message depasse 3
19980712 20190901 - message valide 1
20110429 20121221 - message depasse 1
20241231 20250705 - message futur 1
```

```
motif et chaîne non vides
```

```
00000000 00010000 - message depasse 2
11111111 20151030 - message depasse 3
20110429 20121221 - message depasse 1
```

Il faut faire attention à ne pas mettre d'accents dans le fichier ou la recherche, car les accents font renvoyer des résultats erronés.

## Test Valgrind en appelant toutes les fonctions :

Traitement appliqué :

```
/* Traitement */
```

```
InitLCH(&liste);
LectureFichier(NomFichierLecture, &liste);
ModifierDateDebut(&liste, 19851030, 20301030);
SupprimerMessagesObsoletes(&liste);
MessagesContenantMotif(&liste, "message");
SauvegardeLCH(NomFichierEcriture, &liste);
```

```
SupprimerLCH(&liste);
```

```
/* Traitement */
```

## Resultat :

```
Felix@GE72VR-Felix:/mnt/d/Documents/TP/ZZ1/SDD/TP1$ valgrind ./main Tests/FichierNormal.txt output.txt
==205== Memcheck, a memory error detector
==205== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==205== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==205== Command: ./main Tests/FichierNormal.txt output.txt
==205==
==205== error calling PR_SET_PTRACER, vgdb might block
19241131 20250805 - message futur 2
19241231 20250705 - message futur 1
19980712 20190901 - message valide 1
==205==
==205== HEAP SUMMARY:
==205==    in use at exit: 0 bytes in 0 blocks
==205==   total heap usage: 23 allocs, 23 frees, 11,833 bytes allocated
==205==
==205== All heap blocks were freed -- no leaks are possible
==205==
==205== For counts of detected and suppressed errors, rerun with: -v
==205== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

## Makefile :

all: main.c listeChaine.c fichier.c message.c

gcc -g -Wall -Wextra -Wpedantic -ansi -Wno-endif-labels -o main main.c  
listeChaine.c fichier.c message.c

clean: rm main

## Code Source :

### listeChaine.h :

```
#ifndef LISTE_CHAINEE_H
#define LISTE_CHAINEE_H

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define TAILLE_TEXTE 100
#define TAILLE_DATE 9

typedef struct message {
    int DebutValidite;
```

```
    int FinValidite;
    char * texte;
    struct message * suivant;
} message_t;

typedef struct liste {
    struct message * premier;
} liste_t;

void InitLCH(liste_t * liste);
message_t * CreerCellule(int DebutValidite, int FinValidite, char * texte);
void InsertionApres(message_t * message, message_t ** prec);
message_t ** RecherchePrec(liste_t * liste, int valeur);
void SupprimerCellule(message_t ** prec);
void ParcourLCH(liste_t * liste);
void SupprimerLCH(liste_t * liste);

#endif // LISTE_CHAINEE_H
```

### listeChaine.c :

```
#include "listeChaine.h"

/*****
/*
/* InitLCH initialise une liste chaînée
/*
/* En entrée: liste un pointeur sur une structure liste
/*
/*
*****/

void InitLCH(liste_t * liste) {
    /* on teste si la liste a bien été allouée précédemment */
    if(liste == NULL) {
        exit(EXIT_FAILURE);
    }

    /* la liste est vide, le pointeur de tete doit pointer sur NULL */
    liste->premier = NULL;
}

/*****
/*
/* CreerCellule crée une cellule de liste chaînée à partir des données à stocker
/*
/*
/* En entrée: DebutValidite, FinValidite, texte les données à stocker dans la cellule
*****/
```

```
/* */
/* En sortie:  retourne l'adresse de la cellule créée */
/* */
/* Principe:   on alloue un espace mémoire qui contiendra un élément, et on le remplit */
/*             avec les valeurs données */
/* */
/* */
/*****/

message_t *CreerCellule(int DebutValidite, int FinValidite, char * texte) {
    message_t * cellule = (message_t *) malloc(sizeof(message_t)); /* adresse de la cellule créée */

    if(cellule == NULL) { /* on teste si l'allocation s'est bien passé */
        exit(EXIT_FAILURE);
    }
    cellule->DebutValidite = DebutValidite;
    cellule->FinValidite = FinValidite;
    cellule->texte = texte;
    cellule->suivant = NULL; /* par défaut, la cellule n'a pas de suivant */
    return cellule;
}

/*****/
/* */
/* InsertionApres  insere une cellule dans la liste chaînée apres une cellule de la liste donnée */
/* */
/* En entrée:  message un pointeur vers la cellule à ajouter à la liste */
/*             prec un pointeur sur l'attribut suivant d'une cellule de la liste */
/* */
/*****/

void InsertionApres(message_t * message, message_t ** prec) {
    message->suivant = *prec;
    *prec = message;
}

/*****/
/* */
/* RecherchePrec   recherche le precedent d'une valeur dans une liste chaînée */
/*                 selon la date de debut des messages */
/* */
/* En entrée:  liste un pointeur sur une structure liste initialisée */
/*             valeur la date pour laquelle on cherche le precedent */
/* */
/* En sortie:  retourne le précédant de la valeur */
/* */
/* Principe: on cherche le dernier message de la liste chaînée ayant une date de début */
/*            inférieure à celle donnée. La liste étant triée, on la parcourt depuis son début en */
/*            testant chaque élément */
/* */
/*****/
```

```

/*****/

message_t **RecherchePrec(liste_t * liste, int valeur) {
    message_t * cour = liste->premier; /* pointeur sur le message traité */
    message_t ** prec = &(amp;liste->premier); /* pointeur sur son précédent */

    /* tant que le message n'est pas apres la date cherchée et qu'il reste des messages dans
    la liste, on teste le message suivant*/
    while(cour != NULL && cour->DebutValidite < valeur) {
        prec = &(amp;cour->suivant);
        cour = cour->suivant;
    }

    return prec;
}

/*****/
/*
/* SupprimerCellule supprime une cellule de la liste chaînée et libère la mémoire
/*
/* En entrée: prec le precedent de la cellule à supprimer
/*
/*****/

void SupprimerCellule(message_t ** prec) {
    message_t * cour = *prec; /* pointeur sur le message à supprimer */

    if(cour != NULL) {
        *prec = (*prec)->suivant;
        free(cour->texte); /* on libere d'abord le texte du message */
        free(cour); /* puis on libere le message */
    }
}

/*****/
/*
/* ParcoursLCH afficher tout le contenu de la list chaînée
/*
/* En entrée: liste un pointeur sur une structure liste initialisée
/*
/*****/

void ParcoursLCH(liste_t * liste) {
    message_t * cour = liste->premier; /* pointeur sur le message traité */

    while(cour != NULL) { /* tant qu'il reste des messages à afficher */
        printf("%d %d %s\n", cour->DebutValidite, cour->FinValidite, cour->texte); /* on affiche */
        cour = cour->suivant;
    }
}

```

```
}

/*****
/*
/* SupprimerLCH supprime tous les éléments de la liste chaînée
/*
/* En entrée:  liste un pointeur sur une structure liste initialisée
/*
*****/

void SupprimerLCH(liste_t * liste) {
    while(liste->premier != NULL) { /* tant qu'il reste des messages dans la liste chaînée */
        SupprimerCellule(&(liste->premier)); /* on supprime le premier message */
    }
}
```

### message.h :

```
#ifndef MESSAGE_H
#define MESSAGE_H

#include <time.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "listeChaine.h"

void AfficherMessagesValides(liste_t * liste);
int RecupererDateSysteme();
void SupprimerMessagesObsoletes(liste_t * liste);
void ModifierDateDebut(liste_t * liste, int DateInitiale, int NouvelleDate);
void MessagesContenantMotif(liste_t * liste, char * motif);
int ContientMotif(char * s, char * motif);

#endif // MESSAGE_H
```

### message.c :

```
#include "message.h"

/*****
/*
/* RecupererDateSysteme    retourne la date au moment où la fonction est appelée
/*
/* En sortie:  La valeur retournée est un entier sous la forme aaaammjj
/*
*****/
```

```
int RecupererDateSysteme() {
    char        buffer[TAILLE_DATE]; /* stocke la date sous forme de texte */
    int         date; /* variable resultat */
    time_t      t = time(NULL);
    struct tm * tm = localtime(&t); /* on recupere la date avec une librairie C */

    /* on veut la date sous le format aaammjj: la documentation de la librairie
    nous indique d'utiliser ce formatage */
    strftime(buffer, TAILLE_DATE, "%Y%m%d", tm);

    /* la date retournée par la librairie est une chaine de caracteres; on la convertit en entier */
    date = atoi(buffer);
    return date;
}

/*****
/*
/* AfficherMessagesValides      affiche les messages valides sur la sortie standard
/*
/*
/*En entrée:    liste un pointeur sur une structure liste initialisée
/*
/*
*****/

void AfficherMessagesValides(liste_t * liste) {
    message_t * cour = liste->premier; /* pointeur sur le message traité */
    int         date = RecupererDateSysteme(); /* la date au moment où la fonction est appelée */

    while(cour != NULL) { /* tant qu'il reste des messages à traiter */
        /* si le message est valide */
        if(cour->DebutValidite < date && cour->FinValidite > date) {
            /* on l'affiche avec ses dates de debut et fin */
            printf("%d %d - %s\n", cour->DebutValidite, cour->FinValidite, cour->texte);
        }
        /* on passe au message suivant */
        cour = cour->suivant;
    }
}

/*****
/*
/* SupprimerMessagesObsoletes    supprime les messages dont la date de fin est dépassée de
/*
/*                                la liste chaînée
/*
/*
/*En entrée:    liste un pointeur sur une structure liste initialisée
/*
/*
*****/

void SupprimerMessagesObsoletes(liste_t * liste) {
    message_t * cour = liste->premier; /* pointeur sur le message traité */
```



```
message_t ** prec = &(liste->premier); /* pointeur sur son precedent */
int         date = RecupererDateSysteme(); /* la date au moment où la fonction est appelée */

while(cour != NULL) { /* tant qu'il reste des messages à traiter */
    if(cour->FinValidite < date) { /* si le message n'est plus valide */
        cour = cour->suivant; /* le message courant est le suivant, mais le precedent ne change pas */
    }

    SupprimerCellule(prec); /* on supprime le message invalide de la liste chaînée */
    }
    else { /* si il est valide */
        prec = &(cour->suivant); /* on passe au message suivant */
        cour = cour->suivant;
    }
}

}

/*****
/*
/* ModifierDateDebut      modifie la date de début de tous les messages ayant
/*                          une date de début donnée
/*
/*
/* En entrée:  liste un pointeur sur une structure liste initialisée
/*             DateInitiale la date de début à remplacer
/*             NouvelleDate la nouvelle date remplaçant DateInitiale
/*
/*
/* Principe: La liste chaînée étant triée, on cherche le premier message dont la date est
/*            à changer. Si on l'a trouvé, on change toutes les dates des messages à changer et on
/*            enregistre l'adresse du dernier message qui a changé. On cherche ensuite où on doit
/*            placer la nouvelle date dans la liste chaînée, et on fait les changements de pointeurs
/*            nécessaires pour garder l'ordre chronologique de début de message
/*
/*
*****/

void ModifierDateDebut(liste_t * liste, int DateInitiale, int NouvelleDate) {
    message_t ** prec = NULL;
    message_t ** prec1 = &(liste->premier);
    message_t * cour = liste->premier; /* pointeur sur le message traité */
    message_t * DebutChangement = NULL;
    message_t * FinChangement = NULL;

    while(cour != NULL && cour->DebutValidite < DateInitiale) {
        prec1 = &(cour->suivant);
        cour = cour->suivant;
    }

    if(cour != NULL) {
        DebutChangement = cour; /* debut des dates à changer */
        FinChangement = cour;
    }
}
```

```
while(cour != NULL && cour->DebutValidite == DateInitiale) {
    cour->DebutValidite = NouvelleDate;
    FinChangement = cour; /* fin des dates à changer */
    cour = cour->suivant;
}

prec = RecherchePrec(liste, NouvelleDate);

*prec1 = FinChangement->suivant;
FinChangement->suivant = *prec;
*prec = DebutChangement;
}
}

/*****
/*
/* MessagesContenantMotif affiche les messages contenant un motif (chaîne de caractères) donné
/*
/* quelque soit la date
/*
/*
/* En entrée: liste un pointeur sur une structure liste initialisée
/*
/* motif une chaîne de caractères
/*
/*
/* Principe: on regarde chaque message, et on teste si le message contient le motif. Si oui,
/*
/* on affiche le message
/*
/*
*****/

void MessagesContenantMotif(liste_t * liste, char * motif) {
    message_t * cour = liste->premier; /* pointeur sur le message traité */

    while(cour != NULL) { /* tant qu'il reste des messages à traiter */
        if(ContientMotif(cour->texte, motif)) { /* si le texte du message contient le motif recherché
*/
            /* on affiche le message avec ses dates de but et fin */
            printf("%08d %08d - %s\n", cour->DebutValidite, cour->FinValidite, cour->texte);
        }
        cour = cour->suivant;
    }
}

/*****
/*
/* ContientMotif fonction qui teste si une chaîne de caractères contient un motif donné
/*
/* quelque soit la date
/*
/*
/* En entrée: s, motif des chaînes de caractères
/*
/*
/* En sortie: 1 si s contient le motif, 0 sinon
/*
/*
*****/
```

```

/*****/

int ContientMotif(char *s, char *motif) {
    size_t TailleS = strlen(s); /* taille de la chaine de caracteres dans laquelle on cherche le motif */
    /*
    size_t TailleMotif = strlen(motif); /* taille de la chaine de caracteres du motif */
    size_t i = 0, j = 0; /* variables d'avancement dans les chaines de caracteres */
    int trouve = 0; /* variable retour */

    /* on teste si le motif se superpose depuis chaque caractere de s */
    while(i < TailleS - TailleMotif && !trouve) {
        j = 0;

        /* pour chaque lettre du motif, on verifie si elle se superpose à s */
        while(j < TailleMotif && motif[j] == s[i+j]) {
            j++;
        }
        /* le motif est trouvé si on sort de la boucle car on est arrivé à la fin du motif */
        trouve = (j == TailleMotif);
        i++;
    }

    return trouve;
}

```

### **Fichier.h :**

```

#ifndef FICHIER_H
#define FICHIER_H

#include <stdlib.h>
#include <string.h>
#include "listeChaine.h"

void LectureFichier(char * NomFichier, liste_t * liste);
void suprRN(char* s, int tailleMax);
void SauvegardeLCH(char * NomFichier, liste_t * liste);

#endif // FICHIER_H

```

### **Fichier.c :**

```

#include "fichier.h"

/*****/
/*
/* LectureFichier      construit une liste chaine de messages depuis un fichier de messages
/*
/* En entrée:  NomFichier le nom du fichier à lire depuis le repertoire de l'exécutable
/*
/*             liste un pointeur sur une structure liste initialisée
*/

```

```
/* */
/* En sortie: liste est une liste chaînée des messages du fichier d'entrée */
/* */
/* Principe: chaque ligne du fichier est un nouveau message. */
/* Pour chaque ligne, on lit d'abord la date de debut, la date de fin puis le message. */
/* On crée un nouveau message avec ces informations, qu'on insere au bon endroit dans la */
/* liste chaînée, de sorte qu'elle soit triée selon la date de début des messages */
/* */
/* */
/*****

void LectureFichier(char * NomFichier, liste_t * liste) {
    FILE      * fichier; /* descripteur de fichier de lecture */
    char      textebuf[TAILLE_TEXTE+1] = {0}; /* variable temporaire qui stocke des textes avant
traitement */
    int        LongueurTexte; /* variable temporaire contenant la taille du texte */
    int        DateDebut, DateFin;
    char       * texte;
    message_t * message; /* pointeur temporaire de message en construction */

    fichier = fopen(NomFichier, "r"); /* on ouvre le fichier en lecture seule */

    if(fichier != NULL) { /* si le fichier est valide */
        while(fgetc(fichier) != EOF) { /* tant qu'on a pas atteint la fin du fichier */
            fseek(fichier, -1, SEEK_CUR);
            /* on lit les dates de debut et fin */
            fscanf(fichier, "%d %d ", &DateDebut, &DateFin);
            /* on lit le reste de la ligne jusqu'à atteindre un retour à la ligne */
            fgets(textebuf, TAILLE_TEXTE, fichier);
            /* on supprime le retour à la ligne du message */
            suprRN(textebuf, TAILLE_TEXTE+1);

            LongueurTexte = strlen(textebuf);

            /* on alloue juste la place necessaire pour stocker le texte en mémoire */
            texte = (char *) malloc((LongueurTexte+1) * sizeof(char));
            strcpy(texte, textebuf);

            message = CreerCellule(DateDebut, DateFin, texte);

            /* on peut ajouter le message au bon endroit dans la liste */
            InsertionApres(message, RecherchePrec(liste, message->DebutValidite));
        }

        fclose(fichier); /* on pense à fermer le fichier puisqu'on a fini de l'utiliser */
    }
}

/*****
/* */
```

```
/* SauvegardeLCH      construit un fichier de messages depuis une liste chainee de messages */
/*
/* En entrée:  NomFichier le nom du fichier où écrire depuis le repertoire de l'exécutable */
/*             liste un pointeur sur une structure liste */
/*
/* En sortie:  un fichier representant la liste chainée est créée */
/*
/* Principe: pour chaque message de la liste chainée, on ajoute une ligne au fichier, sur
/* laquelle on copie la date de déb, la date de fin et le texte du message */
/*
/*****

void SauvegardeLCH(char * NomFichier, liste_t * liste) {
    FILE      * fichier; /* descripteur de fichier en écriture */
    message_t  * cour; /* pointeur sur le message traité */

    fichier = fopen(NomFichier, "w"); /* on ouvre un nouveau fichier en écriture */
    cour = liste->premier; /* le premier element à traiter est l'element pointé par la tete de liste */

    if(fichier != NULL) { /* si le fichier est valide */
        while(cour != NULL) { /* tant que l'element à traiter est un message */
            /* on imprime dans le fichier dans le bon format les informations du message */
            fprintf(fichier, "%08d %08d %s", cour->DebutValidite, cour->FinValidite, cour->texte);
            /* si le message a un suivant, on ajoute une ligne au fichier */
            if(cour->suivant != NULL) {
                fputc('\n', fichier);
            }
            /* on passe à l'élément suivant */
            cour = cour->suivant;
        }

        fclose(fichier); /* on pense à fermer le fichier */
    }
}

/*****
/*
/* suprRN      supprime les caractères de retour à la ligne en fin de chaine de caracteres */
/*
/* En entrée:  s une chaine de caracteres */
/*             taillemax la taille maximum de s (sans compter la fin de chaine '\0') */
/*
/* En sortie:  s sans '\r' ou '\n' */
/*
/* Principe: On parcourt la chaine jusqu'à rencontrer un caractere retour de ligne, un caractere */
/* fin de chaine, ou avoir atteint une taille maximum de chaine. On remplace le caractere */
/* à cet emplacement par un caractere fin de chaine */
/*
/*****
```

```
void suprRN(char* s, int tailleMax) {
    int i = 0;
    /* on cherche le premier caractere de retour à la ligne de la chaine de caracteres */
    while(i < tailleMax && (s[i] != '\n' && s[i] != '\r' && s[i] != '\0')) {
        i++;
    }
    /* on le remplace par un caractere de fin de chaine */
    s[i] = '\0';
}
```

### **Main.h :**

```
#include "listeChaine.h"
#include "fichier.h"
#include "message.h"
```

### **Main.c :**

```
#include "main.h"

int main(int argc, char ** argv) {
    liste_t liste;

    char* NomFichierLecture; /* Nom du fichier lu, passé en 1er paramètre du programme */
    char* NomFichierEcriture; /* Nom du fichier écrit, passé en 2eme paramètre du programme */

    if(argc == 2) {
        NomFichierLecture = argv[1];
        NomFichierEcriture = argv[1];
    }
    else if(argc == 3) {
        NomFichierLecture = argv[1];
        NomFichierEcriture = argv[2];
    }
    else {
        exit(EXIT_FAILURE);
    }

    /* Traitement */

    InitLCH(&liste);
```

GAYTON Mathis  
DHUMES Félix

```
LectureFichier(NomFichierLecture, &liste);  
ModifierDateDebut(&liste, 19851030, 20301030);  
SupprimerMessagesObsoletes(&liste);  
MessagesContenantMotif(&liste, "message");  
SauvegardeLCH(NomFichierEcriture, &liste);  
  
SupprimerLCH(&liste);  
  
/* Traitement */  
  
return 0;  
}
```