IUT Paris Descartes 2017-2018

Structures de Données et Algorithmique

Mémoire de Projet

Maud GELLEE Groupe 109

Arsène LAPOSTOLET

Table des matières

I/ Présentation de l'application

Introduction :

Internet se fonde sur le transfert de paquets. C'est pourquoi quand on transmet des données en réseau, on divise ces données en paquets-réseau pour rendre leur transfert possible. A là réception des paquets, il est nécessaire de disposer d'un outil de messagerie pour réorganiser les données afin des les rendre intelligibles pour l'utilisateur.

Problématique :

Programmer une application de gestion de messagerie.

Rôle fonctionnel du projet :

L'objectif de ce projet est de programmer un logiciel dont la fonction est de lire des paquets-réseau dans un fichier texte. Il doit ensuite rassembler les données contenues dans ces paquets et les classer pour les rendre compréhensibles par l'utilisateur. Ces données sont également rangées dans des fichiers texte en fonction de leur destinataire par souci d'organisation. On rapporte les activités du logiciel dans un fichier texte appelé log afin de voir ce qui a été effectué. Le logiciel doit également gérer la perte de paquets. C'est-à-dire que si tous les paquets constituant un message n'ont pas été recueillis via le réseau, on supprime ce message.

Les entrées et sorties :

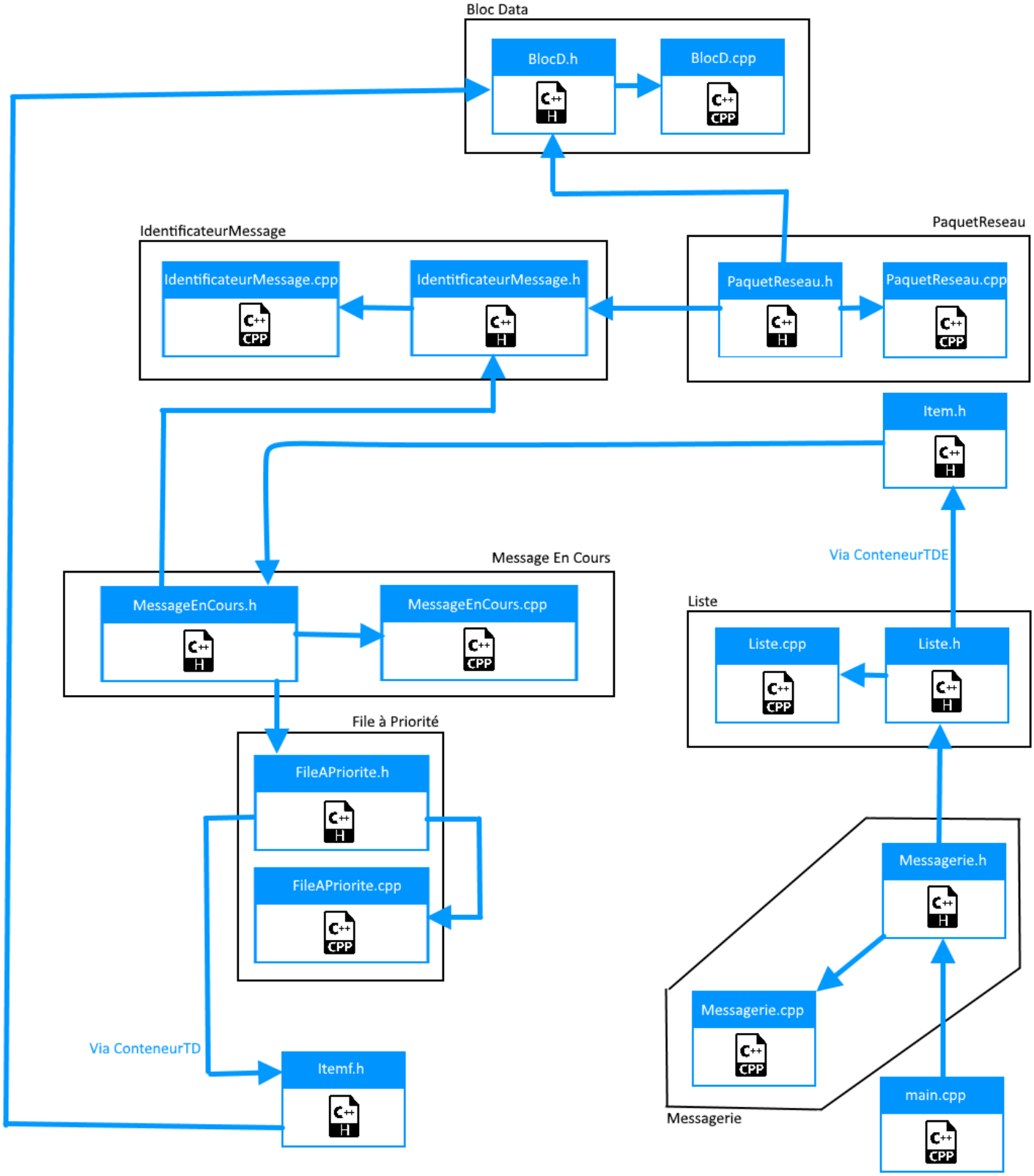
Les entrées du programme correspondent à un fichier texte qui contient les paquet-réseau constitués de leurs identifiants et des données qu'ils contiennent.

Le programme fait l'acquisition de ces informations par le biais d'un flot d'entrée de type *std::ifstream*. Les flots sont issus de la bibliothèque *<ifstream>*. On va ensuite ouvrir ce fichier grâce à la méthode *.open*. Nous avons donc spécifié et

codé des fonctions de lecture pour ranger les données ainsi lues dans les composants de notre application. Le prototype de ces fonctions est par exemple pour un Paquet Réseau : *void lire(std::istream& is, PaquetReseau& packRes);*.

Pour ce qui est des sorties nous avons, pour les mêmes composants, codés des fonctions d'affichage pour envoyer les contenus désirés dans un flot de sortie. Le prototype de ces fonctions est, par exemple, pour un Paquet Réseau : *void afficher (std::ostream& os, PaquetReseau& packRes);*. Nous ouvrons ensuite les flots de sortie dont nous avons besoin. *std::cout* pour la trace écran, *Mailbox* pour écrire dans les Mailbox, et *log* pour écrire dans le fichier log.

II/ Graphe de dépendance fonctionnelle



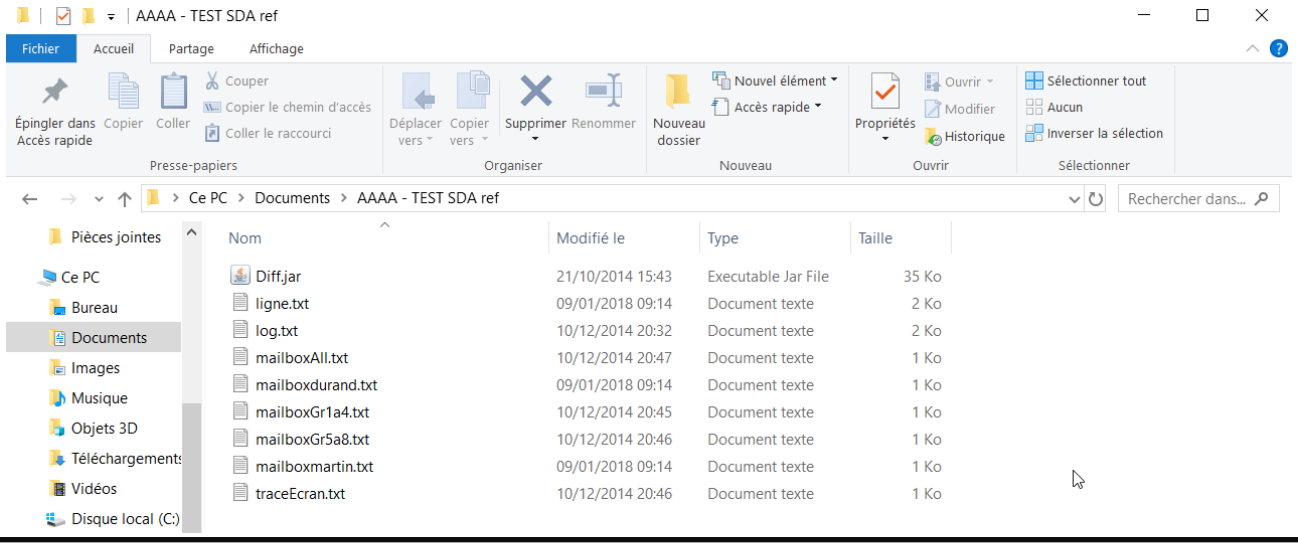
II/ Organisation des tests

Le projet a été mené sous forme de sprints définis par le génie logiciel. Un sprint est défini par une spécification, un jeu de données de test (inSp#) ainsi que des résultats attendus (OutSp#).

Un test permet de valider un objectif. Il utilise donc le jeu de données de test (JDT) et le résultat précis. Nous avons donc utilisé la redirection de flux d'entrée pour tester rapidement et efficacement nos programmes afin de valider nos sprints.

Nous avons ainsi créé un dossier contenant les fichiers requis :

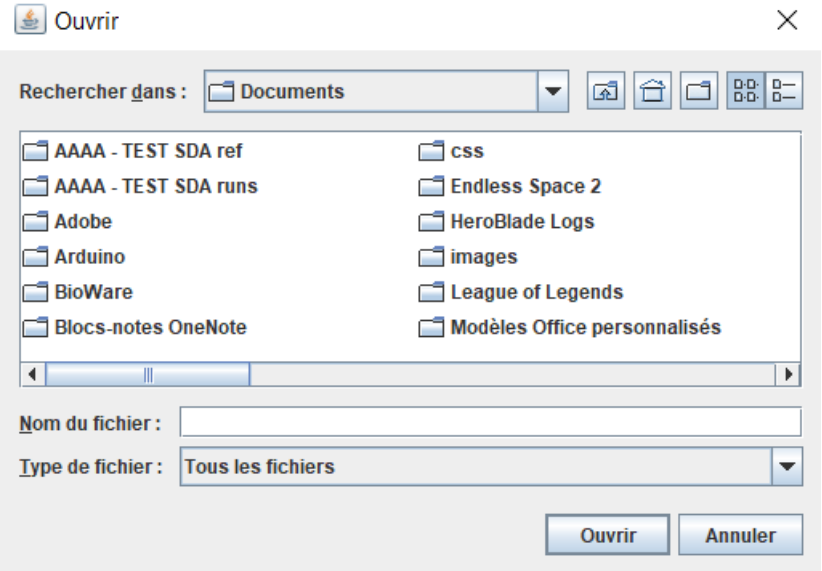
Utilisation de l'invite de commande pour la redirection :




```
Microsoft Windows [version 10.0.16299.19]
(c) 2017 Microsoft Corporation. Tous droits réservés.
C:\Users\arsen>cd Onedrive
C:\Users\arsen\OneDrive>cd Documents
C:\Users\arsen\OneDrive\Documents>cd "TEST_SDA"
C:\Users\arsen\OneDrive\Documents\TEST_SDA">"sprint 5.exe" <inSp5.txt> run.txt
C:\Users\arsen\OneDrive\Documents\TEST_SDA">
```

On crée ainsi un fichier run.txt qui contient les sorties de notre programme.

Il s'agit ensuite de comparer ce fichier de sorties run.txt avec le fichier des sorties attendues OutSp#.txt. Pour cela on utilise un outil fourni par les professeurs, diff.jar :



Ce programme nous montre ainsi les différences entre le fichier de sortie produit par notre programme et le fichier de sortie attendu. Si des différences apparaissent, il convient de revoir le programme, sinon, il est déclaré 0-défaut, et on passe à la suite.

III/ Bilan de validation des tests

Pour le bilan de la validation de tests, notre projet valide les six sprints.

Pour le premier sprint, il faut créer les composants BlocData, IdentificateurMessage *et PaquetReseau*. Il faut également coder leurs fonctions de lecture et d'affichage. Les fonctions de lecture utilisent la lecture mot à mot pour IdentificateurMessage et de la lecture mot à mot combinée à un appel de la méthode *.getline*. Pour PaquetReseau, on a un appel des fonctions de lecture de BlocData et IdentificateurMessage. Il est ensuite nécessaire dans le main de faire une boucle qui va lire le fichier pour stocker ses données dans les composants pour ensuite afficher les composants. Le sprint permet de tester si nos flots fonctionnent correctement. La comparaison des fichiers obtenus par redirection nous a montré que des espaces en début de ligne manquaient. Nous avons donc créé la fonction *NettoyerLigne*, donnée par notre professeur, qui agit come un *ws* en laissant l'espace en début de ligne. Nous sommes ensuite passés au sprint de niveau supérieur.

Pour le deuxième sprint, il faut organiser les données en mémoire. C'est pourquoi on utilise le composant Liste, reposant sur un conteneur en mémoire dynamique des composants tous deux fournis par l'équipe pédagogique. Il faut également créer les composants Messagerie, MessageEnCours et spécialiser le composant Item en MessageEnCours. Nous avons codé pour MessageEnCours des fonctions *initialiser* et *detruire* pour manipuler ce composant. Il faut également coder la fonction *estEgal* dans IdentificateurMessage qui permet de comparer deux IdentificateurMessage. Enfin, dans le composant de Messagerie, nous avons codé la fonction *TraiterPaquetReseau* qui permet de comparer les PaquetReseau et de les ranger dans la liste sous forme de MessageEnCours. Dans le main, on ajoute l'appel de *TraiterPaquetReseau* dans la boucle de lecture. On fait ensuite une boucle d'affichage car ce qui est demandé ici c'est les IdentificateursMessage. La comparaison des fichiers obtenus par redirection nous a permis de déclarer cesprint comme 0-défaut et de passer au sprint de niveau supérieur.

Pour le troisième sprint, il faut stocker les blocs de données en mémoire, pour pouvoir les ranger par identifiant. Nous avons donc implémenté le composant de file fourni par l'équipe pédagogique. Il faut aussi spécifier le type *Itemf* en *BlocD* ainsi que modifier la fonction *TraiterPaquetReseau* et la structure *MessageEnCours*. Nous avons envisagé deux cas : celui où l'identifiant est encore inconnu et celui où on a déjà reçu un paquet de ce message. Etant donné que nous ne devions pas encore trier les blocs de données en fonction de leur place dans le message, il nous a suffit de les implémenter en mémoire grâce à la structure *MessageEnCours*. En effet, la file se trouve dans cette dernière. La comparaison des fichiers obtenus par redirection nous a permis de déclarer ce sprint comme 0-défaut et de passer au sprint de niveau supérieur.

Le quatrième sprint n'a rien à voir avec les précédents. Il consiste à modifier la fonction *ecrire* du composants de file fourni par l'équipe pédagogique. Désormais la fonction *ecrire* devait se rapprocher de la fonction *inserer* d'un composant de liste. En effet, à chaque fois que la fonction *entrer* est appelée on parcourt toute la file afin de comparer le critère de l'*item* à insérer avec celui des *item* déjà existants. Pour cela on utilise la fonction *EnOrdre* qui compare des *int*. Ainsi, la file obtenu est une file à priorité, c'est-à-dire que les *item* qu'elle stocke sont stockés par ordre de priorité (ordre déterminé par le critère de la fonction *EnOrdre*). La comparaison des fichiers obtenus par redirection nous a permis de déclarer ce sprint comme 0-défaut et de passer au sprint de niveau supérieur.

Pour le cinquième sprint, il faut désormais que les blocs de données soient rangés en fonction de leur numéro de bloc. Nous utilisons le composant de file à priorité développé dans le quatrième sprint à la place du composant file. Nous choisissons le numéro de bloc comme critère dans la fonction *EnOrdre*. Ainsi nous obtenons les messages originaux. La comparaison des fichiers obtenus par redirection nous a permis de déclarer ce sprint comme 0-défaut et de

passer au sprint de niveau supérieur.

Pour le sixième sprint, il faut créer les fichiers de Mailbox et le fichier log. Pour chaque action telle que la réception d'un nouveau message, suppression d'un message, nous avons dû réutiliser les flots d'entrée-sortie pour créer un fichier log ainsi que plusieurs fichiers Mailbox qui correspondent à la messagerie de chaque destinataire. Nous avons donc modifié la fonction *TraiterPaquetReseau* à cet effet. De plus, nous devons désormais prendre en compte la perte de paquets réseau. On ne doit afficher le message que lorsque tous les blocs de données d'un message ont été reçus dans les temps. Nous nous sommes heurtés à trois difficultés majeures. La première concernait la suppression de paquets réseau. Au début, nous avions pensé que pour chaque paquet réseau reçu, nous devions vérifier si son numéro de paquet réseau était inférieur ou non à *10 + lastPRecu*. Cette technique fonctionnait parfaitement. Cependant, lorsque nous avons commencé à remplir le fichier log, nous nous sommes rendus compte que cette opération était effectuée trop tard par notre programme par rapport au log attendu. Nous avons donc, après comparaison, dû changer la fonction. Ainsi, à chaque paquet réseau, la fonction *TraiterPaquetReseau* vérifie si pour tous les messages en cours, le numéro du *lastPRecu* est inférieur ou non à 10 + le numéro du paquet actuel. Notre deuxième problème portait sur la création des Mailbox. En effet, nous avions deux manières de procéder : soit en créant un fichier Mailbox dès qu'un nouveau destinataire était repéré, soit en créant la Mailbox lorsque tout le message avait été reçu. La première solution nous permettait d'écrire correctement le titre de cette Mailbox, sans doublons (exemple : Mailbox all qui recevait deux messages différents). Mais nous devions alors supprimer les Mailbox, dans lesquelles des messages étaient perdus et se retrouvaient donc vides en fin de programme (exemple : Mailbox grp9a12). Or, nous ne sommes pas parvenu à supprimer un fichier, malgré nos essais avec la fonction *remove()*. Nous avons donc opté pour la deuxième solution, mais nous devions nous assurer que le titre n'était pas écrit deux fois en cas de réception de deux messages différents pour un même destinataire. Après quelques essais infructueux, nous sommes finalement parvenus grâce à la

fonction *getline()* à vérifier si nous avions déjà écrit le titre dans un fichier. Une fois ces deux problèmes résolus, notre sprint six était pratiquement 0-défaut, mis à part quelques problèmes d'espaces en trop pour les blocs de données, problème que nous n'avons pas réussi à résoudre.

IV/ Bilan du Projet

Ce projet collaboratif en programmation nous a permis de découvrir le langage C++, ses possibilités, mais aussi ses aspects difficiles à appréhender. En effet, ce projet a constitué pour nous une introduction aux mécaniques des flots d'entrée-sortie, mais aussi à la compilation séparée, qui permet une meilleure organisation et un repérage plus intuitif dans le projet, augmentant ainsi la productivité. Cela nous a aidé non seulement à parfaire notre maîtrise des outils de programmation impérative mais aussi à intégrer la rigueur de la documentation formatée, afin de faciliter l'implémentation de mise à jour ainsi que les modifications ultérieures. Tout au long de ce projet, nous avons encore une fois ressenti en nous l'âme du développeur. En effet, le fait de créer un tel logiciel fourni un sentiment d'accomplissement très agréable. Nous nous sommes sentis encore plus proche du programmeur qui fait tout son possible pour remplir son contrat. Chaque difficulté surmontée par la réflexion nous à motivé à avancer plus encore et à persévérer pour parachever notre ouvrage. Quand on développe ces projets, on se sent programmeur, et c'est vraiment une expérience unique.

Enfin, l'aspect collaboratif de ce projet fait partie de ses points forts. En effet, la motivation mutuelle est un moteur si puissant qu'il permet de surmonter tout type de désespoir. De plus, la puissance de deux esprits focalisés sur le même objectif, communicant sur un problème est une des formes les plus efficaces de réflexion. Chacun comblant les faiblesses de l'autre, nous avons pu nous tirer mutuellement vers le haut. Ayant déjà l'expérience du travail en commun, nous avons renforcé notre niveau de collaboration et cela nous a permis d'avancer encore plus vite malgré les difficultés, chacun connaissant les spécificités de l'autre. Encore une fois, cette expérience nous permet d'affirmer que la collaboration sur un tel projet est la forme la plus productive de travail.

Annexes

Annexe 1 : Jeu de données de test (inSp6.txt), sorties attendues (MailboxAll.txt, Mailboxgr1a4.txt, Mailboxgr5a8.txt, traceecran.txt, log.txt) et sorties ((MailboxAll.txt, Mailboxgr1a4.txt, Mailboxgr5a8.txt, traceecran.txt, log.txt) pour le sprint 6.

1 caraty gr9a12 26/11/17 14:16:37 0 1

Commentez vos source

2 caraty gr1a4 26/11/17 14:20:02 0 2

de commenter les fi

3 caraty gr5a8 26/11/17 14:35:23 0 1

Programmez sans nomb

4 caraty gr1a4 26/11/17 14:20:02 0 1

Il est indispensable

5 caraty gr5a8 26/11/17 14:35:23 1 4

ex. 255, ".txt").

6 caraty gr1a4 26/11/17 14:20:02 0 4

uls fichiers a consu

7 caraty gr1a4 26/11/17 14:20:02 0 3

chiers d'entete : se

8 caraty gr5a8 26/11/17 14:35:23 0 3

ntes litterales, par

9 caraty gr1a4 26/11/17 14:20:02 0 5

lter pour la reutili

10 caraty all 26/11/17 15:27:57 0 1

Bon courage a tous..

11 caraty gr1a4 26/11/17 14:20:02 1 6

sation.

12 caraty all 26/11/17 15:25:41 1 1

Bon travail...

13 caraty gr5a8 26/11/17 14:35:23 0 2

res magiques (consta

14 caraty gr9a12 26/11/17 14:16:37 1 2

s.

15 caraty all 26/11/17 15:27:57 1 2

. MJC

Mailbox all

caraty all 26/11/14 15:25:41

Bon travail...

caraty all 26/11/14 15:27:57

Bon courage a tous... MJC

Mailbox gr1a4

caraty gr1a4 26/11/14 14:20:02

Il est indispensable de commenter les fichiers d'entete : seuls fichiers a consulter pour la reutilisation.

Mailbox gr5a8

caraty gr5a8 26/11/14 14:35:23

Programmez sans nombres magiques (constantes litterales, par ex. 255, ".txt").

caraty gr1a4 26/11/14 14:20:02

Il est indispensable de commenter les fichiers d'entete : seuls fichiers a consulter pour la reutilisation.

caraty all 26/11/14 15:25:41

Bon travail...

caraty gr5a8 26/11/14 14:35:23

Programmez sans nombres magiques (constantes litterales, par ex. 255, ".txt").

caraty all 26/11/14 15:27:57

Bon courage a tous... MJC

1 Detection de nouveau message caraty gr9a12 26/11/14 14:16:37 0 1

2 Detection de nouveau message caraty gr1a4 26/11/14 14:20:02 0 2

3 Detection de nouveau message caraty gr5a8 26/11/14 14:35:23 0 1

5 Detection de fin de message caraty gr5a8 26/11/14 14:35:23 1 4

10 Detection de nouveau message caraty all 26/11/14 15:27:57 0 1

11 Detection de fin de message caraty gr1a4 26/11/14 14:20:02 1 6

11 Archivage mailbox gr1a4 26/11/14 14:20:02

11 Perte de paquet, suppression de message caraty gr9a12 26/11/14 14:16:37

12 Detection de nouveau message caraty all 26/11/14 15:25:41 1 1

12 Detection de fin de message caraty all 26/11/14 15:25:41 1 1

12 Archivage mailbox all 26/11/14 15:25:41

13 Archivage mailbox gr5a8 26/11/14 14:35:23

14 Detection de nouveau message caraty gr9a12 26/11/14 14:16:37 1 2

14 Detection de fin de message caraty gr9a12 26/11/14 14:16:37 1 2

15 Detection de fin de message caraty all 26/11/14 15:27:57 1 2

15 Archivage mailbox all 26/11/14 15:27:57

Fin de transmission, suppression de message caraty gr9a12 26/11/14 14:16:37

Mailbox all

caraty all 26/11/17 15:25:41

Bon travail...

caraty all 26/11/17 15:27:57

Bon courage a tous... MJC

Mailbox gr1a4

caraty gr1a4 26/11/17 14:20:02

Il est indispensable de commenter les fichiers d'entete : seuls fichiers a consulter pour la reutilisation.

Mailbox gr5a8

caraty gr5a8 26/11/17 14:35:23

Programmez sans nombres magiques (constantes litterales, par ex. 255, ".txt").

caraty gr1a4 26/11/17 14:20:02

Il est indispensable de commenter les fichiers d'entete : seuls fichiers a consulter pour la reutilisation.

caraty all 26/11/17 15:25:41

Bon travail...

caraty gr5a8 26/11/17 14:35:23

Programmez sans nombres magiques (constantes litterales, par ex. 255, ".txt").

caraty all 26/11/17 15:27:57

Bon courage a tous... MJC

1 Detection de nouveau message caraty gr9a12 26/11/17 14:16:37 0 1

2 Detection de nouveau message caraty gr1a4 26/11/17 14:20:02 0 2

3 Detection de nouveau message caraty gr5a8 26/11/17 14:35:23 0 1

5 Detection de fin de message caraty gr5a8 26/11/17 14:35:23 1 4

10 Detection de nouveau message caraty all 26/11/17 15:27:57 0 1

11 Detection de fin de message caraty gr1a4 26/11/17 14:20:02 1 6

11 Archivage mailbox gr1a4 26/11/17 14:20:02

11 Perte de paquet, suppression de message caraty gr9a12 26/11/17 14:16:37

12 Detection de nouveau message caraty all 26/11/17 15:25:41 1 1

12 Detection de fin de message caraty all 26/11/17 15:25:41 1 1

12 Archivage mailbox all 26/11/17 15:25:41

13 Archivage mailbox gr5a8 26/11/17 14:35:23

14 Detection de nouveau message caraty gr9a12 26/11/17 14:16:37 1 2

14 Detection de fin de message caraty gr9a12 26/11/17 14:16:37 1 2

15 Detection de fin de message caraty all 26/11/17 15:27:57 1 2

15 Archivage mailbox all 26/11/17 15:27:57

Fin de transmission, suppression de message caraty gr9a12 26/11/17 14:16:37

Annexe 2 : Code complet

Main.cpp

```
1   /**
2   *@TestLecture.cpp
3   *Projet SDA
4   *@author Maud Gellée
5   *@author Arsène Lapostolet
6   *@version 1 15/12/2017
7   *@brief lit dans le fichier et affiche le contenu à l'écran
8   */
9
10  #include "PaquetReseau.h"
11  #include "Messagerie.h"
12  #include <iostream>
13  #include <fstream>
14
15  int main() {
16      PaquetReseau p;
17      Messagerie m;
18      initialiser(m, 1, 1);
19      std::ofstream log("log.txt", std::ios::app);
20      std::ifstream flotE;
21      flotE.open("inSp6.txt", std::ios::in);
22      if (flotE.fail()) {
23          std::cerr << "Impossible d'ouvrir le fichier" << std::endl;
24          exit(1);
25      }
26      else {
27          do {
28              lire(flotE, p);
29              TraiterPaquetReseau(m, p);
30          } while (flotE.good());
31
32          log << "Fin de transmission";
33          for (unsigned int j = 0; j < longueur(m.liste); ++j) {
34              if (!estVide(m.liste.c.tab[j].fileB)) {
35                  log << ", suppression de message ";
36                  afficher(log,m.liste.c.tab[j].IdMes);
37                  log << std::endl;
38                  supprimer(m.liste, j);
39              }
40          }
41      }
42      log.close();
43      flotE.close();
44      detruire(m.liste);
45
46      system("pause"); return 0;
47  }
48
```

BlocData.h

```
1   #pragma once
2   #ifndef _BLOCDATA_
3   #define _BLOCDATA_
4
5   /**
6   *@file BlocData.h
7   *Projet SDA
8   *@author Maud Gellée
9   *@author Arsène Lapostolet
10  *@version 1 14/12/2017
11  *@brief fichier d'entête des blocs de données (contenu des messages)
12  */
13
14  #include <fstream>
15
16  /*Structure Bloc de donnée (contenu du message)*/
17  struct BlocData {
18      enum { LGMSG = 21 };
19      unsigned int noBloc;
20      char data[LGMSG];
21  };
22
23  /**
24  *@brief lis un Bloc de donnée de message dans le fichier
25  *@param [in-out] flux d'entrée
26  *@param [in-out] bloc de donnée
27  */
28  void lire(std::istream& is, BlocData& blocD);
29
30  /**
31  *@brief envoie un bloc de donnée de message dans un flux de sortie
32  *@param [in-out] flux de sortie
33  *@param [[in] bloc de donnée à envoyer
34  */
35  void afficher(std::ostream& os, BlocData& blocD);
36
37  /**
38  *@brief permet de nettoyer la ligne d'un flux d'entrée pour finir une ligne sa
39  *ns supprimer les espaces
40  *@param [in-out] flux d'entrée
41  */
42  void nettoyerLigne(std::istream& is);
43
44  /**
45  * @brief relation d'ordre entre 2 positions
46  * @param[in] b1 : le 1er bloc de données
47  * @param[in] b2 : le 2ème bloc de données
48  * @return true si b1 et b2 sont ordonnés, false sinon
49  */
50  bool enOrdre(const BlocData& b1, const BlocData& b2);
51
52  #endif
```

BlocData.cpp

```cpp
/**
*@file BlocData.cpp
*Projet SDA
*@author Maud Gellée
*@author Arsène Lapostolet
*@version 1 14/12/2017
*@brief fichier de définition des blocs de données (contenu des messages)
*/

#include <iostream>
#include <fstream>
#include "BlocData.h"

/**
*@brief lis un Bloc de donnée de message dans le fichier
*@param [in-out] flux d'entrée
*@param [in-out] bloc de donnée
*/
void lire(std::istream& is, BlocData& blocD) {
    is >> blocD.noBloc;
    nettoyerLigne(is);
    is.getline(blocD.data, BlocData::LGMSG);
}

/**
*@brief envoie un bloc de donnée de message dans un flux de sortie
*@param [in-out] flux de sortie
*@param [[in] bloc de donnée à envoyer
*/
void afficher(std::ostream& os, BlocData& blocD) {
    os << blocD.noBloc << std::endl;
    os << blocD.data << std::endl;
}

/**
*@brief permet de nettoyer la ligne d'un flux d'entrée pour finir une ligne sa
*ns supprimer les espaces
*@param [in-out] flux d'entrée
*/
void nettoyerLigne(std::istream& is) {
    is.ignore(std::numeric_limits <std::streamsize> ::max(), '\n');
}

/**
* @brief relation d'ordre entre 2 positions
* @param[in] b1 : le 1er bloc de données
* @param[in] b2 : le 2ème bloc de données
* @return true si b1 et b2 sont ordonnés, false sinon
```

```
49      */
50      bool enOrdre(const BlocData& b1, const BlocData& b2) {
51          if (b1.noBloc < b2.noBloc) {
52              return true;
53          }
54          else {
55              return false;
56          }
57      }
58
```

IdentificateurMessage.h

```
1       #pragma once
2       #ifndef _IDENTIFICATEURMESSAGE_
3       #define _IDENTIFICATEURMESSAGE_
4
5       /**
6       *@file IdentificateurMessage.h
7       *Projet SDA
8       *@author Maud Gellée
9       *@author Arsène Lapostolet
10      *@version 1 14/12/2017
11      *@brief fichier d'entête des identificateurs des messages
12      */
13
14      #include <fstream>
15      #include <iostream>
16
17      /* Strcture de l'identificateur du message */
18      struct IdMessage {
19          enum { MAXNOM = 80 };
20          enum { MAXDH = 10 };
21          char exp[MAXNOM + 1];
22          char dest[MAXNOM + 1];
23          char date[MAXDH];
24          char heure[MAXDH];
25      };
26
27      /**
28      * @brief affiche l'identifiant d'un message
29      * @param [in-out] flux de sortie du programme
30      * @param [in] identifiant du message
31      */
32      void afficher(std::ostream& os, IdMessage& idMes);
33
34      /**
35      * @brief récupère l'identifiant du message dans un fichier
36      * @param [in] flux entrée d'un fichier text
37      */
38      void lire(std::istream& is, IdMessage& idMes);
39
```

```
40      /**
41       * @brief compare deux Identificateurs messages
42       * @param [in] premier identifiant à comparer
43       * @param [in] deuxième idétifiant à comparer
44       * @return renvoie un booléen, 0 s'ils sont pareil, 1 sinon
45       */
46      bool estEgal(const IdMessage& id1, const IdMessage& id2);
47
48      #endif
```

IdentificateurMessage.cpp

```
1       /**
2       *@IdentificateurMessage.cpp
3       *Projet SDA
4       *@author Maud Gellée
5       *@author Arsène Lapostolet
6       *@version 1 14/12/2017
7       *@brief fichier de définition des identificateurs de messages
8       */
9
10      #include <iostream>
11      #include <fstream>
12      #include "IdentificateurMessage.h"
13
14      /**
15      * @brief affiche l'identifiant d'un message
16      * @param [in-out] flux de sortie du programme
17      * @param [in] identifiant du message
18      */
19      void lire(std::istream& is, IdMessage& idMes) {
20          is >> idMes.exp >> idMes.dest >> idMes.date >> idMes.heure;
21      }
22
23      /**
24      * @brief récupère l'identifiant du message dans un fichier
25      * @param [in-out] flux entrée d'un fichier text
26      * @param [in-out] identifiant du message
27       */
28      void afficher(std::ostream& os, IdMessage& idMes) {
29          os << idMes.exp << " " << idMes.dest << " " << idMes.date << " "
30              << idMes.heure/*<<std::endl*/;
31      }
32
33      /**
34       * @brief compare deux Identificateurs messages
35       * @param [in] premier identifiant à comparer
36       * @param [in] deuxième idétifiant à comparer
37       * @return renvoie un booléen, 0 s'ils sont pareil, 1 sinon
38       */
39      bool estEgal(const IdMessage& id1, const IdMessage& id2) {
40          if ((strcmp(id1.dest, id2.dest) == 0) &&
```

```
41          (strcmp(id1.exp, id2.exp) == 0) &&
42          (strcmp(id1.date, id2.date) == 0) &&
43          (strcmp(id1.heure, id2.heure) == 0)) {
44          return true;
45      }
46      else {
47          return false;
48      }
49  }
```

Itemf.h

```
1   #pragma once
2   #ifndef _ITEMF_
3   #define _ITEMF_
4
5   /**
6   * @file Itemf.h
7   * @author Maud Gellée
8   * @author Arsène Lapostolet
9   * @version 1 20/12/2017
10  * @brief Spécialisation du type Item
11  */
12
13  #include "PaquetReseau.h"
14
15  typedef BlocData Itemf;
16
17  #endif
```

Item.h

```
1   #pragma once
2   #ifndef _ITEM_
3   #define _ITEM_
4
5   /**
6   * @file Item.h
7   * Projet sem04-tp-Cpp3
8   * @author l'équipe pédagogique
9   * @version 1 23/12/05
10  * @brief Spécialisation du type Item
11  * Structures de données et algorithmes - DUT1 Paris 5
12  */
13
14  #include "MessageEnCours.h"
15
16  typedef MessageEnCours Item;
17
18  #endif
```

PaquetReseau.h

```
1   #pragma once
2   /**
3   *@file PaquetReseau.h
4   *Projet SDA
5   *@author Maud Gellée
6   *@author Arsène Lapostolet
7   *@version 1 14/12/2017
8   *@brief fichier d'entête des paquets réseau
9   */
10
11  #pragma once
12  #ifndef _PAQUETRESEAU_
13  #define _PAQUETRESEAU_
14
15  #include "IdentificateurMessage.h"
16  #include "BlocData.h"
17  #include <fstream>
18
19  /*Structure du paquet-réseau*/
20  struct PaquetReseau {
21      unsigned int noPR;
22      IdMessage IdMes;
23      BlocData blocD;
24      unsigned int finMess;
25  };
26
27  /**
28  * @brief récupère un paquet réseau
29  * @param [in-out] flux entrée d'un fichier text
30  * @param [in-out] paquet réseau
31  */
32  void lire(std::istream& is, PaquetReseau& packRes);
33
34  /**
35  * @brief affiche un paquet réseau
36  * @param [in-out] flux de sortie
37  * @param [in] paquet réseau
38  */
39  void afficher(std::ostream& os, PaquetReseau& packRes);
40
41  #endif
```

PaquetReseau.cpp

```cpp
/**
*@file PaquetReseau.cpp
*Projet SDA
*@author Maud Gellée
*@author Arsène Lapostolet
*@version 1 14/12/2017
*@brief fichier de définition des paquets réseau
*/

#include <iostream>
#include <fstream>
#include "PaquetReseau.h"
#include "BlocData.h"
#include "IdentificateurMessage.h"

/**
* @brief récupère un paquet réseau
* @param [in-out] flux entrée d'un fichier text
* @param [in-out] paquet réseau
*/
void lire(std::istream& is, PaquetReseau& packRes) {
    is >> packRes.noPR;
    lire(is, packRes.IdMes);
    is >> packRes.finMess;
    lire(is, packRes.blocD);
}
}

/**
* @brief affiche un paquet réseau
* @param [in-out] flux de sortie
* @param [in] paquet réseau
*/
void afficher(std::ostream& os, PaquetReseau& packRes) {
    os << packRes.noPR;
    os << " ";
    afficher(os, packRes.IdMes);
    os << " ";
    os << packRes.finMess;
    os << " ";
    afficher(os, packRes.blocD);
}

```

Messagerie.h

```
1   #pragma once
2   /**
3    * @file Messagerie.h
4    * @author Maud Gellée
5    * @author Arsène Lapostolet
6    * @version 1 - 19/12/2017
7    * @brief Composant de stockage des messages en cours
8    */
9
10  #pragma once
11  #ifndef _MESSAGERIE_
12  #define _MESSAGERIE_
13
14  #include "Liste.h"
15  #include "PaquetReseau.h"
16
17  struct Messagerie {
18      Liste liste;
19  };
20
21  /**
22   * @brief traite un paquet réseau
23   * @param [in-out] messagerie concernée
24   * @param [in-out] paquet réseau à traiter
25   */
26  void TraiterPaquetReseau(Messagerie& Mes, PaquetReseau& packRes);
27
28  /**
29   * @brief Initialise une messagerie
30   * @param [in-out] la messagerie à initiliser
31   * @param [in] la capacité de la liste de cette messagerie
32   * @param [in] pas d'extension de liste de cette messagerie
33   */
34  void initialiser(Messagerie& m, unsigned int capa, unsigned int pas);
35
36  /**
37   * @brief désalloue une messagerie
38   * @param [in-out] la messagerie à désallouer
39   */
40  void detruire(Messagerie& m);
41
42  #endif
```

Messagerie.cpp

```cpp
/**
* @file Messagerie.cpp
* @author Maud Gellée
* @author Arsène Lapostolet
* @version 1 - 19/12/2017
* @brief Composant de stockage des messages en cours
*/

#include "Messagerie.h"
#include "PaquetReseau.h"
#include <string>
#include <fstream>

/**
* @brief traite un paquet réseau
* @param [in-out] messagerie concernée
* @param [in-out] paquet réseau à traiter
*/
void TraiterPaquetReseau(Messagerie& Mes, PaquetReseau& packRes) {
    unsigned int i;
    std::ofstream log("log.txt", std::ios::app);
    for (i = 0; i < longueur(Mes.liste); ++i) {
        if (estEgal(Mes.liste.c.tab[i].IdMes, packRes.IdMes) == true) {
            break;
        }
    }
    if (i == longueur(Mes.liste)) {


        MessageEnCours m;
        initialiser(m);
        m.IdMes = packRes.IdMes;
        m.nbPRecus = 1;
        m.LastPrecu = packRes.noPR;
        inserer(Mes.liste, i, m);
        entrer(Mes.liste.c.tab[i].fileB, packRes.blocD);
        log << packRes.noPR << " Detection de nouveau message ";

        /*std::string grp = packRes.IdMes.dest;
        std::ofstream Mailbox("Mailbox" + grp + ".txt", std::ios::out);
        Mailbox << "Mailbox " << grp << std::endl;*/
        afficher(log, packRes.IdMes);
        log << " " << packRes.finMess << " " << packRes.blocD.noBloc
            << std::endl;
        if (packRes.finMess == 1) {
            Mes.liste.c.tab[i].lgMes = packRes.blocD.noBloc;
            log << packRes.noPR << " Detection de fin de message ";
            afficher(log, packRes.IdMes);
            log << " " << packRes.finMess << " " << packRes.blocD.noBloc
```

```
50                  << std::endl;
51              }
52          }
53          else {
54              entrer(Mes.liste.c.tab[i].fileB, packRes.blocD);
55              (Mes.liste.c.tab[i].nbPRecus)++;
56              Mes.liste.c.tab[i].LastPrecu = packRes.noPR;
57              if (packRes.finMess == 1) {
58                  Mes.liste.c.tab[i].lgMes = packRes.blocD.noBloc;
59                  log << packRes.noPR << " Detection de fin de message ";
60                  afficher(log, packRes.IdMes);
61                  log << " " << packRes.finMess << " " << packRes.blocD.noBloc
62                      << std::endl;
63              }
64          }
65          if ((lire(Mes.liste, i).lgMes == lire(Mes.liste, i).nbPRecus)) {
66
67              log << packRes.noPR << " Archivage mailbox ";
68              log << packRes.IdMes.dest << " " << packRes.IdMes.date << " " <<
69                  packRes.IdMes.heure << std::endl;
70
71              std::string grp= packRes.IdMes.dest;
72              char read[8];
73              char mail [8] = "Mailbox";
74              //std::cout << mail;
75              std::ofstream Mailbox ("Mailbox" + grp + ".txt", std::ios::app);
78              Mailboxin.getline(read, 8);
79              std::cout << read;
80              if (strcmp(read, mail)!=0) {
81                  Mailbox << "Mailbox " << grp << std::endl;
82              }
83              afficher(std::cout, Mes.liste.c.tab[i].IdMes);
84              afficher (Mailbox, Mes.liste.c.tab[i].IdMes);
85              std::cout << std::endl;
86              Mailbox << std::endl;
87              for (unsigned int j = 0; j < Mes.liste.c.tab[i].lgMes; ++j) {
88                  std::cout << tete(lire(Mes.liste, i).fileB).data;
89                  Mailbox << tete(lire(Mes.liste, i).fileB).data;
90                  sortir(Mes.liste.c.tab[i].fileB);
91                  Mes.liste.c.tab[i].nbPRecus--;
92              }
93              std::cout << std::endl << std::endl;
94              Mailbox << std::endl;
95              Mailbox.close();
96          }
97
98          for (unsigned int j = 0; j < longueur(Mes.liste); ++j) {
99              if (packRes.noPR >= lire(Mes.liste, j).LastPrecu + 10) {
100                 //mettre à jour log
101                 log << packRes.noPR << " Perte de paquet, suppression de message ";
```

```
102                 afficher(log, Mes.liste.c.tab[j].IdMes);
103                 log << std::endl;
104                 supprimer(Mes.liste, j);
105             }
106         }
107
108     }
109
110     /**
111     * @brief Initialise une messagerie
112     * @param [in-out] la messagerie à initiliser
113     * @param [in] la capacité de la liste de cette messagerie
114     * @param [in] pas d'extension de liste de cette messagerie
115     */
116     void initialiser(Messagerie& m, unsigned int capa, unsigned int pas) {
117         initialiser(m.liste, capa, pas);
118     }
119
120     /**
121     * @brief désalloue une messagerie
122     * @param [in-out] la messagerie à désallouer
123     */
124     void detruire(Messagerie& m) {
125         detruire(m.liste);
126     }
```

FileAPriorite.cpp (fonction entrer)

```
67  /**
68   * @brief Entrer un item dans la file
69   * @param[in,out] f : la file
70   * @param[in] it : l'item à entrer
71   * @pre f n'est pas pleine
72   */
73  void entrer(File& f, const Itemf& it) {
74      assert(!estPleine(f));
75      unsigned int j;
76      /*if (f.nb == 0) {
77      ecrire(f.c, 0, it); //@see ecrire de ConteneurTD
78      }
79      else {*/
80      for (j = 0; j < f.nb; ++j) {
81          if (enOrdre(it, f.c.tab[j])) {
82              //Item item;
83              for (unsigned int i = f.nb; i > j; --i) {
84                  ecrire(f.c, i, lire(f.c, i - 1));
85              }
86              ecrire(f.c, j, it);
87              break;
88          }
89      }
90      if (f.nb == j) {
91          ecrire(f.c, j, it);
92      }
93      f.indProchain = (f.indProchain + 1) % f.c.capacite;
94      f.nb++;
95  }
```

MessageEnCours.h

```
1   #pragma once
2   #ifndef _MESSAGEENCOURS_
3   #define _MESSAGEENCOURS_
4
5   /**
6   * @file MessageEnCours.h
7   * Projet SDA
8   * @author Maud Gellée
9   * @author Arsène Lapostolet
10  * @version 1 16/12/2017
11  * @brief fichier d'entête des messages en cours
12  */
13
14  #include "IdentificateurMessage.h"
15  #include "FileAPriorite.h"
16
17  struct MessageEnCours {
18      IdMessage IdMes;
19      File fileB;
20      unsigned int lgMes;
21      unsigned int nbPRecus;
22      unsigned int LastPrecu;
23  };
24
25  /**
26  * @brief initialiser un message en cours
27  * @param le Message en cours à initialiser
28  */
29  void initialiser(MessageEnCours& MesEnC/*, const PaquetReseau& p*/);
30
31  /**
32  * @brief détruire un message en cours
33  * @param le Message en cours à détruire
34  */
35  void detruire(MessageEnCours& MesEnC);
36
37
38  #endif
```

MessageEnCours.cpp

```cpp
/**
*@file MessageEnCours.cpp
*Projet SDA
*@author Maud Gellée
*@author Arsène Lapostolet
*@version 1 16/12/2017
*@brief fichier de définition des messages en cours
*/

#include "MessageEnCours.h"
#include "PaquetReseau.h"

/**
*@brief initialiser un message en cours
*@param le Message en cours à initialiser
*/
void initialiser(MessageEnCours& MesEnC/*, const PaquetReseau& p*/) {
    MesEnC.lgMes = 0;
    MesEnC.nbPRecus = 0;
    MesEnC.LastPrecu = 0;
    enum { MAXFILE = 50 };
    initialiser(MesEnC.fileB, MAXFILE);
}

/**
*@brief détruire un message en cours
*@param le Message en cours à détruire
*/
void detruire(MessageEnCours& MesEnC) {
    detruire(MesEnC.fileB);
}
```