TP2 CPS

I) Opérations bit à bit

1) Description

Le début du tp a consisté basiquement à créer les fonction get_bit, set_bit et clr_bit. Ces fonctions sont très simple et ont été pratiquées en ALM2.

get_bit(n, i) : on décale les bits de l'entier manipulé n de i rang, pour faire venir le bit recherché en bit de poids faible puis l'on met à zéro tous les autres bits.

set_bit(n, i) : il suffit de faire un ou bit à bit entre n et 1 placé en position i.

clr_bit(n, i) : il suffit de faire un et bit à bit entre n et 0 en position i et des 1 partout ailleurs.

2) Tests

Du fait de la simplicité de fonctionnement des fonctions, j'ai réalisé les tests avec le fichier bitabit. Cela a été réalisé manuellement car un bug dans le programme aurait été vu directement.

I) Fonctions d'Entrées/sorties bit à bit

1) Description

Tout d'abord, pour traiter un fichier bit à bit, j'ai créé une structure BFILE contenant les champs suivants :

- un pointeur vers le fichier f à traiter
- un char buffer qui servira à stocker les bits en attendant qu'ils puissent être traités par groupe de 8
- un int length repérant le nombre de bits dans le buffer
- un char mode gardant en mémoire le mode dans lequel nous sommes (read/write)

J'ai alors implémenté les différentes fonctions demandées, à savoir :

- `BFILE *bstart(FILE *fichier, const char *mode)`
  J'effectue le malloc de la structure et initialise tous ces champs en prenant soin de vérifier que l'allocation à réussie.
- `int bstop(BFILE *fichier)`
  Je me contente de libérer la structure utilisée.
- `char bitread(BFILE *fichier)`
  Je vérifie dans un premier tant que le mode est compatible avec l'opération à réaliser. On veut ici effectuer une lecture, le mode doit donc être « r ». L'idée est de remplir le buffer lorsqu'il est vide avec un caractère lu dans le fichier. Par la suite, je récupère le bit de poids fort du buffer à chaque lecture de bit. Je décale également les bits dans le buffer pour que le bit à lire soit toujours au même emplacement. Je n'oublie pas de diminuer à chaque fois la valeur de length, nous servant à repérer si le buffer est vide ou non.
- `int bitwrite(BFILE *fichier, char bit)`
  Encore une fois, je vérifie que le mode est compatible avec l'opération à réaliser. L'idée est de remplir le buffer avec le bit donné en paramètre. Tant que le buffer n'est pas plein, on ajoute le bit en bit de poids faible (pour être cohérent avec

bitread). Dés que le buffer est plein, on peut écrire l'octet dans le fichier, vider le buffer et remettre length à zéro.

- `int beof(BFILE *fichier)`
  Je vérifie si le buffer contient le EOF. Cependant, cela ne fonctionne que si les séquences utilisées sont des mutliples de 8.

J'ai par la suite implémenter une solution pour pouvoir avoir une séquence de taille quelconque, non nécessairement multiple de 8. J'ai pour cela effectuer des modifications uniquement dans les fichiers test_read et test_write. Bien que cela me soit dans un premier temps apparu cohérent, je comprend désormais qu'il aurait été préférable d'implémenter ces modifications directement dans les fonctions bitread et bitwrite. J'ai cependant décidé de continuer sur cette voie, ayant déjà grandement avancé.

Concernant le problème de fin de données, j'ai donc implémenté le système de directive, en utilisant les octets recommandés dans le sujet. Ainsi, à chaque séquence de 8 bits, j'intègre, avant cette séquence, l'octet d'indication de directives, suivi de l'octet de directive. Ainsi, chaque octet de donnée est associé avec les octets 0xFF 0xFF sauf le dernier qui est associé avec 0xFF 0x0? avec ? correspondant au nombre de bit supplémentaire.

En pratique, test_write code la séquence en fournissant un octet de donnée à chaque fois. Ainsi, à la fin de la séquence, même si il ne reste que 4 bits à écrire, ces 4 bits seront placé au début d'un octet des 0 partout ailleurs. Les deux octets précédant nous servent alors à définir combien de bits doivent être lus par test_read.

2) Tests

Pour effectuer les tests, j'ai voulu utiliser le générateur de séquence avec le script proposé. Une fois lancé, le script effectue les appels aux fonction test_read et test_write et compare le résultat. Cependant, bien que la séquence initiale et la séquence décodée sont identiques, le script annonce que le résultat n'est pas correcte. Cela pourrait être du à un retour à la ligne dans l'un des deux fichiers mais ne sachant pas d'où il vient, je n'ai pas réussi à résoudre le problème.