TP1:
Gestion de messages

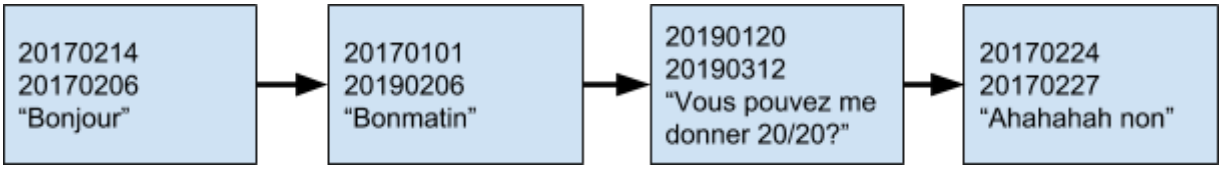
Présentation:

L'objectif de ce TP est de créer un utilitaire permettant de gérer des messages via une liste chaînée. Les messages sont constitué d'une date de début, d'une date de fin et du texte qui est le message. Le programme permet ainsi de lire les messages à partir d'un fichier, d'en modifier certains champs comme la date de début et d'enregistrer les messages dans un nouveau fichier. Il peut également afficher les messages non expirés à la date du jour par rapport à la date de fin, il peut afficher les messages contenant un certain motif et supprimer les messages expirés.

La structure message est composé de la manière suivante:

- un entier représentant la date de début du message.
- un entier représentant la date de fin du message.
- une chaîne de caractère permettant de stocker le message.
- un pointeur sur le message suivant de la liste.

Exemple d'une liste chaînée de structure message:



Les fichiers de donnée sont composé de la manière suivante:

Un fichier contient une structure message par ligne. Chaque ligne contient alors 3 colonnes:

- une colonne contenant un entier représentant la date de début.
- une colonne contenant un entier représentant la date de fin.
- une colonne contenant une chaîne de caractère qui est le message.

Exemple d'un fichier de données:

Date début	Date fin	Message
20170214	20170206	Bonjour
20190120	20190522	Bonmatin
20170101	20190206	Vous pouvez me donner 20/20?
20190120	20190312	Ahahahah non

Organisation du code source:

Les fonctions permettant de manipuler les listes chaînées de messages se trouvent dans un module (LC.c et LC.h). Quant au fichier tp1.c, il contient les fonctions gérant l'interface utilisateur et les fonctions permettant de manipuler les messages dans la liste chaînée ou dans des fichiers.

Compte rendu d'exécution:

Le fichier Makefile:

```
OPT = -g -Wall -Wextra
LOG = @echo "\#MAKE"

all: gestion-message

gestion-message: tp1.c LC.o
    gcc -o gestion-message tp1.c LC.o $(OPT)
    $(LOG) "Exécutable généré"

LC.o: LC.h LC.c
    gcc -o LC.o -c LC.c $(OPT)
    $(LOG) "lib liste chainee généré"

clean:
    rm *.o
    rm gestion-message
```

La règle LC.o permet de compiler le module de liste chaînée (LC.h et LC.c) et la règle gestion-message permet de créer l'exécutable en compilant le code source du programme principal tp1.c avec le module compilé LC.o.

Jeux de tests:

Liste des cas à traiter:

Concernant les fichiers:

- fichier incompatible avec le programme.
- fichier compatible avec le programme.
- le message du fichier fait plus de 100 caractères.
- le message est sur deux lignes.
- le fichier est vide.
- la date de fin est inférieur à la date de début.

Concernant l'utilisateur:

- un mauvais chiffre est rentré dans le menu.
- un chiffre correct est rentré dans le menu.
- une date erronée est rentrée pour la date à modifier, la nouvelle date ou les deux.
- une date correct est rentrée pour la date à modifier mais la nouvelle date rentrée est supérieure à la date de fin.
- une date correct est rentrée pour les deux dates.
- un motif non présent dans les messages est rentré.
- un motif présent dans un ou des messages est rentré.
- un motif plus long que le message est rentré.
- un motif vide est rentré.

Fichier incompatible avec le programme:

Soit le fichier "erreur.txt" contenant:

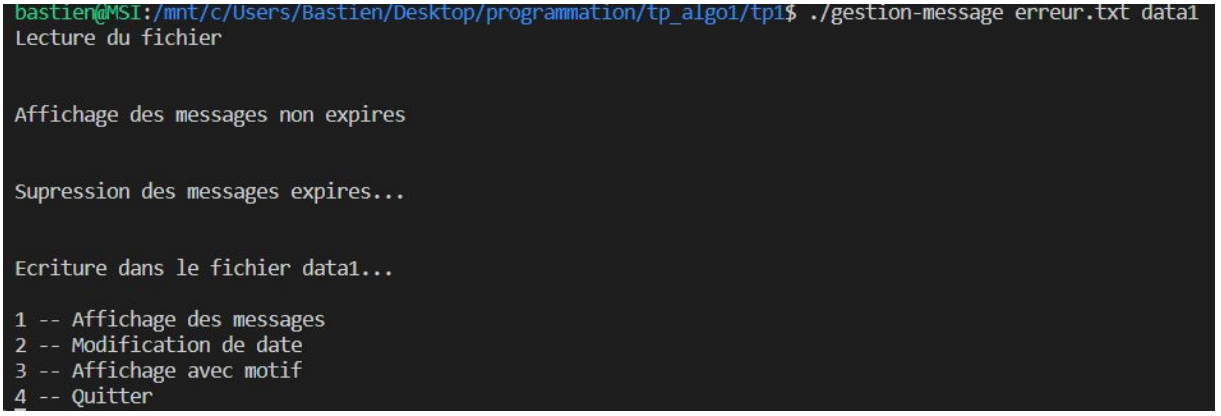
```
Bonjour je ne suis pas compatible avec le programme.
Mince alors!
Vous pouvez nous donner 20/20 ?
```

Exécutons le programme avec ce fichier en premier argument afin qu'il soit lu par le programme.

Données en entré:

```
./gestion-message erreur.txt data1
```

Résultat:



Les lignes non conforme au format(date date message) sont ignorés. En effet dans la lecture de fichier lorsque le programme n'arrive pas à lire 2 entiers représentant les dates il passe à la ligne suivante. Donc pour ce genre de fichiers d'entrées, le programme va jusqu'à la fin sans insérer un seul message dans la liste.

Fichier compatible avec le programme:

Soit le fichier "data" contenant:

```
20170214 20170206 bonjour test
20190120 20190522 tata
20170101 20190206 bonmatin ouai
20190120 20190312 toto
20170224 20170227 bonsoir
20180214 20190215 yolo
20180101 20190201 end
```

Exécutons le programme avec ce fichier en premier argument afin qu'il soit lu par le programme.
Données en entré:

```
./gestion-message data data1
```

Résultat:

```
bastien@MSI:/mnt/c/Users/Bastien/Desktop/programmation/tp_algo/tp1$ ./gestion-message data data1
Lecture du fichier
20170101        20190206        bonmatin ouai
20170214        20170206        bonjour test
20170224        20170227        bonsoir
20180101        20190201        end
20180214        20190215        yolo
20190120        20190312        toto
20190120        20190522        tata


Affichage des messages non expires
20190120        20190312        toto
20190120        20190522        tata


Supression des messages expires...


Ecriture dans le fichier data1...

1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
```

On constate qu’avec ce fichier le programme s'exécute correctement: en effet il lit le fichier, affiche les messages non expiré par rapport à la date d’aujourd’hui, supprime les messages expirés, écrit les messages non expirés dans le fichier “data1” et affiche le menu pour l’utilisateur.

Le message du fichier fait plus de 100 caractères:

Soit le fichier “tropLong.txt”contenant:

```
20170214 20170206 bonjour test
20190120 20190522 tata
20170101 20190206
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabccccccccccccccccccccccccccccccccccccc
cc
20190120 20190312 toto
20170224 20170227 bonsoir
20180214 20190215 yolo
20180101 20190201 end
```

Exécutons le programme avec ce fichier en premier argument afin qu’il soit lu par le programme.

Données en entré:

```
./gestion-message tropLong.txt data1
```

Résultat:

```
bastien@MSI:/mnt/c/Users/Bastien/Desktop/programmation/tp_algo/tp1$ ./gestion-message tropLong.txt data1
Lecture du fichier
20170101        20190206        aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
20170214        20170206        bonjour test
20170224        20170227        bonsoir
20180101        20190201        end
20180214        20190215        yolo
20190120        20190312        toto
20190120        20190522        tata


Affichage des messages non expires
20190120        20190312        toto
20190120        20190522        tata


Supression des messages expires...


Ecriture dans le fichier data1...

1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
```

On constate qu’avec ce fichier le programme s'exécute correctement sauf que le message le plus long se retrouve coupé au 100ème caractère.

Le message est sur deux lignes:

Soit le fichier “deuxLignes.txt” contenant:

```
20170214 20170206 bonjour test
20190120 20190522 tata
20170101 20190206 Hola voyageur
Quel bon vent vous amène ici?
20190120 20190312 toto
20170224 20170227 bonsoir
20180214 20190215 yolo
20180101 20190201 end
```

Exécutons le programme avec ce fichier en premier argument afin qu’il soit lu par le programme.

Données en entré:

```
./gestion-message deuxLignes.txt data1
```

Résultat:

Le programme gère ce cas de la même façon qu'un fichier "corrompus" : il ignore les lignes non conformes. Donc le message sera tronqué au retour à la ligne et le reste du message sera ignoré.

Le fichier est vide:

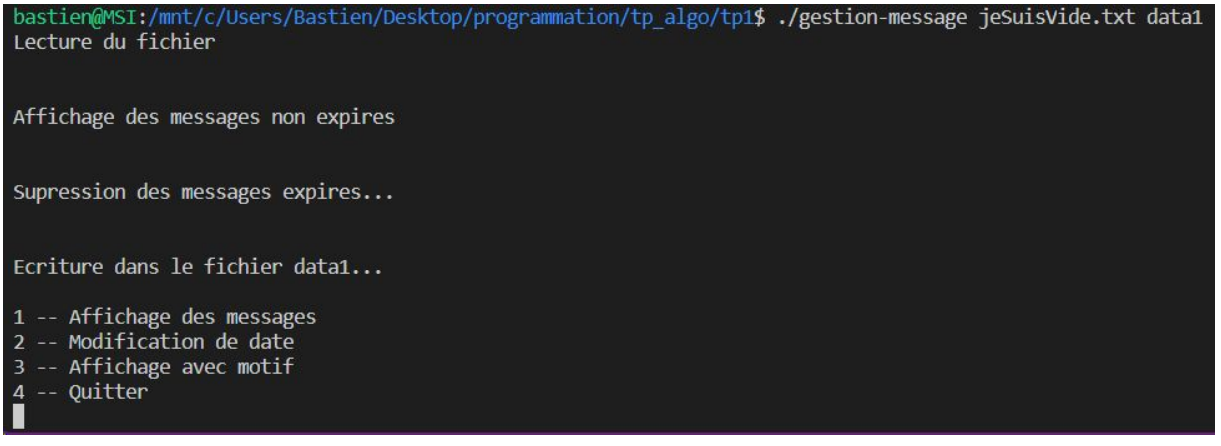
Soit le fichier "jeSuisVide.txt" contenant rien.

Exécutons le programme avec ce fichier en premier argument afin qu'il soit lu par le programme.

Données en entré:

```
./gestion-message jeSuisVide.txt data1
```

Résultat:



Avec ce fichier le programme s'exécute correctement car il ne trouve aucun message et donc conserve une liste chaînée vide.

La date de fin est inférieur à la date de début:

Soit le fichier ”datesInversées.txt” contenant:

```
20170214 20170206 bonjour test
20190522 20190120 tata
20190206 20170101 bonmatin ouai
20190120 20190312 toto
```

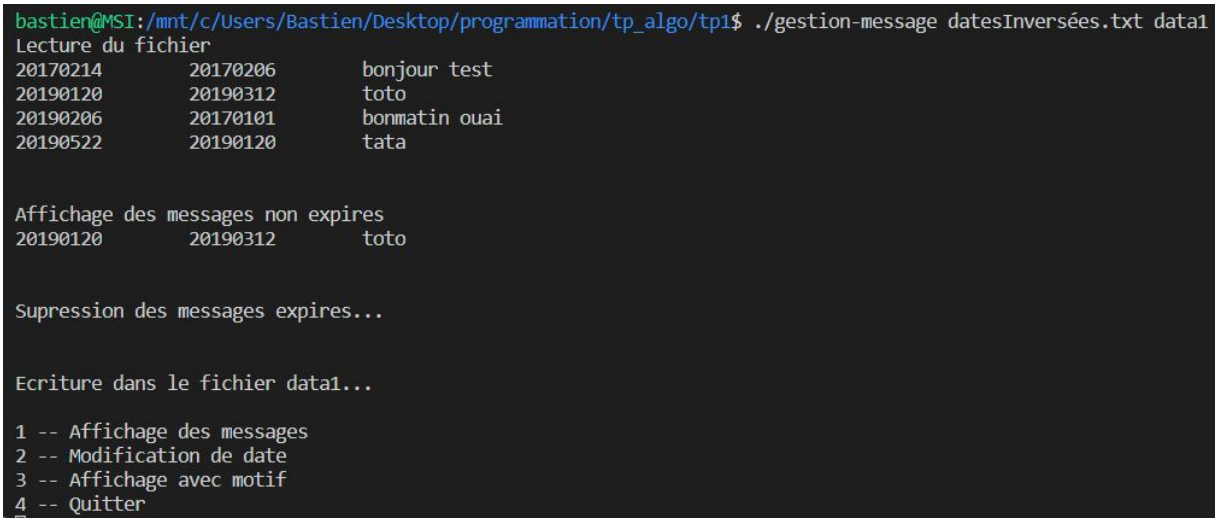
Les lignes surlignées en rose sont celles dont la date fut inversée.

Exécutons le programme avec ce fichier en premier argument afin qu’il soit lu par le programme.

Données en entré:

```
./gestion-message datesInversées.txt data1
```

Résultat:



Avec ce fichier le programme s’exécute malgrés les dates inversées. Le programme ne fait rien pour le gérer. C’est donc la tâche de l’utilisateur de fournir un fichier correct au programme.

Un mauvais chiffre est rentré dans le menu:

Exécutons le programme avec la commande:

```
./gestion-message data data1
```

Ainsi dans le menu utilisateur, rentrons le chiffre 42.

Résultat:

```
bastien@MSI:/mnt/c/Users/Bastien/Desktop/programmation/tp_algo/tp1$ ./gestion-message data data1
Lecture du fichier
20170101        20190206        bonmatin ouai
20170214        20170206        bonjour test
20170224        20170227        bonsoir
20180101        20190201        end
20180214        20190215        yolo
20190120        20190312        toto
20190120        20190522        tata


Affichage des messages non expires
20190120        20190312        toto
20190120        20190522        tata


Supression des messages expires...


Ecriture dans le fichier data1...

1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
42


1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
```

Ainsi on constate que le programme est fait de tel façon qu'il n'accepte que les entrées utilisateur valides. Il réaffiche donc le menu jusqu'à obtenir une entrée valide.

Un chiffre correct est rentré dans le menu:

Exécutons le programme avec la commande:

```
./gestion-message data data1
```

Chiffre 1:

```
1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
1

Messages:
20190120        20190312        toto
20190120        20190522        tata

1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
```

Chiffre 2:

```
1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
2

-- Modification de date --
Date à modifier :
20190120
Date modifiée :
19980721

Modif de la date 20190120 en 19980721
19980721        20190312        toto
19980721        20190522        tata

1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
```

Chiffre 3:

```
Modif de la date 20190120 en 19980721
19980721        20190312        toto
19980721        20190522        tata

1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
3

-- Affichage motif --
Motif à rechercher :
to
19980721        20190312        toto


1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
```

Chiffre 4:

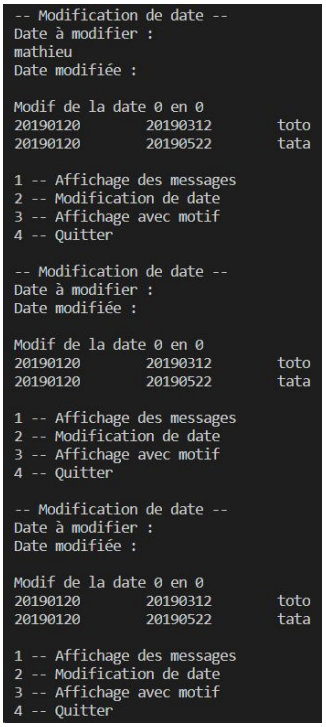
```
1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
4

bastien@MSI:/mnt/c/Users/Bastien/Desktop/programmation/tp_algo/tp1$
```

Pour chaque chiffre on obtient bien l'action désirée.

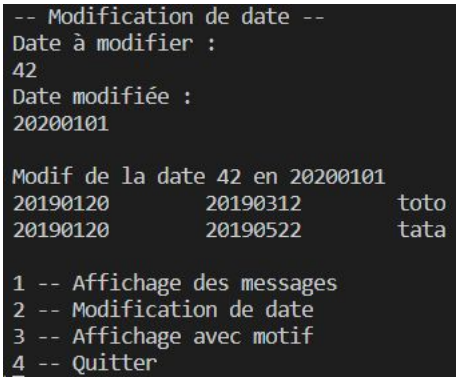
Une date erronée est rentrée pour la date à modifier, la nouvelle date ou les deux:

Date à modifier erronée (cas d'un chaîne de caractères):



On constate ainsi que si l'on rentre une chaîne de caractères ("mathieu" dans ce cas) le programme boucle à l'infini en essayant de modifier la date 0 en 0.

Date à modifier erronée (un chiffre ne correspondant pas à une date):



On constate qu'aucune modifications n'est apporté à la liste chaînée car le programme ne trouve aucune date correspondant à 42.

Nouvelle date erronée (cas d'un chaîne de caractères):

```
-- Modification de date --
Date à modifier :
20190120
Date modifiée :
matieu

Modif de la date 20190120 en 0
0       20190312        toto
0       20190522        tata

1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter

-- Modification de date --
Date à modifier :
Date modifiée :

Modif de la date 0 en 0
0       20190312        toto
0       20190522        tata

1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter

-- Modification de date --
Date à modifier :
Date modifiée :

Modif de la date 0 en 0
0       20190312        toto
0       20190522        tata

1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
```

On obtient un résultat similaire au cas de si on rentre une date à modifier erronée qui s'avère être une chaîne de caractères.

Nouvelle date erronée (un chiffre ne correspondant pas à une date):

```
-- Modification de date --
Date à modifier :
20190120
Date modifiée :
42

Modif de la date 20190120 en 42
42      20190312        toto
42      20190522        tata

1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
```

On constate donc que même si une date non conforme est rentrée le programme va la gérer et va quand même remplacer l'ancienne date.

Une date correct est rentrée pour la date à modifier mais la nouvelle date rentrée est supérieure à la date de fin:

Exécutons le programme en rentrant comme nouvelle date 20500101.

Résultat:

```
-- Modification de date --
Date à modifier :
20190120
Date modifiée :
20500101

Modif de la date 20190120 en 20500101
20500101        20500101        toto
20500101        20500101        tata

1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
```

Lorsque la nouvelle date est supérieur à la date de fin, cette dernière est également modifiée. Le message ne dure donc qu'un jour.

Une date correct est rentrée pour les deux dates:

Nous avons modifier la date de début de la ligne contenant le message "toto" dans le fichier data afin qu'il n'ai pas la même date de début que la ligne contenant le message "tata". Ainsi en exécutant le programme on obtient:

```
Lecture du fichier
20170101        20190206        bonmatin ouai
20170214        20170206        bonjour test
20170224        20170227        bonsoir
20180101        20190201        end
20180214        20190215        yolo
20190120        20190522        tata
20190121        20190312        toto


Affichage des messages non expires
20190120        20190522        tata
20190121        20190312        toto


Supression des messages expires...


Ecriture dans le fichier data1...

1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
```

Modifions la date de début du message contenant "toto" en rentrant la nouvelle date 20190201:

Résultat:

```
-- Modification de date --
Date à modifier :
20190121
Date modifiée :
20190201

Modif de la date 20190121 en 20190201
20190120        20190522        tata
20190201        20190312        toto

1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
```

Ainsi le programme modifie correctement la date de début indiquée parmis les messages non expirés.

Un motif non présent dans les messages est rentré:

Exécutons le programme et entrons le motif “ma”:

Résultat:

```
-- Affichage motif --
Motif à rechercher :
ma

1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
```

Le programme n’affiche rien car aucun message parmis les messages non expirés ne contient ce motif.

Un motif présent dans un ou des messages est rentré:

Exécutons le programme et entrons le motif “to”:

Résultat:

```
-- Affichage motif --
Motif à rechercher :
to
20190121        20190312        toto


1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
```

Le programme affiche le message contenant le motif parmis les messages non expirés.

Un motif plus long que le message est rentré:

Exécutons le programme et entrons le motif “anticonstitutionnellement”:

Résultat:

```
-- Affichage motif --
Motif à rechercher :
anticonstitutionnellement

1 -- Affichage des messages
2 -- Modification de date
3 -- Affichage avec motif
4 -- Quitter
```

Le programme n’affiche rien car aucun message parmis les messages non expirés ne contient ce motif.

Un motif vide est rentré:

Ce cas est impossible à observer car étant donné qu’on utilise un scanf, le programme attend forcément au moins caractère rentré de la part de l’utilisateur.

Détail de chaque fonction:

Fichier LC.h :

```
#ifndef _LC_H_
#define _LC_H_

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

struct Message
{
   int                 dateDebut;
   int                 dateFin;
   char              * message;
   struct Message    * suivant;
};
typedef struct Message Message_t;

Message_t*  creerElement         (int, int, const char*);
Message_t** rechercherPrecedent (Message_t**, Message_t*);
void        ajouterElement       (Message_t**, Message_t*);
void        adjonctionCellule    (Message_t**, Message_t*);
void        suppressionCellule   (Message_t**);
void        libererListe         (Message_t**);
void        debugListe           (Message_t*);
void        debugElement         (Message_t*);


#endif // !_LC_H_
```

Fichier LC.c :

```
#include "LC.h"

/*----------------------------------------------------------------
* creerElement         Creer une cellule "Message"
*
* Entrees: ddebut, dfin 2 entiers representant des dates
*          msg, une chaine de caracteres
*
* Sortie:  adresse d'une nouvelle cellule allouee
*
* On alloue un espace mémoire de la taille d'un élément et on
*  initialise ses champs avec les données passées en argument
*
* Lexique: element, pointeur sur le nouvel élément que l'on créer
*                    et que l'on initialise
*----------------------------------------------------------------
*/
```

```c
Message_t* creerElement(int ddebut, int dfin, const char* msg)
{
   Message_t * element;

   element = (Message_t*)malloc(sizeof(Message_t));
   if(element != NULL)
   {
       element->dateDebut = ddebut;
       element->dateFin = dfin;
       element->message = (char*)malloc(sizeof(char) * (strlen(msg)+1));
       strcpy(element->message, msg);
       element->suivant = NULL;
   }
   return element;
}

/*-------------------------------------------------------------------------
* rechercherPrecedent   Recherche le precedent d'une cellule donnee
*
* Entrees: liste, adresse du pointeur de liste
*          element, adresse de la cellule dont il faut le precedent
*
* Sortie:  adresse de la case "suivant" de la cellule precedente
*
*  En l'occurence, le precedent pointe sur le champs "suivant" de la cellule
*   précédente à la cellule dont la date de debut est egale ou superieur
*   a la date de debut de l'element donne
*
* Lexique: cour, pointeur sur l'élément courant dont on teste la date de début
*          prec, pointeur sur le champs suivant de l'élément précédent à cour
*-------------------------------------------------------------------------
*/
Message_t** rechercherPrecedent(Message_t** liste, Message_t* element)
{
   Message_t* cour = *liste;
   Message_t** prec = liste;

   while(cour != NULL && cour->dateDebut < element->dateDebut)
   {
       prec = &(cour->suivant);
       cour = cour->suivant;
   }
   return prec;
}

/*-------------------------------------------------------------------------
* ajouterElement   Ajoute une cellule a la liste de message
*
* Entrees: liste, adresse du pointeur de liste
*          element, adresse de la cellule qu'il faut ajouter a la liste
*
*   Fonction la uniquement pour "simplifier" dans le programme principal
*   Recherche le precedent avec rechercherPrecedent et appelle
*    adjonctionCellule avec celui-ci
```

```
*
* Lexique: prec, pointeur sur le champs suivant de l'element precedent
*                  a l'endroit ou devrait se situer "element"
*-------------------------------------------------------------------------
*/
void ajouterElement(Message_t** liste, Message_t* element)
{
   Message_t** prec = rechercherPrecedent(liste, element);
   adjonctionCellule(prec, element);
}

/*-------------------------------------------------------------------------
* adjonctionCellule   Ajoute une cellule a la liste de message
*
* Entrees: prec, adresse du champs "suivant" de la cellule precedente
*          element, adresse de la cellule qu'il faut ajouter a la liste
*
*   Edite les liens du precedent et de l'element pour l'ajouter dans la
*    liste chainee
*-------------------------------------------------------------------------
*/
void adjonctionCellule(Message_t** prec, Message_t* element)
{
   element->suivant = *prec;
   *prec = element;
}
/*-------------------------------------------------------------------------
* suppressionCellule   Supprime une cellule de la liste
*
* Entrees: prec, adresse du champs "suivant" de la cellule precedente
*
*   Edite les liens du precedent pour supprimer l'element de la liste
*    libere la memoire du message de l'element supprime
*    libere la memoire de l'element supprime de la liste
*
* Lexique: el, pointeur sur l'élément à supprimer (permet de le libérer une
*               fois la modification des liens faites)
*-------------------------------------------------------------------------
*/
void suppressionCellule(Message_t** prec)
{
   Message_t* el = *prec;
   *prec = el->suivant;
   free(el->message);
   free(el);
}
/*-------------------------------------------------------------------------
* libererListe   Supprime toutes les cellules de la liste
*
* Entrees: liste, adresse du pointeur de liste
*
*   Utilise suppressionCellule pour supprimer tous les elements
*    de la liste. On supprime l'élément en tête jusqu'à ce que la liste
*    soit vide
```

```
*-------------------------------------------------------------------------
*/
void libererListe(Message_t** liste)
{
   while(*liste != NULL)
   {
       suppressionCellule(liste);
   }
}

/*-------------------------------------------------------------------------
* debugListe   Affiche tout les elements d'une liste de message
*
* Entrees: liste, pointeur de la liste
*
*   Affiche les éléments de la liste de manière complète
*   (place dans la liste, adresse mémoire, contenu et adresse du suivant)
*
* Lexique: cour, pointeur sur l'élément à afficher
*          cpt, compteur d'élément pour afficher la place des éléments
*                dans la liste
*-------------------------------------------------------------------------
*/
void debugListe(Message_t* liste)
{
   Message_t* cour = liste;
   int cpt = 1;

   while(cour != NULL)
   {
       printf("Message %d:\n", cpt);
       debugElement(cour);
       cour = cour->suivant;
       cpt++;
   }
}

/*-------------------------------------------------------------------------
* debugElement   Affiche un element d'une liste de message
*
* Entrees: element, pointeur sur un structure message
*
*   Affiche un éléments de la liste de manière complète
*    (adresse mémoire, contenu et adresse du suivant)
*-------------------------------------------------------------------------
*/
void debugElement(Message_t* element)
{
   printf("\tpt: %p\n", element);
   printf("\tDate debut: %d\n", element->dateDebut);
   printf("\tDate fin: %d\n", element->dateFin);
   printf("\tMessage: %s\n", element->message);
   printf("\tSuivant: %p\n", element->suivant);
}
```

Fichier tp1.c :

```c
#include <stdio.h>
#include <time.h>
#include "LC.h"

void    menuModifDate       (Message_t*);
void    menuAfficheMotif    (Message_t*);
int     lireFichier         (Message_t**, const char*);
int     ecrireFichier       (Message_t*, const char*);
void    afficherNonExpire   (Message_t*);
void    supprimerExpire     (Message_t**);
void    modifDateDebut      (Message_t*, int, int);
void    afficherMotif       (Message_t*, const char*);
char*   rechercherMotif     (char*, const char*);
int     dateAj              ();
int     formateChaine       (char*);
void    afficherElement     (Message_t*);
void    afficherListe       (Message_t*);

/*-------------------------------------------------------------------------
* main   Point d'entree du programme
*
* Entrées: argc, entier representant le nombre d'arguments passes au programme
*          argv, tableau de chaine de caractere representants les arguments
*                  passes au programme (en l'occurence le nom du fichier d'entree
*                   et le nom du fichier de sortie)
*
*  Si on n'a pas de nom de fichier d'entree et de nom de fichier de sortie le
*      programme affiche simplement "Pas de fichier" et s'arrete.
*  Sinon on lis le fichier donne, on en creer une liste de message puis
*      on affiche les messages non expires, on les supprime de la liste et
*      on ecris dans le fichier de sortie la liste.
* Puis l'utilisateur peut choisir d'afficher la liste, de modifier la date,
*      d'afficher les messages avec un motif ou de quitter le programme.
*
* Lexique: maListe, pointeur de tete sur la liste chainnee de message
*          choix, entier representant le choix de l'utilisateur pour les menus
*-------------------------------------------------------------------------
*/
int main(int argc, char* argv[])
{
   Message_t*  maListe = NULL;
   int         choix = 0;

   if(argc < 3)
   {
       printf("Pas de fichier\n");
   }
   else
   {
       printf("Lecture du fichier\n");
       if(lireFichier(&maListe, argv[1]))
           afficherListe(maListe);
       else
```

```
            printf("Erreur sur le fichier\n");

        printf("\n\nAffichage des messages non expires\n");
        afficherNonExpire(maListe);

        printf("\n\nSupression des messages expires...\n");
        supprimerExpire(&maListe);

        printf("\n\nEcriture dans le fichier %s...\n", argv[2]);
        if(!ecrireFichier(maListe, argv[2]))
            printf("Erreur sur le fichier\n");

        do {
            do {
                printf("\n");
                printf("1 -- Affichage des messages\n");
                printf("2 -- Modification de date\n");
                printf("3 -- Affichage avec motif\n");
                printf("4 -- Quitter\n");
                scanf("%d", &choix);
                printf("\n");
            } while(choix != 1 && choix != 2 && choix != 3 && choix != 4);

            switch(choix)
            {
            case 1:
                printf("Messages:\n");
                afficherListe(maListe);
                break;
            case 2:
                menuModifDate(maListe);
                break;
            case 3:
                menuAfficheMotif(maListe);
                break;
            default:
                break;
            }

        } while(choix != 4);

        libererListe(&maListe);
    }

    return 0;
}
/*-------------------------------------------------------------------------
* menuModifDate   Permet à l'utilisateur de modifier une date de début
*
* Entrées: liste, pointeur de la liste
*
*  On recupere de l'utilisateur la date de debut des messages qui seront a
*    modifies et la date de debut qui la remplacera puis on appelle la fonction
*    "modifDateDebut".
```

```
*
* Lexique: dateDebut, entier qui represente la date de debut a modifier
*          dateModif, entier qui represente la date modifiee
*-----------------------------------------------------------------------------
*/
void menuModifDate(Message_t* liste)
{
   int dateDebut = 0, dateModif = 0;

   printf("-- Modification de date --\n");
   printf("Date à modifier :\n");
   scanf("%d", &dateDebut);
   printf("Date modifiée :\n");
   scanf("%d", &dateModif);
   printf("\nModif de la date %d en %d\n", dateDebut, dateModif);
   modifDateDebut(liste, dateDebut, dateModif);
   afficherListe(liste);
}
/*-----------------------------------------------------------------------------
* menuAfficheMotif   Permet à l'utilisateur d'afficher les messages selon un motif
*
* Entrées: liste, pointeur de la liste
*
*  On recupere de l'utilisateur le motif, on la formate en chaine de caractere
*      (finissant par '\0') et on appelle la fonction "afficherMotif".
*
* Lexique: motif, un tableau de caracteres contenant le motif (max: 255 caracteres)
*-----------------------------------------------------------------------------
*/
void menuAfficheMotif(Message_t* liste)
{
   char motif[255];
   printf("-- Affichage motif --\n");
   printf("Motif à rechercher :\n");
   scanf("%s", motif);
   formateChaine(motif);
   afficherMotif(liste, motif);
}
/*-----------------------------------------------------------------------------
* lireFichier  Lit un fichier texte et stocke sont contenu dans une liste
*              de messages
*
* Entrees: liste, adresse du pointeur de liste
*          filename, chaine de caractere contenant le nom du fichier
*
* Sortie: erreur, un entier que l'on renvoie a la fin de la foncion
*                 1 -> le fichier a bien ete ouvert
*                 0 -> le fichier n'a pas pu etre ouvert
*
*  Cette fonction lit le fichier filename ligne par ligne. En effet sur
*  une ligne du fichier se trouve dateDebut, dateFin et le message. Une
*  fois la ligne lue, on stocke son contenu dans un nouvel element de
*  type Message_t grace a la fonction creerElement et que l'on insere
*  dans la liste en respectant le tri grace a la procedure ajouterElement
```

```
*
* Lexique: file, un pointeur sur le fichier
*          ddebut, dfin, 2 entiers contenant les dates du message lu dans le fichier
*          msg, un tableau de caractere contenant la chaine de caractere du message
*                  lu dans le fichier (max: 100 caracteres)
*          element, pointeur du nouvel element cree a chaque message lu dans le fichier
*-----------------------------------------------------------------------
*/
int lireFichier(Message_t** liste, const char* filename)
{
   int         erreur = 1;
   FILE*       file = fopen(filename, "r");
   int         ddebut, dfin;
   char        msg[100];
   Message_t*  element;

   if(file != NULL)
   {
       while(!feof(file))
       {
           if(fscanf(file, "%d %d", &ddebut, &dfin) == 2) // On créer un nouvel élément
seulement si les 2 valeurs (début et fin) sont lues
           {
               fgetc(file); // On ignore l'espace entre la date de fin et le message
               fgets(msg, 100, file); // Message de 100 caractere maximum
               if(!formateChaine(msg)) // Finis par '\0' donc on doit aller a la ligne
suivante
                   while(fgetc(file) != EOF && fgetc(file) != '\n');

               if((element = creerElement(ddebut, dfin, msg)) != NULL)
                   ajouterElement(liste, element);
           }
           else
           {
               while(fgetc(file) != EOF && fgetc(file) != '\n'); // Si je la ligne ne
commence pas par 2 entiers (les dates) alors on va a la ligne suivante
           }

       }
       fclose(file);
   }
   else
   {
       erreur = 0;
   }

   return erreur;
}
/*-----------------------------------------------------------------------
*  ecrireFichier Ecrit le contenu d'une liste de messages dans un fichier
*
*  Entrees: liste, pointeur de la liste
*           filename, chaine de caractere contenant le nom du fichier
*
```

```
* Sortie: erreur, un entier que l'on renvoie a la fin de la foncion
*                  1 -> le fichier a bien ete ouvert
*                  0 -> le fichier n'a pas pu etre ouvert
*
*  Cette fonction va stocker chaque elements de la liste dans le fichier
*  a raison d'une ligne par elements.
*
* Lexique: file, un pointeur sur le fichier
*          cour, un pointeur sur le message de la liste que l'on ecrit
*-------------------------------------------------------------------------
*/
int ecrireFichier(Message_t* liste, const char* filename)
{
   int         erreur = 1;
   FILE*       file = fopen(filename, "w");
   Message_t*  cour = liste;

   if(file != NULL)
   {
       while(cour != NULL)
       {
           fprintf(file, "%d %d %s\n", cour->dateDebut, cour->dateFin, cour->message);
           cour = cour->suivant;
       }
       fclose(file);
   }
   else
   {
       erreur = 0;
   }

   return erreur;
}
/*-------------------------------------------------------------------------
*  afficherNonExpire   Affiche tous les messages non expires
*
*  Entrees:    liste, pointeur de la liste
*
*  Parcours les messages de la liste et n'affiche que ceux dont la date de
*      fin est superieur ou egal a la date d'aujourd'hui.
*
* Lexique: cour, pointeur parcourant les messages de la liste
*          today, entier representant la date d'aujourd'hui acquise par
*                  la fonction "dateAj"
*-------------------------------------------------------------------------
*/
void afficherNonExpire(Message_t* liste)
{
   Message_t*  cour = liste;
   int         today = dateAj();

   while(cour != NULL)
   {
       if(cour->dateFin >= today)
```

```
        {
            afficherElement(cour);
        }
        cour = cour->suivant;
    }
}
/*-------------------------------------------------------------------------
*  supprimerExpire   Supprime tous les messages dont leur dateFin est
*                      inferieur a la date d'aujourd'hui
*
*  Entrees:    liste, adresse du pointeur de la liste
*
*  Parcours les messages de la liste et supprime les messages dont la date
*      de fin est inferieur a la date d'aujourd'hui grace a un precedent et
*      a la fonction "suppressionCellule".
*
* Lexique: cour, pointeur parcourant les messages de la liste
*          prec, pointeur sur le champs suivant de l'élément précédent à cour
*          aj, entier representant la date d'aujourd'hui acquise par
*                  la fonction "dateAj"
*-------------------------------------------------------------------------
*/
void supprimerExpire(Message_t** liste)
{
    Message_t*  cour = *liste;
    Message_t** prec = liste;
    int         aj = dateAj();

    while(cour != NULL)
    {
        if(cour->dateFin < aj)
        {
            suppressionCellule(prec);
            cour = (*prec);
        }
        else
        {
            prec = &(cour->suivant);
            cour = cour->suivant;
        }
    }
}
/*-------------------------------------------------------------------------
*  modifDateDebut   Remplace des dates de debut de messages par une autre
*
*  Entrees:    liste, pointeur de la liste
*              dateAModif, entier representant la date de debut a modifier
*              nDate, entier representant la nouvelle date a mettre dans
*                  les messages ayant dateAModif en date de debut
*
*  Parcours la liste de message, teste si la date de debut correspond a
*      dateAModif et si c'est le cas la change avec nDate.
*
* Lexique: cour, pointeur parcourant les messages de la liste
```

```
*-----------------------------------------------------------------------
*/
void modifDateDebut(Message_t* liste, int dateAModif, int nDate)
{
   Message_t* cour = liste;

   while(cour != NULL)
   {
       if(cour->dateDebut == dateAModif)
       {
           cour->dateDebut = nDate;
           if(cour->dateFin < nDate)
               cour->dateFin = nDate;
       }
       cour = cour->suivant;
   }
}
/*-----------------------------------------------------------------------
*  afficherMotif   affiche les messages de la liste ayant un texte avec un
*                      motif donne
*
*  Entrees:    liste, pointeur de la liste
*              motif, tableau de caractere contenant le motif a rechercher
*
*  Parcours la liste de message, teste si le texte du message contient le
*     motif grace a la fonction "rechercherMotif" et si c'est le cas l'affiche
*
* Lexique: cour, pointeur parcourant les messages de la liste
*          avecMotif, pointeur sur le premier caractere du motif dans le
*                      message si il y en a un, NULL sinon.
*-----------------------------------------------------------------------
*/
void afficherMotif(Message_t* liste, const char* motif)
{
   Message_t*  cour = liste;
   char*       avecMotif;

   while(cour != NULL)
   {
       avecMotif = NULL;
       if((avecMotif = rechercherMotif(cour->message, motif)) != NULL)
       {
           afficherElement(cour);
           printf("\n");
       }
       cour = cour->suivant;
   }
}
/*-----------------------------------------------------------------------
*  rechercherMotif   recherche la premiere occurence d'un motif dans une
*                      chaine de caractere
*
*  Entrees:    message, chaine de caractere dans laquelle on cherche un motif
*              motif, chaine de caractere contenant le motif a rechercher
```

```
*
* Sortie: debut, pointeur de caractere pointant sur le debut du motif trouve
*                 dans le message ou NULL si il n'y en a pas
*
*  Parcours le message, si les caracteres sont en commun avec ceux du
*      motif on parcours egalement le motif. Si on arrive a la fin du
*      motif on s'arrete car on a trouve le motif. Si on trouve un caratere
*      different on continue le parcours du message et on recommence le
*      parcours du motif. (Et cela jusqu'a la fin du message)
*
* Lexique: cour, pointeur de caractere parcourant le message
*          courMotif, pointeur de caractere parcourant le motif
*-----------------------------------------------------------------------
*/
char* rechercherMotif(char* message, const char* motif)
{
   char*       cour      = message;
   const char* courMotif = motif;
   char*       debut;

   while(*cour != '\0' && *courMotif != '\0')
   {
       debut = cour;
       while(*courMotif != '\0' && *cour != '\0' && *courMotif == *cour)
       {
           courMotif++;
           cour++;
       }
       if(*courMotif != '\0')
       {
           courMotif = motif;
           cour++;
       }
   }

   if(*courMotif != '\0')
       debut = NULL;

   return debut;
}
/*-----------------------------------------------------------------------
*  dateAj   renvoie la date d'aujourd'hui (date systeme)
*
* Sortie: entier representant la date d'aujourd'hui sous la forme "aaaammjj"
*
*  Utilise les structures et les fonctions systemes (time_t, struct tm,
*      time(), localtime() et strftime()) pour ecrire dans lun buffer la date
*      systeme et la renvoyer sous forme entiere.
*
* Lexique: t, temps systeme
*          date, structure contenant la date du temps systeme
*          buffer, tableau de caractere contenant la date sous la forme "aaaammjj"
*-----------------------------------------------------------------------
*/
```

```c
int dateAj()
{
   time_t      t;
   struct tm * date;
   char        buffer[10];

   time(&t);
   date = localtime(&t);
   strftime(buffer, sizeof(buffer), "%Y%m%d", date);

   return atoi(buffer);
}
/*-------------------------------------------------------------------------
*  formateChaine   formate un tableau de caractere en chaine de caractere
*
* Entrees: chaine, tableau de caractere finissant pas '\0' ou '\n'
*
* Sortie: modif, entier (booleen) 0 -> la chaine n'a pas ete modifiee
*                                 1 -> la chaine a ete modifiee
*
*  Parcours le tableau jusqu'au '\0' ou '\n'. Si c'est un '\n',
*      le remplace par '\0' et renvoie 1 sinon renvoie 0;
*
* Lexique: i, entier permettant de parcourir le tableau
*-------------------------------------------------------------------------
*/
int formateChaine(char* chaine)
{
   int i = 0;
   int modif = 0;
   while(chaine[i] != '\0' && chaine[i] != '\n')
       i++;

   if(chaine[i] == '\n')
   {
       chaine[i] = '\0';
       modif = 1;
   }
   return modif;
}
/*-------------------------------------------------------------------------
* afficherListe   Affiche tout les elements d'une liste de message
*
* Entrees: liste, pointeur de la liste
*
*  Parcours la liste et en affiche tous les elements
*
* Lexique: cour, pointeur de message parcourant la liste
*-------------------------------------------------------------------------
*/
void afficherListe(Message_t* liste)
{
   Message_t* cour = liste;
```

```
   while(cour != NULL)
   {
       afficherElement(cour);
       cour = cour->suivant;
   }
}

/*-------------------------------------------------------------------------
* afficherElement   Affiche un element
*
* Entrees: element, pointeur sur un structure message
*
* Affiche le contenu d'un element : sa date de debut, sa date de fin et le
*  texte associé.
*-------------------------------------------------------------------------
*/
void afficherElement(Message_t* element)
{
   printf("%d\t%d\t%s\n", element->dateDebut, element->dateFin, element->message);
}
```