TP 4 : Entrées/Sorties, Fichiers, Chaînes de caractères

Objectifs :

- Maîtriser les entrées/sorties au clavier et à l'écran ;
- Savoir manipuler les fichiers de données ;
- Savoir manipuler des chaînes de caractères.

STEPHANIE CHOLLET, Ioannis PARISSIS 2018/2019

Entrées/Sorties

1. Rappel sur les flux

En C, un *flux* est un canal destiné à transmettre ou à recevoir de l'information. Il peut s'agir de périphériques ou de fichiers. La bibliothèque *stdio.h* définit trois types de flux pour les périphériques de communication de base :

- *stdin* pour l'entrée standard ;
- *stdout* pour la sortie standard ;
- *stderr* pour la sortie d'erreur.

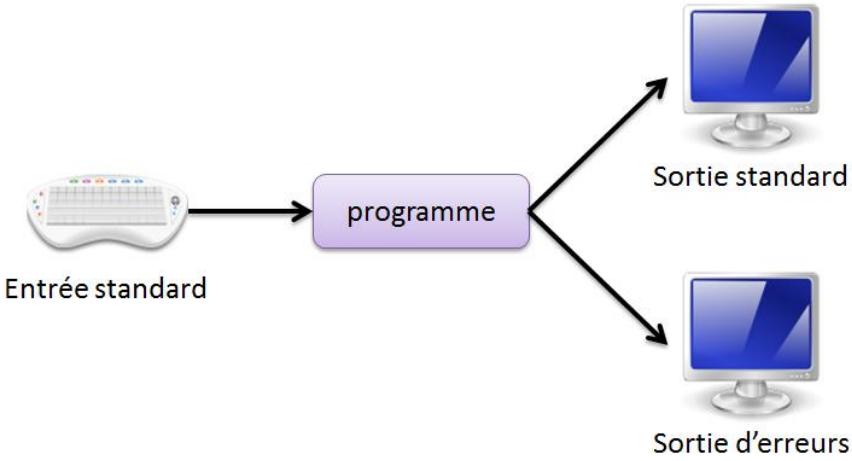

Figure 1 : Les différents flux.

Les fonctions qui sont présentées dans la première partie de ce TP permettent d'écrire et de lire sur ces différents flux. La suite du TP concerne ces mêmes actions mais pour les fichiers.

2. Les fonctions printf() et scanf()

Ces fonctions ont déjà été présentées lors du TP1. Pour rappel, le détail de ces fonctions est donné dans les sections 1.16 et 3.14 du document « Introduction au langage C ».

Ces fonctions nécessitent l'utilisation de la bibliothèque stdio.h. Leurs prototypes sont :

```
int printf(const char * format, liste d'expressions) ;
int scanf(const char * format, liste d'expressions) ;
```

La fonction *printf()* a été prévue pour afficher des caractères, des chaînes de caractères, des numériques ou des pointeurs. La fonction *printf()* fournit en retour le nombre de caractères qu'elle a écrits lorsque l'opération s'est bien déroulée.

La fonction *scanf()* a été prévue pour lire des caractères, des chaînes de caractères, des numériques ou des pointeurs; pour cela, on lui fournit les adresses des variables dans lesquelles on souhaite placer les valeurs lues – c'est la raison de l'utilisation de l'esperluette « & » devant les noms des variables[1]. La fonction *printf()* fournit en retour le nombre de valeurs lues convenablement.

[1] Voir plus loin le TP sur les pointeurs, &a est une expression qui renvoie l'adresse mémoire de la variable a

Formats possibles :

%c	1 caractère
%d	1 entier
%f	1 réel, on peut préciser le nombre de décimales
%e	1 réel en notation exponentielle
%s	1 chaîne de caractères

Exemple : Lecture et écriture

```
#include <stdio.h>

int main(int argc, char * argv[]) {
    int n ;
    printf("Donnez une valeur pour n :") ;
    scanf("%d", &n) ;
    printf("merci pour la valeur %d\n", n) ;
}
```

Les fichiers

Dans cette partie, nous présentons les fonctions principales nécessaires pour la manipulation des fichiers, en particulier pour les fichiers en mode texte. Pour en savoir plus sur les fichiers et la manipulation des fichiers binaires, vous pouvez vous référer au chapitre 5 page 81 du document « *Introduction au langage C* » de B. Cassagne.

Les fonctions présentées nécessitent l'utilisation de la bibliothèque *stdio.h*. Il faut donc l'inclure avec la directive `#include <stdio.h>`.

1. Ouverture d'un fichier : fopen()

Le rôle de la fonction *fopen()* est d'ouvrir le fichier concerné dans *le mode* indiqué. Le prototype de la fonction est le suivant :

```
FILE * fopen(const char * nomFichier, const char * mode) ;
```

Elle fournit comme valeur de retour un flux (pointeur sur une structure de type prédéfini FILE) qui sert à toutes les opérations de lecture ou écriture dans le fichier. En cas d'échec, un pointeur null est retourné. Les causes d'un tel échec peuvent être :

- le fichier est inexistant, en cas d'ouverture en lecture ou en mise à jour ;
- on ne dispose pas des droits d'accès voulus au fichier ou au répertoire ;
- …

La valeur de mode fait intervenir trois sortes d'indicateurs qu'il est possible de combiner.

Type d'indicateur	Valeurs	Signification
Principal (obligatoire)	r	Ouverture en lecture seule d'un fichier existant.
	w	Ouverture en écriture seule dans un nouveau fichier ou **écrasement** d'un fichier existant.

	a	Ouverture en écriture à la fin d'un fichier existant ou d'un nouveau.
Mode d'ouverture (optionnel)	b	Manipulation d'un fichier binaire. En l'absence de cet indicateur, manipulation d'un fichier texte.
Mise à jour (optionnel)	+	Autorise à la fois la lecture et l'écriture.

Exemple : Ouverture d'un fichier texte « monFichier.txt » en lecture/écriture.

```
FILE * fic = fopen("monFichier.txt", "r+") ;
```

Combinaisons possibles des indicateurs :

r	Ouverture d'un fichier texte en lecture	**rb**	Ouverture d'un fichier binaire en lecture
w	Ouverture d'un fichier texte en écriture	**wb**	Ouverture d'un fichier binaire en écriture
a	Ouverture d'un fichier texte en écriture à la fin	**ab**	Ouverture d'un fichier binaire en écriture à la fin
r+	Ouverture d'un fichier texte en lecture/écriture	**rb+**	Ouverture d'un fichier binaire en lecture/écriture
w+	Ouverture d'un fichier texte en lecture/écriture	**wb+**	Ouverture d'un fichier binaire en lecture/écriture
a+	Ouverture d'un fichier texte en lecture/écriture à la fin	**ab+**	Ouverture d'un fichier binaire en lecture/écriture à la fin

2. Fermeture d'un fichier : fclose()

La fonction *fclose()* permet de fermer un fichier. Son prototype est le suivant :

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

Exemple : Fermeture du fichier « *monFichier.txt* » qui a été ouvert dans l'exemple précédent.

```
fclose(fic) ;
```

3. Ecriture dans un fichier en mode texte : fprintf()

Pour écrire dans un fichier de façon formatée, il faut utiliser la fonction *fprintf()* qui est, comme la fonction *printf()*, une fonction à arguments variables. La fonction *fprintf()* prend en premier argument le flux dans lequel le texte sera enregistré, les arguments suivants sont les mêmes que ceux de la fonction *printf()*.

Exemple : Ecrire les 10 premiers entiers dans le fichier « monFichier.txt » qui a été ouvert dans l'exemple précédent.

```
for(i = 0 ; i < 10 ; i++) {
    fprintf(fic, "Valeur = %d\n", i) ;
}
```

4. Lecture dans un fichier en mode texte : fscanf()

La lecture dans un fichier se fait avec la fonction *fscanf()* qui est, comme la fonction *scanf()*, une fonction à arguments variables. La fonction *fscanf()* prend en premier argument le

fichier dans lequel la lecture sera faite, les arguments suivants sont les mêmes que ceux de la fonction *scanf()*.

La lecture dans un fichier nécessite de détecter la fin du fichier. La fonction *feof()* prend la valeur vrai (non nul) lorsque la fin du fichier a été atteinte. Son prototype est le suivant :

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

Il faut noter que la fonction feof() ne travaille pas par anticipation, ce qui signifie qu'il n'est pas suffisant d'avoir lu le dernier octet du fichier pour que cette condition prenne la valeur vrai. Il est nécessaire d'avoir tenté de lire au-delà.

Exemple : Lecture d'entiers dans un fichier :

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char * argv[]) {
    FILE * fic;
    int nb;
    /* Ouverture du fichier */
    fic = fopen("monFichier.txt", "r");
    if(fic == NULL) {
          perror("Probleme ouverture fichier monFichier.txt");
          exit(1);
    }

    /* Lecture dans le fichier */
    while(!feof(fic)) {
          fscanf(fic, "%d", &nb);
          printf("Valeur lue = %d\n", nb) ;
    }

    /* Fermeture du fichier */
    fclose(fic);

    return 0;
}
```

Remarque : Les fonctions ci-dessus fonctionnent également sur les flux d'entrée/sortie introduits dans la section 1. Il suffit de passer en premier paramètre stdin, stdout ou stderr (par exemple `fprintf(stdout, "Valeur = %d\n", i) ;`).

Exercice

Exercice 1 : Manipulation de fichier et tri de tableau

Réaliser un module qui permet de lire dans un fichier une suite d'entiers puis de les trier avant de les enregistrer dans un autre fichier.

Dans un premier temps, préparer plusieurs tests sous la forme de plusieurs fichiers texte contenant des suites d'entiers différentes.

Ecrire les fonctions suivantes :

1. `int lireDonnees(char nomFichier[], int T[])` qui lit les données dans le fichier nommé *nomFichier* des entiers, puis les stocke dans un tableau *T*. La valeur de retour est le nombre d'entiers qui ont été lus (c'est-à-dire le nombre d'éléments utiles du tableau). On suppose la taille du tableau suffisante pour contenir tous les entiers.
2. `void afficherTableau(int T[], int nb)` qui affiche le contenu du tableau *T* qui comprend *nb* éléments.
3. `void triABulles(int T[], int nb)` qui trie le tableau *T* de *nb* éléments avec la méthode du tri à bulles[2] et qui retourne le tableau trié.
4. `void enregistrerDonnees(char nomFichier[], int T[], int nb)` qui enregistre les *nb* valeurs du tableau *T* dans le fichier nommé *nomFichier*.

Pour tester vos fonctions, réaliser un programme principal. L'exécution de ce programme doit prendre en paramètre le nom du fichier pour la lecture des données ainsi que celui pour l'enregistrement des valeurs.

Conserver chaque fichier de données d'entrée créé pour le test. Préciser, dans un fichier à part (explicationsTest.txt), le choix des données d'entrée fait et pourquoi celles-là testent correctement la fonction de tri.

Exemple d'exécution :

```
machine@debian$./progTri donnees.txt resultat.txt
```

[2] Le principe du tri à bulles est le suivant : on imagine que le tableau est vertical, le plus petit indice en haut, et que les éléments les plus petits sont "moins lourds" que les autres. On effectue des passages successifs sur le tableau de bas en haut. À chaque étape, si deux éléments sont en ordre inverse (plus "léger" dessous), on les inverse. Les éléments "légers" remontent comme des bulles vers la surface tandis que les éléments "plus lourds" tombent lentement vers le fond. L'algorithme se termine après n-1 passages.

Les chaînes de caractères

En langage C, on fait la distinction entre un tableau de caractères et une "chaîne de caractères" : une chaîne de caractères est contenue dans un tableau de caractères; par convention, sa fin est indiquée par le caractère particulier '\0' (caractère nul, différent de '0'). Le respect de cette convention est notamment indispensable si l'on souhaite utiliser les fonctions de la librairie string.h.

1. Déclaration et initialisation d'une chaîne de caractères

Une chaîne de caractères peut être allouée de manière statique[3] comme nous l'avons vu avec les tableaux :

```
char ch[TAILLE]
```

D'après cette déclaration, ce tableau peut contenir une chaîne **d'au plus TAILLE-1** caractères (plus le caractère '\0').

L'initialisation d'une chaîne de caractères peut se faire au moment de sa déclaration de deux façons différentes : soit comme une constante chaîne soit par énumération des caractères.

Exemple :

```
char ch[11] = "bonjour" ;
char ch[11] = {'b', 'o', 'n', 'j', 'o', 'u', 'r', '\0'} ;
```

Ces deux déclarations sont équivalentes. Pour la seconde, il est important de noter qu'il faut ajouter le caractère **'\0'** à la fin alors qu'il est ajouté automatiquement dans la première déclaration. Le résultat de cette initialisation est le suivant :

b	o	n	j	o	u	r	\0			
0	1	2	3	4	5	6	7	8	9	10

2. Modification du contenu d'une chaîne de caractères

Une chaîne de caractères peut être modifiée caractère par caractère (comme les cellules d'un tableau).

Exemple : Transformer la chaîne contenue dans `ch` de « bonjour » en « bonsoir ».

```
ch[3] = 's';
ch[5] = 'i';
```

3. Manipulation de chaînes de caractères

La bibliothèque `string.h` contient un ensemble de fonctions permettant de manipuler des chaînes de caractères :

- la fonction *strlen()* qui renvoie la longueur d'une chaîne de caractères ;
- la fonction *strcpy()* qui permet de copier une chaîne de caractères dans une autre ;
- la fonction *strstr()* qui permet de chercher la première occurrence d'un caractère dans une chaîne
- …

[3] Il est également possible de faire une allocation dynamique (cf. TP 7)

Exercice

Exercice 2 : Manipulation de chaînes de caractères

Réaliser un module offrant les fonctions suivantes. Pour chaque fonction, préciser au préalable les tests utilisés.

1. Ecrire deux fonctions *myStrlen()* et *myStrcpy()* qui réalisent le même travail que *strlen()* et *strcpy()* – on peut trouver la spécification de ces dernières dans *man*.
2. Ecrire une fonction *affichagerEnHexadecimal()* qui affiche à l'écran le mot en utilisant la valeur hexadécimale de chacune des lettres. Cette fonction prend en entrée un mot.
3. Ecrire une fonction *affichagerEnDecimal()* qui affiche à l'écran le mot en utilisant la valeur décimale de chacune des lettres. Cette fonction prend en entrée un mot.
4. Ecrire une fonction *mettreEnMajuscule()* qui transforme les caractères en minuscule en caractères en majuscule. Les autres caractères restent inchangés. Cette fonction prend en entrée une chaîne de caractères en minuscule et retourne une chaîne en majuscule.
5. Ecrire une fonction *mettreEnMinuscule()* qui transforme les caractères en majuscule en caractères en minuscule. Les autres caractères restent inchangés. Cette fonction prend en entrée une chaîne de caractères en majuscule et retourne une chaîne en minuscule.
6. Ecrire une fonction *transformerMinMaj()* qui change la casse des caractères. Les caractères en minuscule passent en majuscule et inversement. Cette fonction prend en paramètre une chaîne de caractères et retourne la chaîne de caractères modifiée.
7. Ecrire une fonction *retournerMot()* qui renvoie un mot écrit à l'envers (exemple : oiseau\0 devient uaesio\0). Cette fonction prend en entrée une chaîne de caractères et retourne la chaîne de caractères modifiée.

   Les fonctions suivantes effectuent des recherches dans une chaîne de caractères. Le schéma classique d'une itération de recherche est le suivant (et il faudra l'appliquer) : while (<il reste des éléments à examiner> && <non trouvé>) {<passer à l'élément suivant>},.
8. Ecrire une fonction *rechercherCaractereG()* qui renvoie la première occurrence d'un caractère dans une chaîne de caractères en partant de la gauche. Cette fonction prend en entrée la chaîne de caractères ainsi que le caractère recherché. Elle retourne la position dans la chaîne si le caractère est présent, -1 si le caractère n'a pas été trouvé.
9. Ecrire une fonction *rechercherCaractereD()* qui renvoie la première occurrence d'un caractère dans une chaîne de caractères en partant de la droite. Cette fonction prend en entrée la chaîne de caractères ainsi que le caractère recherché. Elle retourne la position dans la chaîne si le caractère est présent, -1 si le caractère n'a pas été trouvé.
10. Ecrire une fonction *estPalindrome()* qui détermine si le mot donné en argument est un palindrome. Cette fonction retourne la valeur 0 si c'est le cas, -1 sinon.

11. Ecrire une fonction *comparerChaine()* qui permet d'effectuer une comparaison de deux chaînes. Elle prend en paramètre d'entrée deux chaînes de caractères et elle retourne :
    - la valeur 0 en cas d'égalité : toto == toto ;
    - une valeur positive si la première chaîne de caractères est supérieure alphabétiquement à la seconde : toto > titi ;
    - une valeur négative si la première chaîne de caractères est inférieure alphabétiquement à la seconde : titi < toto.