TP8 : Surcouche d'entrées / sorties

Borne

1 Fonction `ouvrir`

1.1 Spécification

`FICHIER* ouvrir(char *nom, char mode)`

Prends en paramètre un nom de fichier `nom` et un caractère `mode`, retourne l'adresse d'une structure `FICHIER`.

1.2 Sémantique

La fonction `ouvrir`, utilise l'appel système `open` pour ouvrir le fichier dont le nom est fourni en paramètre et initie un flux de données en écriture ou en lecture vers/depuis le fichier. La nature du flux est spécifiée par le paramètre `mode` qui peut prendre les valeurs :

- `'R'` pour un flux en lecture.
- `'W'` pour un flux en écriture.

La fonction initialise une structure `FICHIER` contenant :

- Le descripteur du fichier retourné par l'appel système `open`
- Un tampon destiné à envoyer/recevoir des données vers/depuis le fichier par blocs d'octet.

La fonction retourne `NULL` si l'ouverture de fichier a échoué ou si le mode est invalide.

2 Fonction `fermer`

2.1 Spécification

`int fermer(FICHIER* f)`

Prends en paramètre l'adresse d'une structure FICHIER.

2.2 Sémantique

La fonction `fermer`, utilise l'appel système `close` pour fermer le fichier dont le descripteur appartient à la structure `f`. En cas d'échec la fonction retourne -1.

2.3 Pré-condition

le paramètre `f` est un pointeur vers une structure `FICHIER` retournée par la fonction `ouvrir`.

2.4 Post-condition

Les données présentes dans le buffer de la structure sont écrites dans le fichier dont le descripteur est `f->file`. Le fichier est fermé. L'espace mémoire de la structure `FICHIER` pointée par `f` est libéré.

3 Fonction `lire`

Définition. On appelle *élément* un bloc d'octets.

3.1 Spécification

`int lire(void *p, unsigned int taille, unsigned int nbelem, FICHIER *f)`

Prends en paramètre :

- Une adresse `p` où stocker les éléments lus.
- Un entier positif `taille` qui détermine la taille d'un élément à lire en nombre d'octets.
- Un entier positif `nbelem` qui correspond au nombre d'éléments à lire depuis le fichier.
- L'adresse d'une structure `FICHIER`.

3.2 Sémantique

La fonction lire, lis un nombre d'éléments `nbelem` depuis le fichier dont le descripteur appartient à la structure `f` fournie en paramètre. Un élément est un 'bloc' d'octet dont la taille est spécifiée par le paramètre `taille`. Les éléments lus sont stockés à l'adresse `p` fournie en paramètre. Retourne le nombre d'éléments lus en cas de succès, -1 en cas d'échec.

Remarque. Si $nbelem * taille > taille\_fichier$ (en octets). On lit le nombre maximum possible d'éléments que le fichier peut contenir.

3.3 Mise en oeuvre

On calcule le nombre total d'octets que l'on doit lire dans le fichier. On itère l'opération suivante. Tant que l'on a pas lu `nb_total_octets_a_lire` :

On remplit le buffer de la structure `FICHIER` passée en paramètre avec un appel `read(File, buffer, BUFFER_SIZE)` puis on procède à la lecture des octets depuis ce buffer.

3.4 Pré-condition

Le paramètre `f` est un pointeur vers une structure `FICHIER` valide retournée par la fonction ouvrir avec le mode `'R'` (lecture). Suffisamment d'espace doit être disponible à l'adresse `p` pour stocker les données lues.

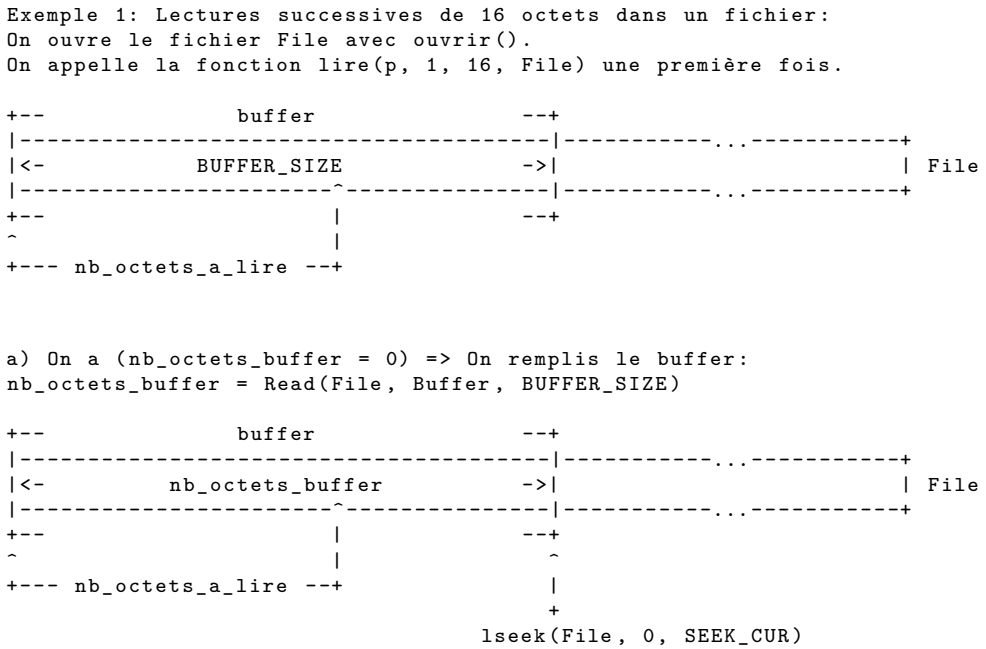
3.5 Post-condition

Les `nbelem` éléments lus depuis le fichier décrit par le descripteur `f->file` sont stockés à l'adresse `p`.

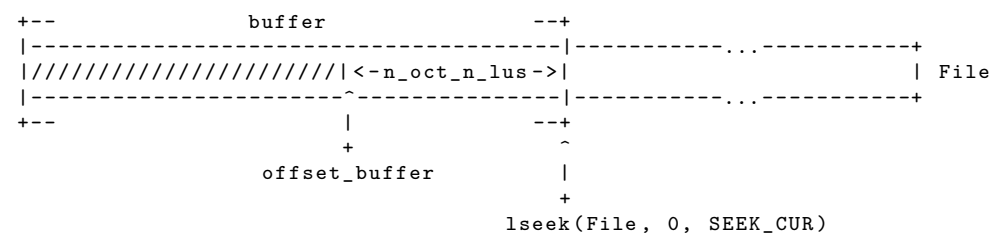
3.6 Valeur retour

Nombre d'éléments lus ou -1 si échec lors de la lecture.

3.7 Illustrations

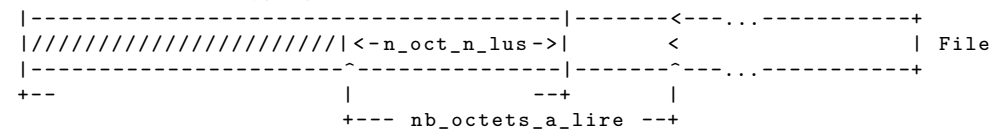


b) On lis les nb_octets_a_lire_buffer_courant et on les écrit en mémoire à
l'adresse p.
A l'issue de la lecture la structure FICHIER est dans l'état suivant:

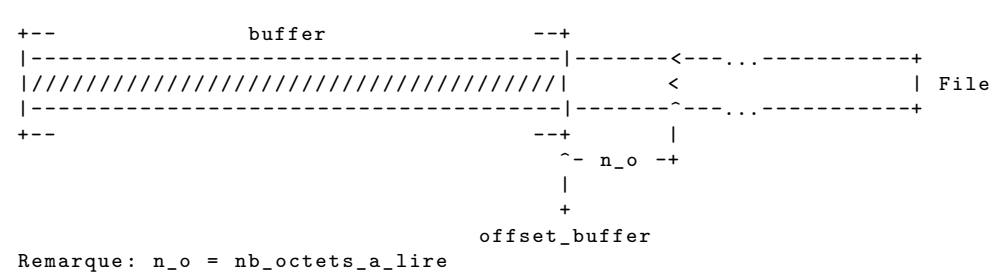


Remarque: offset_buffer est la position du dernier octet du buffer courant lu et écrit
en mémoire.

c) On Appelle une nouvelle fois la fonction lire(p, 1, 16, File):

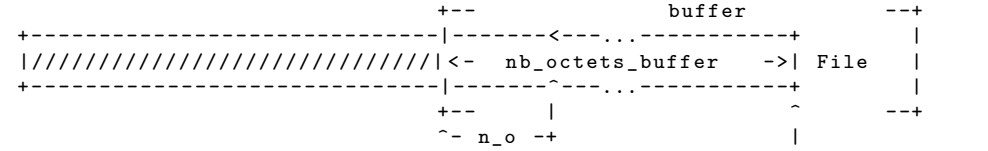


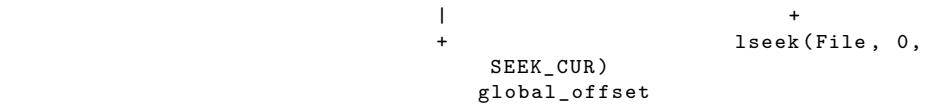
d) On lis les nb_octets_non_lus_buffer depuis le buffer et on les stocke à l'adresse p.
A l'issue de la lecture la structure FICHIER est dans l'état suivant:



Remarque: n_o = nb_octets_a_lire

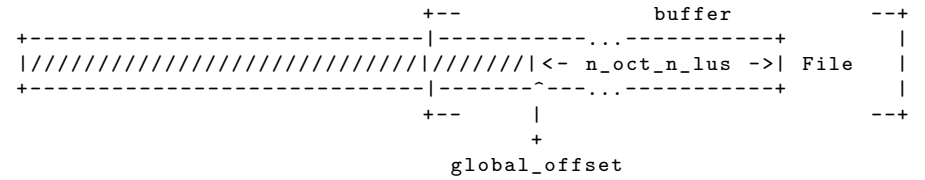
e) On a plus aucun élément à lire dans le buffer courant donc on remplit le buffer avec les données suivantes du fichier.





Remarque: global_offset est la position du dernier octet du fichier lu et écrit en mémoire.

f) On lis les n_o octets restant à lire depuis le buffer courant et on les stocke à l'adresse p.
A l'issue de la lecture la structure FICHIER est dans l'état suivant:



4 Fonction `écrire`

4.1 Spécification

`int ecrire(const void *p, unsigned int taille, unsigned int nbelem, FICHIER *f)`

Prends en paramètres :

— Une adresse `p` où sont stockées les données à écrire.

— Un entier positif `taille` indiquant la taille en nombre d'octets d'un élément à lire.

— entier positif `nbelem` indiquant le nombre d'éléments à lire.

— L'adresse `f` d'une structure de type `FICHIER`.

4.2 Sémantique et mise en oeuvre

Écrit `nbelem` éléments dans le fichier `f->file`. L'écriture s'effectue par l'intermédiaire du buffer de la structure `f`. On écrit les données du buffer dans le fichier seulement quand le buffer est plein ou que l'on ferme le fichier à l'aide de la fonction `fermer()`.

4.3 Pré-conditions

le paramètre `f` est un pointeur vers une structure `FICHIER` valide retournée par la fonction ouvrir avec le mode `'W'` (écriture). `p` pointe vers une zone mémoire contenant au moins `nbelem` éléments.

4.4 Post-conditions

Les `nbelem` éléments lus à l'adresse `p` sont soit écrits dans le fichier dont le descripteur est `f->file` soit dans la mémoire tampon de la structure `f` en attente d'être écrits. Attention, si, après son utilisation, le fichier `f` n'est pas fermé de manière approprié en appelant la fonction `fermer(f)` les données présentes dans le buffer seront perdues.

4.5 Valeur de retour

Le nombre d'éléments écrits ou -1 en cas d'échec d'écriture.

5 Fonction `fecriref`

5.1 Spécification

`int fecriref (FICHIER *fp, char *format,...)`
Prends en paramètres :
— L'adresse f d'une structure de type FICHIER.
— Une chaîne de caractères "format".
— Une liste de données formatées optionnelle.

5.2 Chaîne "format" et paramètres optionnels

La chaîne format est une séquence de caractères destinée à être écrite dans le fichier décrit par la structure fp. Cette séquence peut contenir les mots spéciaux suivants : "%s", "%c", "%d". Chaque occurrence d'un de ces mots est substituée, lors de l'écriture dans le fichier, par la valeur d'une donnée passée en option et dont le type correspond au type associé au mot.

5.3 Types/formats des données en paramètre

Les types/formats des données optionnelles sont interprétés selon l'utilisation des marqueurs %c %s ou %d dans la chaîne de caractère "format" passée en paramètre.
— %c : caractère
— %s : chaîne de caractères
— %d : entier

5.4 Pré-conditions

1. Le paramètre f est un pointeur vers une structure FICHIER valide retournée par la fonction ouvrir avec le mode "W" (écriture).

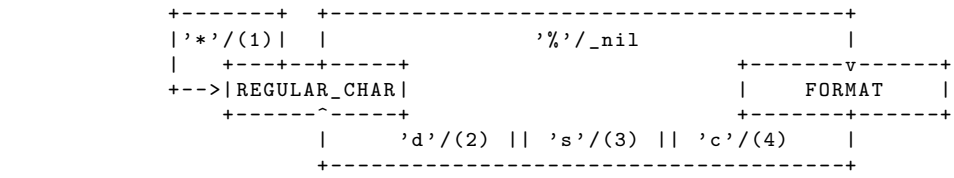
2. Si des données formatées sont fournies dans les paramètres optionnels : Il doit exister une correspondance bi-univoque entre les données formatées passées en paramètre et les formats spécifiés dans la chaîne format. En particulier :
   — Si dans la chaîne "format" on trouve n mots spéciaux "on doit avoir n données formatées en paramètres.
   — L'ordre et le type des formats spécifiés dans la chaîne format doit correspondre à l'ordre et au type des données. (voir exemples)

5.5 Exemples

— `fecriref(file, "%c", 'c')` écrit dans le fichier `file` la valeur `ASCII` du caractère 'c'.
— `fecriref(file, "%s", "exemple")` écrit dans le fichier `file` la chaine "exemple".
— `fecriref(file, "%d", 1245)` écrit dans le fichier la représentation `ASCII` de l'entier 1245 i.e la chaîne de caractère "1245".
— `fecriref(file, "texte %d %s", 45, "fin texte")` écrit dans `file` la chaîne texte suivie de la représentation `ASCII` de l'entier 45 suivie de la chaîne "fin texte".
— `fecriref(file, "%d %s", "texte", 45)` : l'ordre des données ne correspond pas au format spécifié -> comportement non spécifié.

5.6 Mise en oeuvre

Dans la chaîne de caractère "format" on distingue deux types de caractères. Les caractère normaux qui sont écrits directement dans le fichier et les caractères spéciaux '%' qui permettent d'insérer des données formatées fournies dans les arguments optionnels. On utilise l'automate d'état fini suivant pour lire, caractère par caractère, la chaîne `format` et traiter si besoin les données formatées à insérer dans le fichier :




```
Légende:
  - char / (x): lit l'entrée char et exécute l'action x.
  - -->: transitions.
  - '*': tout caractère sauf '%'.
Actions:
```

```
- (1): écrit le caractère courant de la chaîne "format" dans le
    fichier.
- (2): récupère la prochaine donnée (qui doit être un entier)
        dans la liste des arguments optionnels et écrit sa
            représentation ASCII
        dans le fichier.
- (3): récupère la prochaine donnée (qui doit être une chaîne de
    caractères)
            dans la liste des arguments optionnels et l'écrit dans le
            fichier.
- (4): récupère la prochaine donnée (qui doit être un caractère)
        dans la liste des arguments optionnels et l'écrit dans le
            fichier.
```

6 Fonction `fliref`

Définition. On appelle *caractère normal* tout caractère n'étant pas un espace ou le caractère '%'.

6.1 Spécifications

`int fliref (FICHIER *fp, char *format,...)`
Prends en paramètres :
— L'adresse `f` d'une structure de type `FICHIER`.
— Une chaîne de caractères `format`.
— Une liste optionnelle de pointeurs.

6.2 Sémantique

Lis depuis le fichier représenté par la structure `fp`, un ensemble de données dont le format attendu est spécifié dans la chaîne `format`. Les données lues sont stockées aux adresses de pointeurs passés en option à la fonction.

6.3 Chaîne `format`

La chaîne format est une séquence de caractères décrivant le format des données que l'on s'attend à lire dans le fichier `fp`. Cette séquence peut contenir les mots spéciaux suivants : `"%s"`, `"%c"`, `"%d"` des caractères `ASCII` normaux et des caractères espaces ' '. Les types/formats des données sont interprétés par l'utilisation des mots %c %s ou %d dans la chaîne de caractère `format`. passée en paramètre.

%c : caractère

On lit un unique caractère dans le fichier.

%s : chaîne de caractères

On lit une séquence de caractères non `' '` (espace).

%d : entier

On lit une séquence de caractères décimaux depuis le fichier.

Caractères normaux et espaces

La présence d'un caractère espace dans la chaîne format signifie que l'on s'attend à lire un ou plusieurs espaces dans le fichier. La présence d'un caractère `ASCII` 'normal' signifie que l'on s'attend à lire le caractère en question dans le fichier.

6.4 Pré-conditions

1. Le paramètre `f` est un pointeur vers une structure `FICHIER` valide retournée par la fonction `ouvrir` avec le mode `L` (lecture).
2. Il doit exister une correspondance bi-univoque entre les mots spéciaux (présents dans la chaîne format et les pointeurs passés en paramètres optionnels. En particulier :
   — Si dans la chaîne `format` on trouve n mots spéciaux "%x" ,$x \in \{s, c, d\}$ on doit avoir n pointeurs de type correspondant en paramètre.
   — L'ordre et le type des formats spécifiés doit correspondre à l'ordre et au type des pointeurs passées en paramètres.
3. L'ordre des types de données spécifiés dans la chaîne `format` doit correspondre à l'ordre dans lequel on rencontre les types de données dans le fichier.

6.5 Post-conditions

Les données formatées lues depuis le fichier sont stockées aux adresses des pointeurs fournis en paramètre.

6.6 Valeur de retour

Nombre de caractères lus depuis le fichier ou -1 en cas d'échec de la lecture.

7 Tests

Nous avons fourni dans le répertoire `Tests` un ensemble de programmes que l'on peut compiler et exécuter à l'aide de la commande `make tests`. L'ensemble

des fonctions implémentées sont testées dans ces programmes, dans différentes cas d'utilisation (ouverture, lecture fichier, écriture, fermeture, ...).

8 Pistes d'améliorations

Nous souhaiterions en premier lieu améliorer notre méthodologie lors des procédures de test par l'utilisation d'un framework de test unitaires. Nous allons essayer d'investir le temps nécessaire à la mise en place d'outils de test plus efficaces pour nos développements futurs. D'autre part la fonction `fliref` comporte, pour la gestion des entiers dans la chaîne `format`, un traitement itératif dont l'écriture devrait être factorisée (traitement du signe en dehors de la boucle) et simplifiée (condition i==1 dans la terminaison). Le code présenté ici souffre de nombreuses lacunes qui mériteraient d'êtres corrigées. Les fonctions ne sont pas thread-safe. Le comportement du buffer d'écriture pourrait être modifié pour forcer l'écriture dans le fichier lorsqu'un caractère de retour à la ligne est rencontré en entrée. Les invariants de boucles devrait êtres énoncés. La correction des fonctions `fliref` et `fecriref` n'est pas parfaitement garantie dans l'état actuel. Bien que nos tests indiquent des résultats positifs ceux-ci devraient être refondus pour traiter plus de cas et mettre en avant les situations pathologiques.

9 Conclusions

Nous avons adopté un style de programmation par contrat, et avons essayé de spécifier au mieux l'utilisation attendue de nos fonctions par l'utilisateur en vue d'un résultat correct. Nous avons essayé de fournir une documentation lisible et exhaustive. Nous avons perçu sur ces fonctions simples certaines difficultés à énumérer les différents cas à traiter lors de la lecture ou l'écriture de données formatées dans un fichier. Nous avons apprécié travailler sur ce sujet et espérons avoir fourni une documentation agréable à lire.