École de technologie supérieure

THÈME 12
Fiches et fichiers

INF125 Introduction à la programmation
Sylvie Ratté et Hugues Saulnier

Jusqu'à présent, nous avons su tirer parti des capacités de redirection en DOS pour lire des fichiers texte et en produire. Malheureusement, lorsque nous avons à manipuler plusieurs fichiers cet astuce ne peut plus être utilisé. De plus, la lecture "binaire" (lecture par bloc de "n" octets) des fichiers ne peut pas être effectuée par redirection. Il faut donc ici introduire des variables d'un type permettant l'accès à des fichiers en mode texte ou binaire. La section 1 de ce thème présente les fonctions de "stdio.h" vous permettant de manipuler des fichiers en mode texte tandis que la seconde partie, présente les fonctions de la même librairie vous permettant de manipuler des fichiers en mode binaire (ce qui est plus efficace).

1. La librairie "stdio" en bref

A. Présentation

La librairie "stdio" présente les groupes de fonctions suivantes (nous considérons uniquement les fonctions entièrement ANSI):

Fonctions générales d'ouverture et de fermeture:

fopen	ouverture d'un fichier dans le mode texte ou binaire, en écriture ou en lecture
fclose	fermeture du fichier

Fonctions particulières à l'ouverture en mode texte

fprintf	un "printf" qui s'effectue dans un fichier ouvert en écriture plutôt que sur "stdout"
fscanf	un "scanf" qui s'effectue dans un fichier ouvert en lecture plutôt que sur "stdin"
fgets	un "gets" qui s'effectue dans un fichier ouvert en lecture plutôt que sur "stdin"
fputs	un "puts" qui s'effectue dans un fichier ouvert en écriture plutôt que sur "stdout"
fgetc	un "getchar" qui s'effectue dans un fichier ouvert en lecture plutôt que sur "stdin"
fputc	un "putchar" qui s'effectue dans un fichier ouvert en écriture plutôt que sur "stdout"

Fonctions particulières à l'ouverture en mode binaire

ftell	retourne la position de l'indicateur dans le fichier
fseek	déplace l'indicateur dans le fichier
fread	permet de lire un bloc de N octets
fwrite	permet d'écrire un bloc de N octets

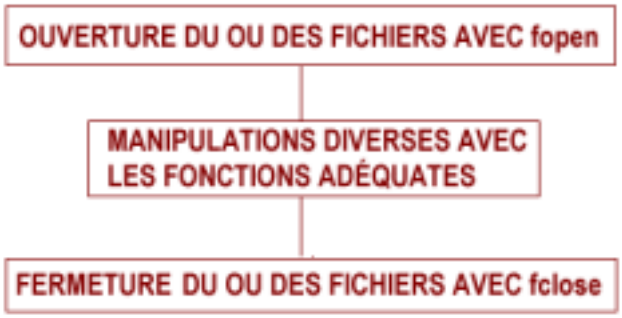
Fonctions de contrôle d'erreurs

feof	indique que la fin du fichier est atteinte
ferror	indique la présence d'une erreur ou non
clearerr	permet d'évacuer les erreurs
fflush	vide le tampon
rewind	permet d'évacuer les erreurs tout en retournant au début du fichier

Fonctions de manipulation de fichiers

remove	permet d'effacer un fichier sur disque
rename	permet de renommer un fichier sur disque

Pour manipuler un fichier en mode texte ou binaire, il faut évidemment d'abord l'ouvrir. On y effectue ensuite le traitement désiré. Lorsque les manipulations sont terminées, on doit terminer en fermant le fichier. Schématiquement, cela revient à:



L'ouverture, la fermeture et le contrôle des erreurs sont des tâches commune aux deux modes d'utilisation des fichiers (texte ou binaire). Rien de mieux qu'un exemple pour comprendre le tout. Voici un petit programme qui ouvre, en lecture, un fichier contenant uniquement des nombres entiers, y lit 3 petits entiers (les trois premiers) pour les afficher un à un à l'écran.

```
int main(void){

  FILE * source;                          pointeur vers le fichier qui sera ouvert en lecture

  int n;                                  un nombre entier quelconque que nous lirons dans le fichier

  source = fopen("nombre.txt", "rt");     ouverture du fichier...

  if (!source) return 0;                  si le fichier n'existe pas, on sort !

  fscanf(source, "%i", &n);               lecture d'un premier entier

  printf("%i ", n);                       affichage à l'écran

  fscanf(source, "%i", &n);               lecture d'un deuxième entier

  printf("%i ", n);                       affichage à l'écran

  fscanf(source, "%i", &n);               lecture d'un troisième entier

  printf("%i ", n);                       affichage à l'écran

  fclose(source);                         fermeture du fichier

  return 0;                               on a terminé

}
```

Examinons d'abord les possibilités des tâches communes aux deux modes.

B. Ouverture et fermeture d'un fichier

B.1. OUVERTURE

Lorsqu'on désire ouvrir un fichier, on doit connaître son nom (et son emplacement) et savoir si l'on désire y lire des données ou en écrire. Pour faire le lien entre notre code et le fichier sur disque on utilise une variable de type "`FILE *`". La fonction "fopen" effectue sa tâche en exigeant deux arguments: le premier est un pointeur sur une chaîne de caractères (plus précisément, c'est donc un pointeur sur le premier caractère d'une suite de caractères) contenant le nom du fichier (possiblement précédé du chemin d'accès sur disque) et le second est un pointeur sur une chaîne de caractères décrivant le mode d'ouverture désiré. La fonction retourne un pointeur sur un objet `FILE`. Dit autrement, la fonction retourne un résultat de type "`FILE *`". Son prototype est le suivant:

```
FILE * fopen(char * nom, char * mode);
```

Le mode d'ouverture peut combiner les valeurs suivantes:

- `"t"` : ouverture en mode texte (mode d'ouverture par défaut)
- `"b"` : ouverture en mode binaire
- `"r"` : ouverture en lecture (**erreur** si le fichier n'existe pas)

- `"w"` : ouverture en écriture (si le fichier n'existe pas, il est créé)
- `"a"` : ouverture en écriture de type "ajout à la fin" (append) (si le fichier n'existe pas, il est créé)

Les combinaisons suivantes sont ainsi possibles :

- `"rt"` : ouverture en lecture en mode texte
- `"r"` : idem (le mode texte est le mode par défaut)
- `"rb"` : ouverture en lecture en mode binaire
- `"wt"` : ouverture en écriture en mode texte
- `"w"` : idem (le mode texte est le mode par défaut)
- `"wb"` : ouverture en écriture en mode binaire
- `"r+t"` : ouverture en lecture/écriture d'un même fichier en mode texte
- `"r+b"` : ouverture en lecture/écriture d'un même fichier en mode binaire

En cas d'échec, la fonction "fopen" retourne NULL. Ainsi, habituellement lorsqu'on ouvre un fichier, on effectue un test sur le résultat:

```
{
  FILE * source;
  source = fopen("message.txt", "rt");
  if (!source) {
    printf("Échec lors de l'ouverture du fichier!");
    return 0;
  }
  ... le reste du traitement...
```

B.2. FERMETURE

Peu importe le mode d'ouverture du fichier, celui-ci doit toujours être fermé si l'on espère y retrouver les modifications que nous y avons apportées. La fonction "feof" s'applique ainsi sur notre pointeur de type FILE *. Elle retourne 0 si le fichier a été correctement fermé, sinon (échec) elle retourne EOF. Son prototype est le suivant:

```
int fclose(FILE * flux);
```

La fonction a également pour effet de vider tous les tampons liés à ce fichier.

C. Le contrôle des erreurs

Quatre fonctions permettent de gérer les erreurs éventuelles: feof, ferror, clearerror et fflush.

```
int feof(FILE * flux);
```

La fonction "feof" retourne une valeur différente de zéro si la fin du fichier EST ATTEINTE. Une boucle de lecture typique pourrait ainsi s'écrire:

```
while (!feof(source)) {
 ... traitement ...
}
```

Nous verrrons cependant que ce type de boucle de lecture est plutôt utilisée dans les fichiers ouverts en mode texte et non les fichiers ouverts en mode binaire.

```
int ferror(FILE * flux);
```

La fonction "ferror" retourne une valeur différente de zéro si une erreur est survenue. Il peut s'agir d'une erreur de flux (ex. tentative d'écrire dans un fichier ouvert en lecture) ou que la fin du fichier est atteinte (ex. tentative de lire après la fin du fichier).

```
void clearerr(FILE * flux);
```

La fonction "clearerr" remet les indicateurs d'erreur (fin de fichier et erreur de flux) à zéro.

```
int fflush(FILE * flux);
```

La fonction "fflush" vide les tampons associés au fichier. Retourne 0 s'il y a succès et EOF s'il y a échec. on l'utilise lorsque le système effectue de la tamporisation" Les écritures, par exemple ne sont pas effectuées immédiatement dans le fichier mais sont effectuées dans un tampon en mémoire. Lorsque le "fclose" est effectué, ou encore lorsque le système le décide, le contenu du tampon est transféré sur disque. Pour forcer ce transfert, on peut utiliser "fflush". En lecture, la fonction est rarement utilisée car après un "fflush" sur un fichier ouvert en lecture, la prochaine lecture rencontrera la fin du fichier, le "fflush" ayant pour effet de "sauter" ce qui reste à lire dans le fichier.

```
void rewind(FILE * flux);
```

La fonction "rewind" permet de retourner l'indicateur au début du fichier. Elle élimine du même coup toutes les erreurs. Autrement dit, elle effectue un "clearerr".

D. Manipulation de fichiers

```
int remove(const char * flux);
```

La fonction "remove" permet, comme son nom l'indique, d'effacer le fichier dont le nom est fourni en argument. En cas de succès, la fonction retourne 0. En cas d'échec, la fonction retourne -1 et place dans la variable globale "errno" la valeur ENOENT ("No such file or directory") ou EACCES ("permission denied").

La variable "errno" est une variable globale appartenant à "stdio". Pour l'utiliser, on doit donc la déclarer comme suit (rappelez-vous du thème 8):

```
extern int errno;
```

```
int rename(const char * anciennom, const char * nouveaunom);
```

La fonction "rename" permet, comme son nom l'indique, de modifier le nom d'un fichier, En cas de succès, la fonction retourne 0. En cas d'échec, la fonction retourne -1 et place dans la variable globale "errno" la valeur ENOENT ("No such file or directory"), EACCES ("permission denied") ou ENOTSAM ("Not same device"). Cette fonction n'est pas portable sous UNIX.

2. Fichiers en mode texte ou binaire

A. Fichiers ouverts en mode texte

Un fichier ouvert en mode texte s'utilise exactement comme si l'on effectuait des lectures en interactif. La seule différence est la présence d'un argument supplémentaire, la variable FILE *, dans les appels de fonctions.

```
int fscanf(FILE * flux, char * format [, adresses]);
```

Si vous connaissez "scanf" alors vous connaissez "fscanf". La fonction retourne le nombre de conversions de type effectuées avec succès. En cas d'échec, la fonction retourne EOF.

```
int fprintf(FILE * flux, char * format [, arguments]);
```

Si vous connaissez "printf" alors vous connaissez "fprintf". La fonction retourne le nombre d'OCTETS écrits. En cas d'échec, elle retourne zéro.

```
char * fgets(char * s, int N, FILE * flux);
```

La fonction "fgets" permet de lire des caractères et des les placer dans la chaîne "s". La fonction s'arrête de lire lorsque (n-1) caractères ont été lus ou lorsque le caractère "\n" est rencontré. La fonction conserve le "\n" à la fin de la chaîne s'il y a lieu mais dans tous les cas, elle s'assure que la chaîne est bien terminée en insérant le caractère "\0". En cas d'échec, elle retourne NULL. En cas de succès, elle retourne "s" bien remplie.

```
int fputs(char * s, FILE * flux);
```

La fonction "fputs" copie dans le "flux" la chaîne "s" et s'arrête dès qu'elle rencontre le caractère nul "\0". En cas de succès, la fonction retourne le dernier caractère écrit. Elle retourne EOF en cas d'échec.

```
int fgetc(FILE * flux);
```

Si vous connaissez "getchar" alors vous connaissez "fgetc". La fonction récupère un caractère à partir du "flux". En cas de succès, elle retourne le caractère lu. En cas d'échec, elle retourne EOF.

```
int fputc(int c, FILE * flux);
```

Si vous connaissez "putchar" alors vous connaissez "fputc". La fonction écrit le caractère "c" dans le "stream". En cas d'échec, elle retourne EOF autrement elle retourne le caractère écrit.

EXEMPLE EN TROIS SAVEURS

Un fichier appelé "NOMBRE.TXT" contient une série de nombres entiers. Il y en a plusieurs par ligne. On désire en faire une copie avec l'aide des fonctions vues précédemment. Le fichier contenant la copie se nommera "NOMBRE.COP". On peut décliner la solution en plusieurs saveurs. En voici trois.

UTILISATION DU COUPLE FSCANF, FPRINTF

Ici nous effectuons une lecture entier par entier. Le problème de cette solution est évidemment la détection des fins de ligne. Le "fscanf" ne nous permet pas ce raffinement. Il faudrait lire un caractère après chaque entier (avec "%c" par exemple) afin de pouvoir effectuer un test sur le caractère "\n", autrement la copie n'est évidemment pas identique à l'original. Ce n'est pas bien grave. Le fichier peut être récupéré ici: FICHE1.C.

```
int main(void){
   FILE * source;
   FILE * dest;
   int n;

   source = fopen("nombre.txt", "rt");
   if (!source) return 0;

   dest = fopen("nombre.cop", "wt");
   if (!dest) { fclose(source); return 0; }

   while (fscanf(source, "%i", &n) != EOF)
     fprintf(dest, "%i ", n);

   fclose(source);
   fclose(dest);

   return 0;
}
```

UTILISATION DU COUPLE FGETS, FPUTS

Dans cette version, nous effectuons une lecture ligne par ligne. Nous lirons ainsi un maximum de 256 caractères à la fois. Le fichier peut être récupéré ici: FICHE2.C.

```
int main(void){
   const int nbCarParLigne = 256;
   FILE * source;
   FILE * dest;
   char ligne[257];

   source = fopen("nombre.txt", "rt");
   if (!source) return 0;

   dest = fopen("nombre.cop", "wt");
   if (!dest) { fclose(source); return 0; }

   while (fgets(ligne, nbCarParLigne, source))
     fputs(ligne, dest);

   fclose(source);
   fclose(dest);

   return 0;
}
```

UTILISATION DU COUPLE FGETC, FPUTC

Dans cette version, nous effectuons une lecture caractère par caractère. Le traitement est évidemment beaucoup plus long. Le fichier peut être récupéré ici: FICHE3.C.

```
int main(void){
   FILE * source;
   FILE * dest;
   char car;

   source = fopen("nombre.txt", "rt");
   if (!source) return 0;

   dest = fopen("nombre.cop", "wt");
   if (!dest) { fclose(source); return 0; }

   while ((car = fgetc(source)) != EOF)
     fputc(car, dest);

   fclose(source);
   fclose(dest);

   return 0;
}
```

B. Fichiers ouverts en mode binaire

Les fonctions "fread" et "fwrite" caractérisent l'utilisation de ce mode d'ouverture. Quand on dit "ouvert en mode binaire" cela ne veut pas nécessairement dire que le fichier ainsi ouvert (ou créé) sera illisible dans un éditeur de texte. Cela veut simplement dire que nous y récupérons (ou y écrivons) des informations par bloc de N octets sans égard aux valeurs véritables manipulées de la sorte.

Cette manière de procéder rend la manipulation des fichiers en mode binaire extrêmement efficace puisqu'on peut lire de plus gros morceaux à la fois (5Ko par exemple au lieu d'une petite chaîne de caractères de 256 octets...).

Pour bien comprendre les prototypes associés aux fonctions "fread" et "fwrite", il faut savoir que le type "size_t" est en fait un "unsigned int" qui a fait l'objet (dans la librairie "stdio") du "typedef" suivant:

```
typedef unsigned int size_t;
```

```
size_t fread(void * tampon, size_t taille,
             size_t nombre, FILE * flux);
```

La fonction "fread" effectue la lecture d'un bloc de "nombre * taille" octets et place ce bloc dans le "tampon". L'argument "taille" indique donc la taille en octets de chaque élément à lire et l'argument "nombre" indique le nombre d'éléments à lire. Le tampon est

(void *) afin de pouvoir accepter tout type d'arguments (char *, int*, double*, etc.). Elle retourne le NOMBRE D'ÉLÉMENTS (et non d'octets!) effectivement lus. On notera que "fread" lit tout ce qu'elle peut. Elle peut donc lire un nombre d'éléments plus petit ou égal à "nombre". Étant donné que le type "size_t" représente "unsigned int", le plus gros bloc de lecture possible serait donc 64Ko.

```
size_t fwrite(void * tampon, size_t taille,
              size_t nombre, FILE * flux);
```

La fonction "fwrite" effectue l'écriture d'un bloc de "nombre * taille" octets provenant du "tampon". Elle retourne le NOMBRE D'ÉLÉMENTS (et non d'octets!) effectivement insérés dans le "flux". On notera ici aussi que "fwrite" écrit tout ce qu'elle peut. Elle peut donc écrire un nombre d'éléments plus petit ou égal à "nombre".

Les fonctions "ftell" et "fseek" ne sont pas particulières au mode binaire mais on les rencontre plus fréquemment dans ce contexte d'utilisation.

```
long ftell(FILE * flux);
```

Si le fichier est ouvert en mode binaire, la fonction "ftell" retourne la position actuelle exprimée en comptant le nombre d'octets la séparant du début du fichier. En cas d'échec, la fonction retourne -1. Si le fichier est ouvert en mode texte, la position est exprimée en termes de nombre de caractères.

```
int fseek(FILE * flux, long decalage, int base);
```

La fonction "fseek" permet de se déplacer à l'intérieur du fichier. Le décalage indiqué est exprimé en octets. Ce chiffre est évalué relativement à la position indiquée par la "base" qui peut être une des trois valeurs suivantes:

- SEEK_SET: le début du fichier. Dans ce cas, le "decalage" mentionné doit être un nombre positif (ou zéro).
- SEEK_END: la fin du fichier. Dans ce cas, le "decalage" mentionné doit être un nombre négatif (ou zéro).
- SEEK_CUR: la position courante. Dans ce cas, le "decalage" mentionné peut être un nombre positif ou négatif (ou zéro).

La fonction retourne 0 en cas de succès et retourne un nombre différent de zéro en cas d'échec. Il faut cependant noter que le cas d'échec se produit uniquement lorsque le "stream" n'est pas correctement lié. Ainsi, "fseek" peut retourner 0 même si le pointeur dans le fichier n'a pas été déplacé! Pour s'assurer que le déplacement a bel et bien eu lieu, on utilise "ftell" afin de vérifier.

TROIS EXEMPLES EN UNE SEULE SAVEUR!

Nous présentons ici trois petits programmes vous permettant de vous initier aux manipulations de fichiers en mode binaire. Le premier programme imite la commande DOS "TYPE". Il permet donc d'afficher (très rapidement) à l'écran, le contenu d'un fichier texte. Le second programme imite la commande COPY. Finalement, le dernier programme présente quelques déplacements dans un fichier. Les trois programmes peuvent être récupérés ici: MTYPE.C, MCOPY.C, WAZO.C.

IMITATION PARTIELLE DE LA COMMANDE "TYPE"

`#define NB 1024`	la taille du tampon de lecture
`int main(void){`	les paramètres habituels
`  FILE * source;`	
`  char tampon[NB];`	notre tampon de lecture
`  char nom[255];`	nom du fichier source
`  int nb_lu;`	contiendra le nombre d'ÉLÉMENTS lus
OUVERTURE DU FICHIER EN LECTURE ET TRAITEMENT D'ERREUR SUR CELLE-CI	

`  printf("Nom du fichier source : ");`
`  gets(nom);`
`  source = fopen(nom, "rb");`
	Le programme ouvre le fichier spécifié en lecture (r) et en binaire (b).

`  if (!source) {`
`    printf("\nLe fichier source est introuvable");`
`    return 0;`
`  }`
	Petite validation.

FONCTIONNEMENT DE STYLE "commande TYPE"	
`    while(1) {`	on sortira du "while" infini grâce à un break...
`       nb_lu = fread(tampon, 1, NB, source);`	place dans le "buffer", les éléments trouvés dans "source". On essaie de récupérer "NB" éléments chacun nécessitant 1 octet pour le représenter. La variable "nb_lu" contiendra le nombre d'éléments réellement lus. Dans le meilleure des cas, ce nombre = "NB", autrement, il est plus petit que "NB".
`       if (!nb_lu)   break;`	Il n'y avait plus rien à lire. On ferme la source et l'on quitte le programme.
`       fwrite(tampon, 1, nb_lu, stdout);`	On retranscrit le contenu du "buffer" sur "stdout". On retranscrit ainsi très exactement "nb_lu" éléments, chacun ayant une taille de 1 octet.
`       if (nb_lu < NB)   break;` `    }`	C'était les derniers éléments disponibles dans le fichier puisque nous n'avons même pas été en mesure d'en lire "NB".

`    fclose(source);`
`    return 0;`
`}`
	On ferme la source et l'on quitte le programme.

IMITATION PARTIELLE DE LA COMMANDE "COPY"

`#define NB 1024`	la taille du tampon de lecture
`int main(void){`	les paramètres habituels
`  FILE * source;`	
`  FILE * cible;`	
`  char tampon[NB];`	notre tampon de lecture
`  char nomSrc[255];` `  char nomDes[255];`	nom des fichiers
`  int nb_lu;`	contiendra le nombre d'ÉLÉMENTS lus
OUVERTURE DU PREMIER FICHIER EN LECTURE ET TRAITEMENT D'ERREUR SUR CELLE-CI	

`  printf("Nom du fichier source : ");`
`  gets(nomSrc);`
`  source = fopen(nomSrc, "rb");`
	Le programme ouvre le fichier spécifié en lecture (r) et en binaire (b).

`  if (!source) {`
`    printf("\nLe fichier source est introuvable");`
`    return 0;`
`  }`
	Petite validation.

OUVERTURE DU SECOND FICHIER EN ÉCRITURE ET TRAITEMENT D'ERREUR SUR CELLE-CI	

`    printf("Nom du fichier destination : ");`
`    gets(nomDes);`
`    cible = fopen(nomDes, "wb");`
	Le programme ouvre le fichier spécifié en écriture (w) et en binaire (b).

`  if (!cible){`
`    printf("\nLe fichier cible est inutilisable");`
`    fclose(source);`
`    return 0;`
`  }`
	Petite validation sur le résultat. N'oublions surtout pas de fermer la source si la cible n'est pas disponible.

FONCTIONNEMENT DE STYLE "commande COPY "	
`  while(1){`	on sortira du "while" infini grâce à un return...
`    nb_lu = fread(tampon, 1, NB, source);`	idem que précédemment
`    if (!nb_lu)  break;`	Il n'y avait plus rien à lire. On ferme la source ET la cible et l'on quitte le programme.
`    fwrite(tampon, 1, nb_lu, cible);`	On retranscrit le contenu du "buffer" dans la cible. On retranscrit ainsi très exactement "nb_lu" éléments, chacun ayant une taille de 1 octet.
`    if (nb_lu < NB)  break;` `  }`	C'était les derniers éléments disponibles dans le fichier puisque nous n'avons même pas été en mesure dans lire "taille".

`  fclose(source);`
`  fclose(cible);`
`  return 0;`
`}`
	On ferme la source ET la cible et l'on quitte le programme.

DES WAZOS PARTOUT!

Le petit programme qui suit s'amuse à polluer votre fichier texte favori en inscrivant aléatoirement dans le fichier la séquence "WAZO"! Nous avons utilisé un fichier texte afin que vous puissiez examiner tranquillement le résultat de la pollution dans votre éditeur de texte préféré!

`long aleaMinMax(long inf, long sup);`	la fonction nous servira à générer des positions aléatoires dans le fichier à polluer!
`int main(void){`	les paramètres habituels
`  FILE * cible;`	le fichier cible de la pollution
`  char * message = "WAZO";`	le message!
`  char nom[255];`	nom du fichier cible

`  long tailleFichier;`
`  long position;`
`  long NbDeWazo;`
`  long i;`
	respectivement, la taille (en octets) du fichier, la position actuelle dans le fichier, le nombre de "WAZO" que nous insèrerons dans le fichier et un simple compteur de boucle "for".

OUVERTURE DU FICHIER EN LECTURE/ÉCRITURE ET TRAITEMENT D'ERREUR SUR CELLE-CI	

`  printf("Quel est le nom du fichier à polluer ? ");`
`  gets(nom);`
`  cible = fopen(nom, "r+b");`
	Le programme ouvre le fichier spécifié en lecture (r) et en binaire (b). Le "+" indique que les écritures seront possibles.

`  if (!cible) {`
`    printf("\nLe fichier cible est introuvable");`
`    return 0;`
`  }`
	Petite validation.

ON DÉTERMINE LE NOMBRE DE "WAZO" QUI POLLUERONT LE FICHIER...	
`  fseek(cible, 0, SEEK_END);`	on se rend à la toute fin du fichier
`  tailleFichier = ftell(cible);`	on récupère la position où nous sommes donc... la taille en octets
`  NbDeWazo = aleaMinMax(1, tailleFichier/2);`	entre 1 et taille/2 wazo apparaîtront dans le fichier
DÉBUT DE LA POLLUTION DU FICHIER	
`  for (i = 0; i < NbDeWazo; i++) {`	petite boucle toute simple
`    position = aleaMinMax(0, tailleFichier);`	on se choisit une position alétoirement
`    fseek(cible, position, SEEK_SET);`	on s'y rend...
`    printf("position = %li ", ftell(cible));`	j'affiche à l'écran la position choisie... (cela est évidemment optionnel)
`    fwrite(message, 1, 4, cible);`	on inscrit les 4 caractères du message

`  }`
 	
`  printf("Pollution terminée!");`
`  fclose(cible);`
 	
`  return 0;`
`}`
	On ferme la cible et l'on quitte le programme.