C

Ch 6 – Fichiers

I. INTRODUCTION 1

A. FLUX D'ENTREES-SORTIES 1
1. Flux d'Entrées-sorties clavier/écran 1
2. Flux d'Entrées-sorties disque : les fichiers 2
B. LIBRAIRIE C POUR LES FLUX D'ENTREES-SORTIES FICHIERS 2

II. FICHIERS TEXTES 2

A. FLUX ET FONCTION DE GESTION DES FICHIERS TEXTES 2
B. OUVERTURE ET FERMETURE D'UN FICHIER TEXTE 2
C. ECRITURE D'UN FICHIER : SORTIE FORMATEE AVEC FPRINTF 3
D. LECTURE D'UN FICHIER : LECTURE FORMATEE AVEC FSCANF 3
E. ECRITURE D'UN FICHIER PAR LIGNE COMPLETES : FPUTS 4
F. LECTURE D'UN FICHIER PAR LIGNE COMPLETES : FGETS 4

III. FICHIERS BINAIRES 5

A. FLUX ET FONCTION DE GESTION DES FICHIERS BINAIRE 5
B. OUVERTURE ET FERMETURE D'UN FICHIER BINAIRE 5
C. ECRITURE D'UN FICHIER BINAIRE : FWRITE 5
D. LECTURE D'UN FICHIER BINAIRE : LECTURE AVEC FREAD 6
E. SE POSITIONNER DANS UN FICHIER BINAIRE : FSEEK 7

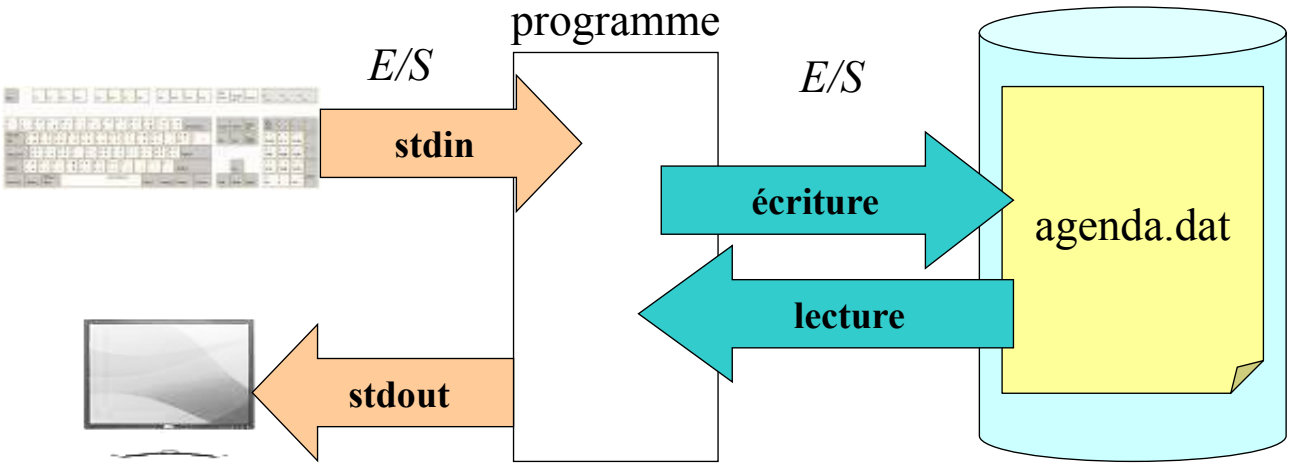
IV. GERER LES ERREURS D'ENTREES/SORTIES 7

A. TEST A L'OUVERTURE D'UN FICHIER 7
B. VALEUR DE RETOUR DES FONCTIONS D'E/S 8

I. Introduction

A. Flux d'Entrées-Sorties

Les flux (*anglais : stream*) correspondent aux échanges réalisés entre un programme et les périphériques connectés à l'ordinateur : clavier, écran, disque dur, etc.



1. Flux d'Entrées-sorties clavier/écran

Les entrées-sorties sont toujours réalisées comme des flots de données (flux) qu'on peut symboliser comme des canaux entre le programme en mémoire et les différents périphériques :

- Clavier → programme : flux d'entrée **stdin** (entrée standard)
  - Lecture à partir du clavier et réception par une variable
- Programme → écran : flux de sortie **stdout** (sortie standard)
  - Ecriture vers l'écran

2. Flux d'Entrées-sorties disque : les fichiers

Les fichiers permettent la conservation des informations saisies de manière permanente (*on parle de persistance des informations*) sur un support magnétique afin de pouvoir les réutiliser ultérieurement.
Comme la lecture du clavier et l'écriture vers l'écran, les flux disques vont également dans les 2 sens :

- Disque dur → programme : flux d'entrée
- Programme → disque dur : flux de sortie

L'utilisation d'un fichier comporte en général 3 phases :

- L'ouverture d'un flux selon un certain mode (lecture, écriture, etc)
- La lecture ou l'écriture selon le mode
- La fermeture du flux.

B. Librairie C pour les Flux d'Entrées-Sorties fichiers

La bibliothèque <stdio> met à disposition du programmeur un ensemble de fonctions permettant la gestion des entrées/sorties vers les fichiers. (inclure le fichier d'en-tête **<stdio.h >)**

II. Fichiers textes

Les fichiers textes sont constitués uniquement de caractères ASCII visibles (mis à part les caractères de fin de ligne : CR LF (Carriage Return, Line Feed), OD OA en hexadécimal.
La longueur de chaque ligne dépend de la longueur des valeurs qui y ont été enregistrées. Chaque valeur est séparée de la suivante par un caractère séparateur (espace, point-virgule) qui va permettre la relecture.
Un fichier texte peut être ouvert par un éditeur de texte : mais attention aux modifications qui pourraient nuire à la relecture du fichier par le programme.

A. Flux et fonction de gestion des fichiers textes

La bibliothèque définit le type FILE.

```
FILE * fichier;
```

Déclare la variable « fichier » de type « pointeur vers le type FILE »

B. Ouverture et fermeture d'un fichier texte

Syntaxe:

```
fichier = fopen(nom_fichier, mode) ; // ouverture
```

- **fichier:** nom logique attribué au flux (pointeur vers le type FICHIER)
- **nom_ fichier :** chaine de caractères contenant le nom du fichier physique (chemin d'accès, nom, extension)
- **mode :** mode d'ouverture du fichier, parmi
  - **« r » :** en lecture
  - **« w » :** en écriture
  - **« a » :** en ajout

ATTENTION : l'ouverture d'un flux de sortie créée le fichier s'il n'existe pas, MAIS peut l'écraser s'il existe déjà.

Pour fermer le flux :

```
fclose(fichier) ;
```

- **fichier :** nom logique

C. Ecriture d'un fichier : sortie formatée avec fprintf

La fonction « fprintf» permet, comme la fonction « printf », d'écriture des données dans un flux de sortie.

Syntaxe:

```
fprintf(fichier, «chaine de contrôle », expressions) ;
```

Exemple :

```
/* la procedure pEcrire
void pEcrire(char nomfic[NBCAR]) {
  /* declarations */
  FILE * f;
  char nom[24], prenom[24];
  /* initialisations */
  f = fopen(nomfic,"w"); // ouvrir le fichier en écriture
  printf("\nSaisir nom et prenom séparés par une espace :\n");
  scanf("%s %s",nom,prenom);
  /* traitement */
  while (nom[0] != 'z' && prenom[0] != 'z') {
    fprintf(f,"%s %s\n", nom, prenom);  // ecriture fichier
    scanf("%s %s",nom,prenom); // lecture d'un couple suivant
  }
  /* finalisation */
  fclose(f);
}
```

```
1  tim burtonCRLF
2  john goudaleCRLF
3
```

D. Lecture d'un fichier : lecture formatée avec fscanf

Syntaxe:

```
fscanf(fichier, «chaine de contrôle », variables) ;
```

Exemple :

```
/* lire un fichier jusqu'à la fin et afficher les données */
void pLire(char nomfic[NBCAR]) {
    /* declarations */
    FILE * f;
    char nom[NBCAR], prenom[NBCAR];
    /* initialisations */
    f = fopen(nomfic,"r");
    fscanf(f,"%s %s",nom,prenom); // lecture 1ere enregistrement
    /* traitement/resultats */
    while (!feof(f)) { // tant qu'on n'est pas à la fin du fichier
        printf("\nlu : %s %s", nom, prenom);
```

```
        fscanf(f,"%s %s",nom,prenom); // lecture suivant
    }
    /* finalisation */
    fclose(f);
}
```

E. Ecriture d'un fichier par ligne complètes : fputs

La fonction « fputs» permet l'écriture d'une ligne complète.

Syntaxe:

```
fputs(variable_chaine, fichier) ;
```

Exemple :

```
#define NBCAR 200

/* la procedure pEcrire
void pEcrire(char nomfic[NBCAR]) {
  /* declarations */
  FILE * f;
  char ligne[200];
  /* initialisations */
  f = fopen(nomfic,"w"); // ouvrir le fichier en écriture
  printf("\nSaisir un texte :\n");
  fgets(ligne,NBCAR,stdin); // lecture flux stdin avec fgets
  /* traitement */
  while (ligne[0] != 'z') {
    fputs(ligne, f);  // ecriture fichier
    fgets(ligne,NBCAR,stdin);  // lecture suivant
  }
  /* finalisation */
  fclose(f);
}
```

F. Lecture d'un fichier par ligne complètes : fgets

La fonction « fgets» permet la lecture d'une ligne complète.

Syntaxe:

```
fgets(variable_chaine, taille, fichier) ;
```

Exemple :

```
/* la procedure pLire
void pLire(char nomfic[NBCAR]) {
    /* declarations */
    FILE * f;
    char ligne[200];
    /* initialisations */
    f = fopen(nomfic,"r");
    fgets(ligne,NBCAR,f); // lecture  premier
    /* traitement/resultats */
    while (!feof(f)) { // tant qu'on n'est pas à la fin
        puts(ligne);
        fgets(ligne,NBCAR,f); // lecture  suivant
```

```
    }
    /* finalisation */
    fclose(f);
}
```

III.Fichiers binaires

Alors que les fichiers textes sont constitués uniquement de caractères ASCII visibles, les fichiers binaires sont codés en fonction du type de donnée : les chaînes de caractères seront codées en ASCII, les nombres entiers en binaire, les nombres réels en binaire au format IEEE-754.
Contrairement aux fichiers textes, leur contenu n'est généralement pas lisible directement.

Les fichiers binaires sont adaptés au stockage des données structurées. Mais la longueur du type doit être fixe et connue à l'avance.

Un **enregistrement** correspond aux données relatives à la structure gérée (un bloc de données de longueur fixe). Un fichier binaire comporte autant d'enregistrements que de structures stockées. Chaque enregistrement peut être accédé directement à partir de sa position dans le fichier.

A. Flux et fonction de gestion des fichiers binaire

La bibliothèque définit le type FILE.

```
FILE * fichier;
```

Déclare la variable « fichier » de type « pointeur vers le type FILE »

B. Ouverture et fermeture d'un fichier binaire

Syntaxe:

```
fichier = fopen(nom_fichier, mode) ; // ouverture
```

- **fichier:** nom logique attribué au flux (pointeur vers le type FICHIER)
- **nom_ fichier :** chaine de caractères contenant le nom du fichier physique (chemin d'accès, nom, extension)
- **mode :** mode d'ouverture du fichier, parmi
  - **« rb » :** en lecture d'un fichier binaire
  - **« wb » :** en écriture d'un fichier binaire
  - **« ab » :** en ajout d'un fichier binaire

ATTENTION : l'ouverture d'un flux de sortie créée le fichier s'il n'existe pas, MAIS peut l'écraser s'il existe déjà.

Pour fermer le flux :

```
fclose(fichier) ;
```

- **fichier :** nom logique

C. Ecriture d'un fichier binaire : fwrite

Syntaxe:

```
fwrite(enregistrement, longueur, nbre ecriture,fichier) ;
```

Exemple :

```
#define NBCAR 16

/* DECLARATIONS GLOBALES */
struct strPersonne {
    char nom[NBCAR];
    char prenom[NBCAR];
};
typedef struct strPersonne personne;
. . .
/* écrire dans un fichier binaire */
void pEcrire(char nomfic[NBCAR]) {
    /* declarations */
    FILE * f;
    personne unePersonne;
    /* initialisations */
    f = fopen(nomfic,"wb"); // ouvrir le fichier en écriture
    printf("\nSaisir nom et prenom séparés par un espace :\n");
    scanf("%s %s",unePersonne.nom,unePersonne.prenom); // lecture1
    /* traitement */
    while  (unePersonne.nom[0]!='z'  &&  unePersonne.prenom[0]!='z')
{
        fwrite(&unePersonne, sizeof(unePersonne), 1, f); // ecrire
        scanf("%s %s",unePersonne.nom,unePersonne.prenom);
    }
    /* finalisation */
    fclose(f);
}
```

```
1  timNUL...burtonNUL...johnNUL...goudaleNUL...
```

D. Lecture d'un fichier binaire : lecture avec fread

La lecture d'un fichier avec **fread** nécessite la connaissance précise de la structure et l'encodage des données utilisées pour l'écriture.

Syntaxe:

```
fread(enregistrement, longueur, nbre lecture,fichier) ;
```

Exemple :

```
/* lire un fichier binaire */
void pLire(char nomfic[NBCAR]) {
    /* declarations */
    FILE * f;
    personne unePersonne;
    /* initialisations */
    f = fopen(nomfic,"rb");
    fread(&unePersonne, sizeof(unePersonne),1,f); // lecture 1
    /* traitement/resultats */
    while (!feof(f)) { // tant qu'on n'est pas à la fin
        printf("\nlu      :      %s      %s",      unePersonne.nom,
unePersonne.prenom);
        fread(&unePersonne, sizeof(unePersonne),1,f);// suivant
    }
    /* finalisation */
```

```
    fclose(f);
}
```

E. Se positionner dans un fichier binaire : fseek

La fonction **fseek** positionne le curseur de lecture à un certain endroit du fichier.

Syntaxe:

```
fseek(fichier, déplacement (offset), origine) ;
```

où :

- **fichier** est la variable fichier
- **déplacement** en nombre d'octets à partir de l'origine spécifié ci-dessous
- **origine** parmi SEEK_SET (début de fichier), SEEK_CUR (position courante), SEEK_END (fin du fichier)

Exemple : lecture du dernier enregistrement

```
void pLireDernier(char nomfic[NBCAR]) {
    /* declarations */
    FILE * f;
    personne unePersonne;
    /* initialisations */
    f = fopen(nomfic,"rb");

    if (fseek(f,-1*sizeof(unePersonne),SEEK_END) == 0) {
        fread(&unePersonne, sizeof(unePersonne),1,f);
        printf("\nlu      :      %s      %s",      unePersonne.nom,
unePersonne.prenom);
    }
    else {
        printf("enregistrement inaccessible (fichier vide)");
    }
    /* finalisation */
    fclose(f);
}
```

IV. Gérer les erreurs d'entrées/sorties

Les entrées-sorties dépendent d'éléments extérieurs : ceux-ci ne sont pas forcément ceux attendus et ces fonctions peuvent échouer. Il est alors indispensable de savoir si une opération s'est bien déroulée ou pas.

A. Test à l'ouverture d'un fichier

Le premier point de contrôle se situe à l'ouverture du fichier. Après l'ouverture, la variable fichier a été affectée d'une valeur[1] . Si cette valeur est nulle, cela signifie que ce fichier n'a pas été ouverte correctement.

Exemple :

```
/* lire un fichier jusqu'à la fin et afficher les données */
void pLire(char nomfic[NBCAR]) {
    /* declarations */
    FILE * f;
    char nom[NBCAR], prenom[NBCAR];
```

[1] La valeur retournée par l'instruction fopen est en un pointeur vers l'adresse de la zone de communication avec le fichier ouvert

```
    /* initialisations */
    f = fopen(nomfic,"r");
     if (f == NULL) {
         printf("erreur d'ouverture") ;
         return ; // cf. remarque plus bas
     }
    fscanf(f,"%s %s",nom,prenom); // lecture 1ere enregistrement
    /* traitement/resultats */
    while (!feof(f)) { // tant qu'on n'est pas à la fin du fichier
        printf("\nlu : %s %s", nom, prenom);
        fscanf(f,"%s %s",nom,prenom); // lecture suivant
    }
    /* finalisation */
    fclose(f);
}
```

Remarque : dans ce cas, on quitte la fonction et on retourne au point d'appel sans que la fonction appelante puisse savoir si pLire s'est bien déroulée ou pas.

B. Valeur de retour des fonctions d'E/S

Les fonctions retournent généralement une valeur qui permet de savoir si celle-ci s'est bien déroulée ou pas.

Par exemple pour la fonction **fscanf** (identique pour les fonctions scanf, fscanf, sscanf, vscanf, vsscanf, vfscanf) : ces fonctions retournent le nombre d'éléments lus de manières correcte et affectés (celui-ci pouvant être inférieur à celui prévu). Une valeur **EOF** (constante fournie) est renvoyée si la fin de l'entrée est atteinte avant que la 1ère conversion ait pu avoir lieu ou si une correspondance échoue. **EOF** est aussi renvoyé en cas d'erreur de lecture auquel cas une valeur d'erreur est affectée à **errno**.

Exemple :

```
void pLire(char nomfic[NBCAR]) {
    /* declarations */
    FILE * f;
     int res;
    char nom[NBCAR], prenom[NBCAR], autre[NBCAR];
    /* initialisations */
    f = fopen(nomfic,"r");
```

➔ Manque la vérification de l'ouverture

```
    res = fscanf(f,"%s %s",nom,prenom); // lecture 1
    if (res != 2) { // 2 valeurs lues
         printf("pb lecture : %d %d",errno, res);
         return;
    }
    /* traitement/resultats */
    while (!feof(f)) { // tant qu'on n'est pas à la fin
        printf("\nlu : %s %s", nom, prenom);
        fscanf(f,"%s %s",nom,prenom); // lecture suivant
        if (res != 2) { printf("pb lecture");}
     }
    /* finalisation */
    fclose(f); }
```