SCR.1.2 TP 11 ⊥ :

Appels système pour la réalisation

des entrées sorties

Note. Les boucles de lecture dont il est question dans ce TP doivent être de la forme `while(n=read(....,....,....))` et surtout pas `do ... while`.

Pour chacun des cas suivants, écrire le programme C correspondant.

1. Un programme `read_file.c` affiche le contenu d'un fichier à l'écran. Le nom du fichier est donné à l'exécutable à la ligne de commande. Le processus demande l'ouverture du fichier à l'aide de l'appel sysème `open`, puis entre dans une boucle de lecture dans le fichier par tranches, disons de 256 octets ; valeur introduite par une directive `#define`. La sortie de la boucle est contrôlée par la valeur de retour de l'appel système `read`. L'affichage est réalisé par l'appel système `write`.

   On doit gérer l'erreur sur le nombre de paramètres à la ligne de commande. On doit gérer les erreurs sur les appels systèmes à l'aide de la routine `perror`.

2. Faire une copie de `read_file.c` dans `copy.c`. Modifier `copy.c` pour obtenir un programme qui réalise la copie d'un fichier. Les noms des fichiers source et destination sont donnés à l'exécutable à la ligne de commande. La gestion des erreurs doit être réalisée. Le programme crée le fichier destination s'il n'existe pas et le tronque (remet sa taille à 0) s'il existe.

3. Un programme `store_numb.c` saisit des nombres entiers par interaction avec l'utilisateur en utilisant l'appel système `read` dans une boucle dont on sort lorsque la fin de fichier sur l'entrée standard est détectée. Chaque nombre saisi est enregistré sous sa forme texte, un nombre par ligne, dans un fichier dont le nom a été donné en argument à l'exécutable. On rappelle que la fin de fichier sur l'entrée standard est signifiée par `CTRL-D`. La gestion des erreurs doit être réalisée. Le programme crée le fichier s'il n'existe pas et le tronque (remet sa taille à 0) s'il existe.

```
$ ./store_numb
Usage: ./store_numb <file_name>
$ ./store_numb numbers.txt
Numb --> 15
Numb --> -1
Numb --> 4
Numb --> 0
Numb --> 12
Numb --> 7
Numb --> $ cat numbers.txt
15
-1
4
0
12
7
$ hexdump -C numbers.txt
00000000  31 35 0a 2d 31 0a 34 0a  30 0a 31 32 0a 37 0a     |15.-1.4.0.12.7.|
0000000f
```

4. Un programme `store_nb_rep.c` saisit des nombres entiers par interaction avec l'utilisateur en utilisant l'appel système `read` dans une boucle dont on sort lorsque la fin de fichier sur l'entrée standard est détectée. **La représentation 32 bits** de chaque nombre saisi est enregistrée à l'aide de l'appel système `write`, l'une à la suite de l'autre, dans un fichier dont le nom a été donné en argument à l'exécutable. On utilisera la fonction `strtol` (`man strtol`). On rappelle que la fin de fichier sur l'entrée standard est signifiée par `CTRL-D`.

La gestion des erreurs doit être réalisée. Le programme crée le fichier s'il n'existe pas et le tronque s'il existe.

```
$ ./store_nb_rep
Usage: ./store_nb_rep <file_name>
$ ./store_nb_rep numbers.dat
Numb --> 15
Numb --> -1
Numb --> 4
Numb --> 0
Numb --> 12
Numb --> 7
Numb --> $ hexdump -C numbers.dat
00000000  0f 00 00 00 ff ff ff ff  04 00 00 00 00 00 00 00  |................|
00000010  0c 00 00 00 07 00 00 00                           |........|
00000018
```

5. Un programme `get_numb.c` dont l'exécutable reçoit deux paramètres à la ligne de commande considère que son premier argument est un fichier dont le contenu est une suite de représentations de nombres entiers 32 bits. Le programme affiche en hexadécimal la valeur du nombre au déplacement `i` où la valeur de `i` est donnée sous forme décimale en deuxième argument à ligne de commande. Les déplacements comptent le nombre d'entiers à partir de 0. On utilisera la fonction `strtol` pour convertir le deuxième argument de la forme chaîne de caractères à la forme représentation d'un entier.

   On peut utiliser `hexdump` pour vérifier que la valeur affichée est bien celle du nombre se trouvant dans le fichier au déplacement donné.

```
$ ./get_numb numbers.dat
Usage: ./get_numb <file_name> <offset>
$ ./get_numb numbers.dat 4
The number at offset 4 is 0x0000000c --> 12
$ ./get_numb numbers.dat 1
The number at offset 1 is 0xffffffff --> -1
$ ./get_numb numbers.dat 0
The number at offset 0 is 0x0000000f --> 15
$ ./get_numb numbers.dat 6
Offset is out of range!
$ ./get_numb numbers.dat 5
The number at offset 5 is 0x00000007 --> 7
$
```

6. Un programme `put_numb.c` dont l'exécutable reçoit trois paramètres à la ligne de commande considère que son premier argument est un fichier dont le contenu est une suite de représentations de nombres entiers 32 bits. Le programme écrit au déplacement `i` la représentation de l'entier dont la valeur est donnée en hexadécimal en deuxième argument à la ligne de commande. La valeur du déplacement est le troisième argument de l'exécutable donnée sous forme hexadécimale. Les déplacements comptent le nombre d'entiers à partir de 0. Lorsque la valeur de l'offset amène au-delà de la taille du fichier, le programme écrit en fin de fichier.

   On utilisera la fonction `strtol` pour convertir les arguments de la forme chaîne de caractères à la forme représentation d'un entier.

```
$ hexdump -C numbers.dat
00000000  0f 00 00 00 ff ff ff ff  04 00 00 00 00 00 00 00
00000010  0c 00 00 00 07 00 00 00
00000018
$ ./put_numb 5
Usage: ./put_numb <file_name> <32-bit int in Hex> <Offset in Hex>
$ ./put_numb numbers.dat 5 0
```

```
$ hexdump -C numbers.dat
00000000  05 00 00 00 ff ff ff ff  04 00 00 00 00 00 00 00
00000010  0c 00 00 00 07 00 00 00
00000018
$ ./put_numb numbers.dat 11221122 3
$ hexdump -C numbers.dat
00000000  05 00 00 00 ff ff ff ff  04 00 00 00 22 11 22 11
00000010  0c 00 00 00 07 00 00 00
00000018
$ ./put_numb numbers.dat 33443344 0
$ hexdump -C numbers.dat
00000000  44 33 44 33 ff ff ff ff  04 00 00 00 22 11 22 11
00000010  0c 00 00 00 07 00 00 00
00000018
$ ./put_numb numbers.dat babababa 7
$ hexdump -C numbers.dat
00000000  44 33 44 33 ff ff ff ff  04 00 00 00 22 11 22 11
00000010  0c 00 00 00 07 00 00 00  ba ba ba ba
0000001c
$ ./put_numb numbers.dat ffffffff ab
$ hexdump -C numbers.dat
00000000  44 33 44 33 ff ff ff ff  04 00 00 00 22 11 22 11
00000010  0c 00 00 00 07 00 00 00  ba ba ba ba ff ff ff ff
00000020
$
```