INF-145 Travail pratique #2
École de technologie supérieure
Découpage. Regroupement. Reconstruction..
Travail en équipe Remise : semaine 9

**Objectifs du travail :**

- Construire une simulation numérique qui va utiliser des tableaux utilisés sous diverses formes dans des objets de nouveaux types aux comportements bien définis.
- Utiliser des types structurés capables de réduire la difficulté d'un problème global en sous-problèmes locaux traités par l'introduction de fonctions dans des modules spécialisés. La factorisation va amener une résolution individuelle aux sous-problèmes rencontrés et une bien plus grande facilité de tests et de débogage de tous les éléments du projet.
- Respecter les exigences de programmation qui assurent la qualité du logiciel : modularité, factorisation, présentation et homogénéité dans le code et dans ses commentaires.

**Description du problème :**
DÉCOUPAGE, REGROUPEMENT & RECONSTRUCTION d'ensembles de blocs épars d'information

L'information d'un fichier quelconque, dans un long flot (stream) d'octets, doit fréquemment être traitée et transportée par blocs. Prenons les cas certains des encryptions/ transports/ décryptions des blocs d'informations sensibles, du transport et du traitement par blocs des signaux audio-vidéo par les grands médias ou plus simplement du transport de l'information sur le Web.

Tout le logiciel se résume à l'utilisation de trois modules de service contenant la définition de types structurés. Le module du découpage-fichiers en blocs vous est fourni et vous ne devez pas le modifier. Les modules de regroupement et de reconstruction traitent des ensembles de blocs et sont à faire selon les descriptions données. Votre équipe y ajoutera son fichier principal qui réalise la simulation et contient entre autres la fonction « main ».

### 3 Le module de découpage

Le module de découpage va, à partir de fichiers donnés, produire des blocs d'octets identifiables individuellement. Les blocs seront de tailles diverses obtenues aléatoirement dans un intervalle bien défini. Le module offre en interface (fichier *.H) le type *t_block* et toute une gamme de fonctions publiques (fonctions informatrices et mutatrices).

Voici ce type extrait de l'interface du module :

```
/*=========================================================================*/
//  Le type t_block est publique,
//  il donne accès à un élément de l'ensemble des blocs obtenus
//  en découpant un fichier en blocs de bytes de différentes tailles.
typedef struct {

  unsigned int f_identifiant; //identifiant unique du fichier

  unsigned int num_bloc;      //numéro de ce bloc dans l'énumération des blocs

  unsigned int taille_bloc;   //nombre d'octets extraits du fichier dans ce bloc

  unsigned char * buffer;     //l'adresse du tableau dynamique contenant le bloc
```

```
  unsigned char bloc_final;   //1 si c'est le dernier bloc du découpage, 0 sinon

} t_block;
/*=========================================================================*/
```

---

**Mandat #1** -
**lire avec attention** l'interface (fichier *.H) du module de découpage, **compléter** les fonctions *print_bloc* et *print_etat_fichier* et **tester** toutes les fonctions du module jusqu'à une compréhension complète des fonctions offertes.

Vous devez écrire une première fonction de test dans votre fichier principal qui initialise le découpage puis donne un fichier à découper. La fonction obtient un à la fois tous les blocs émis. Périodiquement vous affichez le contenu d'un bloc obtenu et les informations sur l'état de découpage du fichier. Vous affichez le dernier bloc reçu et les informations finales du fichier avant de retirer ce fichier du module de découpage.

Faites une seconde fonction de test, elle est semblable à la première mais va donner deux ou trois fichiers au module de découpage puis va récupérer et séparer dans des tableaux distincts tous les blocs émis (chaque fichier aura son propre tableau de *t_block*). Périodiquement vous devez afficher le contenu d'un bloc obtenu et les informations sur l'état du découpage de chacun des fichiers. ATTENTION chacun des blocs finaux doit être affiché à la réception.

Testez le module avec des fichiers de tailles moyennes mais bien distinctes comme des images au format JPG, le logiciel n'est pas fait pour traiter de micro-fichiers texte et attention dans le second test à ne pas débordez vos tableaux.

**Ces fonctions de tests devront être remises avec votre travail.**

---

**Description de la conception modulaire demandée**
La conception complète se base sur trois modules de types structurés :

- Le module de **découpage** qui est déjà offert et que vous comprenez bien.
- Le module de **regroupement** des blocs d'un même fichier offre un type qui permet de séparer tous les blocs obtenus selon leur fichier associé et de pouvoir conserver un nombre maximum de fichiers défini à priori par une constante dans ce module.
- Le module de **reconstruction** d'un fichier dont le type permet de conserver en ordre les blocs d'un fichier jusqu'à les contenir tous et de pouvoir alors déclencher le processus de reconstruction d'un fichier-copie du fichier original.

On vous demande de développer les deux modules dans l'ordre de leur présentation et de tester aussitôt un module et l'ensemble de ses fonctions associées. Une fois le module bien testé, il pourra servir dans toute la simulation numérique. **Vous aurez à remettre ces deux tests.**

**Description du module « m_regroupement »**
Le module *m_regroupement* définit le type *t_regroupement* dont une variable sera capable de conserver un nombre limité d'objets *t_block* d'un unique fichier juste après le découpage.

Une variable de type *t_regroupement* contient ces membres :

- L'identifiant unique (*unsigned int*) d'un fichier
- Un pointeur donnant accès à un tableau dynamique de *t_block*
- La taille du tableau précédent
- Le nombre de *t_block* actuellement présents

Un *t_regroupement* fonctionne à la manière d'une pile, ce qui reste particulièrement facile à implémenter dans un tableau, tout le comportement de la pile de *t_block* dépend uniquement du nombre actuel de *t_block* dans la pile :

- On ne peut empiler un *t_block* que s'il a le bon identifiant-fichier.
- On peut empiler seulement si le nombre actuel de *t_block* est inférieur à la taille du tableau.
- On peut dépiler seulement si le nombre actuel de *t_block* est supérieur à 0.
- On empile un *t_block* à la position du nombre actuel de *t_block* qu'on incrémente ensuite.
- Pour dépiler, on décrémente le nombre actuel de *t_block* puis on retourne ce *t_block*.

Les fonctions publiques associées à un *t_regroupement* sont (8 au minimum) :
Le constructeur,
*t_regroupement init_regroupement*(**unsigned int** id, **int** taille);
reçoit l'identifiant unique d'un fichier et la taille maximale de la pile. Elle construit le tableau dynamique nécessaire et en cas de succès de l'allocation va fixer la taille, l'identifiant unique et le nombre d'éléments (en cas d'échec de l'allocation, tous les membres du *t_regroupement* seront nuls (0 ou NULL)). Elle retourne le nouvel objet.

**int** empiler_bloc(**t_regroupement** * reg, **t_block** bloc);
Va empiler le block reçu si les conditions nécessaires sont satisfaites, elle copie le bloc dans le tableau et retourne 1 si l'action réussit et 0 sinon.

**int** depiler_bloc(**t_regroupement** * reg, **t_block** * bloc);
Va dépiler un bloc si les conditions nécessaires sont satisfaites, elle copie le t_block du haut dans la référence reçue en paramètre et retourne 1 si l'action a réussie et 0 sinon.

**int** pile_blocs_pleine(**const t_regroupement** * reg);
Retourne 1 si la pile est pleine et 0 sinon.

**int** pile_blocs_vide(**const t_regroupement** * reg);
Retourne 1 si la pile est vide et 0 sinon.

**int** pile_blocs_taille(**const t_regroupement** * reg);
Retourne le nombre maximum de t_block dans la pile.

**int** pile_blocs_nombre(**const t_regroupement** * reg);
Retourne le nombre de t_block actuellement dans la pile.

**void** free_pile_blocs(**t_regroupement** * reg);
Libère le tableau dynamique et remet tous les membres à 0 ou à NULL.

---

**Mandat #2 -**
Vous testez dans votre « main » chacune des fonctions du module dès son implémentation, la qualité et la validité de ces tests sont essentielles.

Vous devez écrire ensuite une fonction de test dans votre fichier principal qui donne un fichier de taille moyenne au module de découpage et qui va récupérer tous les blocs émis dans un **t_regroupement** dont la pile est très grande et pourra contenir tous les blocs issus du fichier. Le découpage terminé, dépilez tous les t_block dans un tableau dynamique de la taille exacte nécessaire. Vous terminez en libérant toute la mémoire dynamique.

Refaites ce test mais sur deux fichiers simultanément. **Ces tests devront être remis.**

---

**Description du module « m_reconstruction »**

Le module *m_reconstruction* définit le type *t_reconstruction* dont une variable sera capable de conserver en ordre tous les objets *t_block* d'un unique fichier.
Une variable de type *t_reconstruction* contient les membres suivants :

- L'identifiant unique (*unsigned int*) d'un fichier
- Un pointeur donnant accès à un tableau dynamique de *t_block*
- La taille du tableau précédent
- Le nombre de *t_block* actuellement présents
- Le nombre total de *t_block* du fichier (qu'on obtient en recevant le dernier bloc du découpage)

*t_reconstruction* implémente un tableau dynamique capable de s'étirer durant l'exécution (avec realloc()). Chaque *t_block* admis dans une reconstruction possède le bon identifiant fichier et son numéro d'ordre d'extraction donnera sa case à occuper dans le tableau de reconstruction. Quand il recevra le dernier *t_block* du découpage (qui possède cette information unique), c'est avec cette réception qu'on pourra conclure sur la complétude des blocs du fichier-copie.

Les fonctions publiques associées à un *t_reconstruction* sont (7 au minimum) :
Le constructeur,
t_reconstruction init_reconstruction(**unsigned int** id, **int** taille);
reçoit l'identifiant unique d'un fichier et la taille initiale de son tableau de reconstruction. Il construit le tableau dynamique nécessaire et en cas de succès de l'allocation, le tableau est traversé et les identifiants-fichier sont tous mis à 0 dans les blocs puis la fonction fixe la taille, l'identifiant unique du fichier et le nombre total de blocks à 0 (en cas d'échec de l'allocation, tous les membres du t_reconstruction seront nuls (0 ou NULL)). On retourne le nouvel objet.

**int** redim_reconstruction(**t_reconstruction *** rec, **int** nouvelle_taille);
Reçoit l'adresse d'une reconstruction et d'une taille, elle essaie de redimensionner son tableau en conservant les blocs présents (allouez un nouveau tableau et copiez-y tous les blocs du tableau actuel avec memcpy). La fonction met l'identifiant fichier à 0 dans les nouvelles cases du tableau et ajuste les autres membres de la structure. On retourne 1 en cas de succès, 0 sinon.

**int** ajouter_bloc(**t_reconstruction *** rec, *t_block* bloc);
Si les conditions nécessaires sont satisfaites, on copie le bloc dans le tableau à sa position et on retourne 1 si l'action réussit, 0 sinon. On redimensionne le *t_reconstruction* si nécessaire.

**void** ajouter_pile_blocs(**t_reconstruction *** rec, **t_regroupement *** reg);
Si les deux structures sont du même fichier, la fonction va dépiler tous les blocs du regroupement pour les ajouter à la reconstruction (avec la fonction précédente).

**int** blocs_dans_reconstruction(**const t_reconstruction *** rec);
Retourne le nombre actuel de blocs du fichier dans la structure.

**int** etat_reconstruction (**t_reconstruction *** rec);
Retourne 1 si tous les blocs du fichier sont dans la structure et 0 sinon. Ceci se produit lorsque le nombre de t_block actuellement présents devient égal au nombre total de t_block du fichier.

**int** reconstruire_fich(*t_reconstruction* * rec, **const char *** nom_fichier);
Si la reconstruction est prête (avec fonction précédente) et le fichier binaire est bien ouvert en écriture, tous les blocs d'octets des *t_block* sont écrits, en ordre, dans le fichier puis le fichier est fermé. La fonction libère les tableaux dynamiques d'octets dans tous les *t_block* et libère aussi le tableau de *t_block* dans la structure. Elle retourne 1 en cas de succès et 0 sinon.

---

**Mandat #3** -
Vous devez écrire une fonction de test dans votre fichier principal qui initialise le découpage pour lui donner un fichier de taille moyenne à découper. La fonction se sert d'un *t_regroupement* initialisé volontairement avec une petite pile et d'un *t_reconstruction* initialisé volontairement trop court pour contenir tous les blocs issus du découpage. La gestion de vider la pile du *t_regroupement* ou d'étirer le tableau du *t_reconstruction* doit être testée et bien identifiée dans vos messages de débogage jusqu'à la reconstruction correcte d'une copie d'un fichier original quelconque.

Refaites cet exercice mais sur deux fichiers simultanément. **Ces tests devront être remis.**

---

**Première remise, la validation des trois mandats**

Avant jeudi le 1ier mars, vous devrez nous montrer (à moi ou Hugues) en laboratoire l'exécution correcte de vos fonctions de tests issues des trois mandats exigés dans le TP2. Cette remise partielle représente le tiers de la valeur totale du TP. Une documentation sommaire mais correcte est alors exigée, vous comprenez bien qu'on parle ici des commentaires (de module, de déclaration, de définition et des tests). Vous comprenez que sans réussite de certains mandats ou sans commentaires acceptables minimalement, des points sont automatiquement retirés.
Cette évaluation se fait obligatoirement en présence des membres de l'équipe. En terminant cette évaluation, nous récupérons une copie de tout votre code, preuve que votre présentation est complétée.

**Description du programme principal**
Votre fichier principal contient évidemment votre « main » mais aussi plusieurs fonctions qui vont factoriser la réalisation du TP2 à travers ses trois modules.

**Ainsi votre main devra effectuer les tâches suivantes :**
Déclarer un tableau de structures *t_regroupement* sans l'initialiser
Déclarer un tableau de structures *t_reconstruction* sans l'initialiser
Initialiser le découpage
Ajouter un ou des fichiers au découpage (ici ou même dans la boucle principale)

Répéter (boucle principale) :

- Récupérer un bloc
- Déclencher une fonction qui reçoit le bloc obtenu et les deux tableaux de structures, toute la gestion du bloc est automatique dont initialiser une structure au besoin, vider la pile d'un t_regroupement ou allonger le tableau d'une t_reconstruction
- Tester si une t_reconstruction est complète, si oui choisir un nom de fichier (différent de l'original) et déclencher la reconstruction complète de la copie et ensuite éliminer toute trace de ce fichier dans le programme

Tant qu'il y a des blocs à récupérer.

Vous devez noter que le deuxième point de la boucle doit être factorisé, vous pourrez avec les trois mandats remplis visualiser les taches indépendantes séquentiellement nécessaires au déroulement de cette tâche. La factorisation du programme principal est un point crucial du TP2.

Le restant de votre fichier principal contient toutes les fonctions de tests utilisées pour tester les 3 mandats. Veuillez placer toutes vos fonctions de tests en dernier dans le fichier principal.

**Exigences du travail et de la remise**

La date de remise finale du travail est prévue au plan de cours. La remise est avant tout électronique, directement zippé par courriel, avec tous vos fichiers-sources et le module fourni.

- L'ensemble de ce qu'on nomme la qualité du code dans votre implémentation représente au **minimum** le tiers des points de l'évaluation.
- Tout plagiat sera automatiquement traité selon les règlements de notre université.
- La politique du 10% pour la qualité du français sera appliquée en cas d'abus flagrant.
- Il ne devra contenir aucune variable globale, aucun *Goto* ni aucun *Exit*.

N'hésitez pas à nous demander de l'aide.
Bon travail !