



Structures de Données et Algorithmique

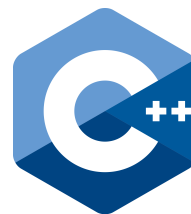
Mémoire de Projet





Page de Garde.....	1
Table des matières.....	2
Présentation de l'application.....	3
Graphique de dépendances fonctionnelles.....	5
Organisation des tests.....	6
Bilan de validation des tests.....	8
Bilan de Projet.....	12
Annexe 1 : Jeux de test et sorties.....	14
Annexe 2 : Sources Sprint 6 Validé.....	19

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 la 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.

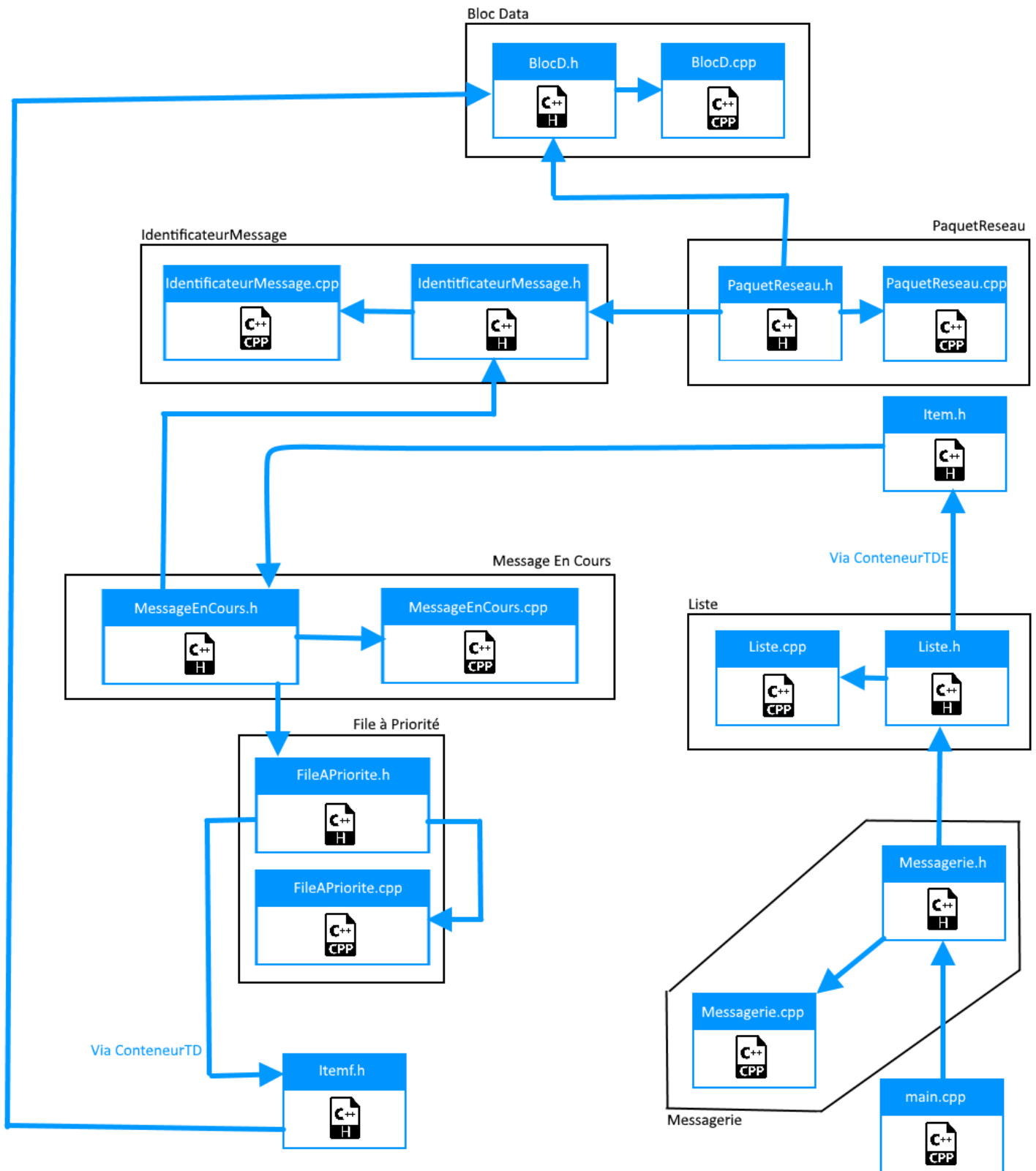
Le programme fait l'acquisition de ces informations par le biais d'un flot d'entrée de type *std::ifstream*. Les flots sont issus de la bibliothèque *<ifstream>*. On va ensuite ouvrir ce fichier grâce à la méthode *.open*. Nous avons donc spécifié et



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

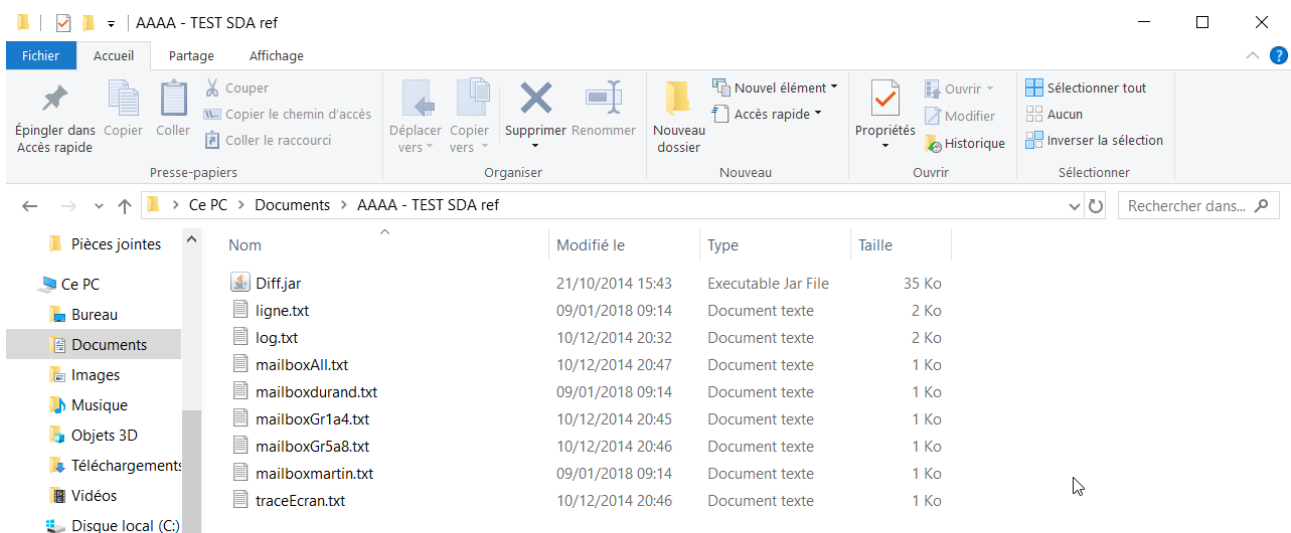


Le projet a été mené sous forme de sprints définis par le génie logiciel. Un sprint est défini par une spécification, un jeu de données de test (inSp#) ainsi que des résultats attendus (OutSp#).

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"
```

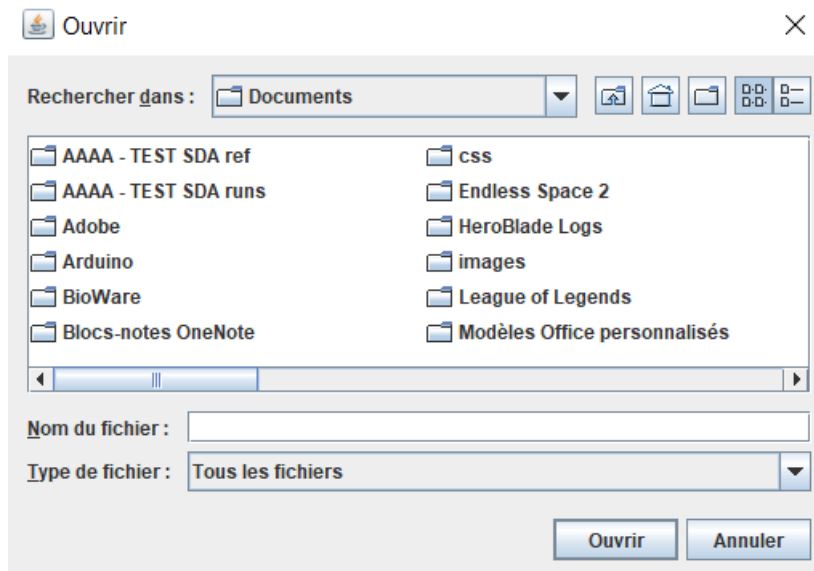
```
C:\Users\arsen\OneDrive\Documents\TEST_SDA">"sprint 5.exe" <inSp5.txt>  
run.txt
```

```
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 comme 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 `destruire` 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 ce sprint 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 devons pas encore trier les blocs de données en fonction de leur place dans le message, il nous a suffi 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 *insérer* 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



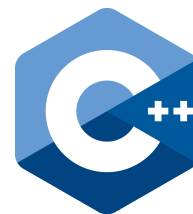
passer au sprint de niveau supérieur.

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 avons 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 avons 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 devons 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 avons 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 a 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, communiquant 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.



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

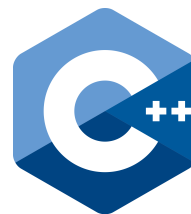
Bon travail...

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