# Langage de programmation C :

# Les Fichiers

# 1) Introduction, définition et propriétés, mémoire tampon

- Le langage C offre la possibilité de lire et d'écrire des données dans un fichier. Pour des raisons d'efficacité, les accès à un fichier se font par l'intermédiaire d'une mémoire-tampon (on parle de buffer), ce qui permet de réduire le nombre d'accès aux périphériques (disque...).
- Pour pouvoir manipuler un fichier, un programme a besoin d'un certain nombre d'informations : l'adresse de l'endroit de la mémoire-tampon où se trouve le fichier, la position de la tête de lecture, le mode d'accès au fichier (lecture ou écriture)
- Ces informations sont rassemblées dans une structure, dont le type est **FILE *.** Ce type est défini dans <stdio.h>.
- Un objet de type FILE * est appelé flot de données (stream en anglais).

## Définition et propriétés

- Un fichier est une suite de données conservées en permanence sur un support externe(disquette, disque dur,…). Ces données regroupent, le plus souvent, plusieurs composantes(champs) d'une structure.
- En C, les fichiers sont considérés comme une suite d'octets(1 octet=caractère).
- Principe de manipulation d'un fichier:
  - Ouverture du fichier
  - Lecture, écriture et déplacement dans le fichier
  - Fermeture du fichier
- Deux techniques pour manipuler un fichier
  - **L'accès séquentiel** : pour atteindre l'information souhaité, il faut passer par la première puis la deuxième et ainsi de suite.
  - L'accès direct : consiste à se déplacer directement sur l'information souhaité sans avoir à parcourir celles qui la précédent.

# Mémoire tampon

- L'accès au fichier se fait par l'intermédiaire d'une zone mémoire de stockage appelé mémoire tampon(buffer).
- Le buffer une zone de la mémoire centrale qui stocke une quantité, assez importante, de données du fichier.
- Son rôle est d'accélérer les entrées/sorties à un fichier.

# 2) Types de Fichiers

- **Fichier de texte** : est une suite de lignes, chaque ligne est une suite de caractères terminée par le caractère spécial '\n'.
- **Fichier binaire** : est une suite d'octets pouvant représenter toutes sortes de données.
- Fichiers standards : se sont des fichiers spéciaux prédéfinis qui s'ouvrent automatiquement lorsqu'un programme commence à s'exécuter :
  - Stdin (standard input): entrée standard(par défaut, lié au clavier)
  - Stdout (standard output): sortie standard(par défaut, lié à l'écran)
  - Stderr stderr (standard error) : affichage des messages d'erreur (par défaut, à l'écran).

# 3)Déclaration d'un fichier, ouverture et fermeture de fichiers

**FILE** ***<pointeurfichier>;**

- Le type File est défini dans <stdio.h> en tant que structure. A l'ouverture d'un fichier, la structure FILE contient un certain nombre d'informations sur ce fichier telles que :
  - Adresse de la mémoire tampon,
  - Position actuelle dans le tampon,
  - Nombre de caractères déjà écrits dans le tampon,
  - Type d'ouverture du fichier,…
- Pour travailler avec un fichier dans un programme, il faut ranger l'adresse de la structure FILE dans le pointeur de fichier et tout accès ultérieur au fichier se fait par l'intermédiaire de ce pointeur.

# Ouverture de fichiers

- Lorsqu'on désire accéder à un fichier, il est nécessaire avant tout accès d'ouvrir le fichier à l'aide de la fonction fopen.
- Cette fonction, de type **FILE *** ouvre un fichier et lui associe un flot de données. **Sa syntaxe est** :

```
fopen("nom-de-fichier","mode");
```

- **Sémantique des paramètres**
  - Le premier argument de fopen fournit donc le nom du fichier.
  - Le second argument, mode, est une chaîne de caractères qui spécifie le mode d'accès au fichier.

# Ouverture de fichiers

- Les différents modes d'accès sont
  - "r" ouverture d'un fichier texte en lecture
  - "w" ouverture d'un fichier texte en écriture
  - "a" ouverture d'un fichier texte en écriture à la fin
  - "r+" ouverture d'un fichier texte en lecture/écriture
  - "w+" ouverture d'un fichier texte en lecture/écriture
  - "a+" ouverture d'un fichier texte en lecture/écriture à la fin

# Ouverture de fichiers

- Conditions particulières et cas d'erreur
  - Si le mode contient la lettre r, le fichier doit exister, sinon c'est une erreur(la fonction fopen retourne NULL).
  - Si le mode contient la lettre w, le fichier peut ne pas exister. Dans ce cas, il sera créé, et si le fichier existait déjà, son ancien contenu est perdu.
  - Si le mode contient la lettre a, le fichier peut ne pas exister. Comme pour le cas précédent, si le fichier n'existe pas, il est créé ; si le fichier existe déjà, son ancien contenu est conservé.
  - Si un fichier est ouvert en mode "écriture à la fin" toutes les écritures se font à l'endroit qui est était la fin du fichier lors de l'exécution de l'ordre d'écriture. Cela signifie que si plusieurs processus partagent le même FILE*, résultat de l'ouverture d'un fichier en écriture à la fin, leurs écritures ne s'écraseront pas mutuellement.

# Ouverture de fichiers

- **Utilisation typique de fopen**

```
#include <stdio.h>
FILE *fp;
.....
if ((fp = fopen("donnees.txt","r")) == NULL)
    {
      fprintf(stderr,"Impossible d'ouvrir le fichier données
  en
       lecture\n");
       exit(1);
    }
```

# Fermeture de fichiers : la fonction fclose

- La fonction fclose permet de terminer la manipulation d'un fichier ouvert par la fonction fopen.
- Sa syntaxe est :
  - **fclose**(nom_du_fichier)
- où nom_du_fichier est de type FILE* retourné par la fonction fopen correspondante.
- La fonction fclose retourne
  - 0 si l'opération s'est déroulée normalement et
  - EOF si il y a eu une erreur.
- Exemple :

```
FILE *fp;
fp = fopen("donnees.txt","r")
fclose(fp);
```

# 4)La fonction d'écriture en fichier: fprintf

- La fonction fprintf, analogue à printf, permet d'écrire des données dans un flot.
- Sa syntaxe est :
  - fprintf(nom_du_fichier, "format", **expression$_1$, . . . , expression$_n$**);
- où nom_du_fichier est le flot retourné par la fonction fopen.
- Les spécifications de format utilisées pour la fonction fprintf sont les mêmes que pour printf
- Puisque
  - printf(...) ⇐⇒ fprintf(stdout,...)
- Expression$_i$ est une expression délivrant une valeur à écrire.

# 5) La fonction de saisie en fichier : fscanf

- La fonction fscanf, analogue à scanf, permet de lire des données dans un fichier.
- Sa syntaxe est semblable à celle de scanf :
  - fscanf(nom_du_fichier, "format", **adresse$_1$, . . . , adresse$_n$**);
- où nom_du_fichier est le flot de données retourné par fopen.
- Les spécifications de format sont ici les mêmes que celles de la fonction scanf.
- **Adresse$_i$ :** adresse des variables à affecter à partir des donner.
- Un format et une adresse doivent être fournis pour chaque variable.

# Exemple

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

int main() {
    FILE *pf;
    int i = 100;
    char c = 'c';
    float t = 1.234;

    pf = fopen("test.txt", "w+");  // ouverture et mise à jour
    if (pf == NULL) {
        printf("Erreur d'ouverture du fichier.\n");
        return 1;
    }

    fprintf(pf, "%d %c %g\n", i, c, t);

    // Repositionner le curseur au début du fichier avant la lecture
    rewind(pf);

    fscanf(pf, "%d %c %f", &i, &c, &t);
    fprintf(pf, "%d %c %g\n", i, c, t);

    printf("Bonjour\n");

    fclose(pf);

    // Pause alternative compatible Windows/Linux
    getchar();
    return 0;
}
```

# 6)Lecture et écriture par caractère : fgetc et fputc

- Similaires aux fonctions getchar et putchar les fonctions fgetc et fputc permettent respectivement de lire et d'écrire un caractère dans un fichier.
- La fonction fgetc retourne le caractère lu dans le fichier et la constante EOF lorsqu'elle détecte la fin du fichier. Sa syntaxe est :
  - int fgetc(pf); /*où pf est de type FILE* */
- La fonction fputc écrit un caractère dans le flot de données . Sa syntaxe est :
  - fputc(int caractere, FILE *flot)
- Elle retourne l'entier correspondant au caractère lu (ou la constante EOF en cas d'erreur).

# Exemple

```c
#include <stdio.h>
#define ENTREE "entree.txt"
#define SORTIE "sortie.txt"

int main() {
    FILE *f_in, *f_out;
    int c;
    // Ouverture du fichier ENTREE en lecture
    if ((f_in = fopen(ENTREE, "r")) == NULL) {
        fprintf(stderr, "\nErreur: Impossible de lire %s\n", ENTREE);
        return 1; // Arrêt du programme en cas d'échec
    }
    // Ouverture du fichier SORTIE en écriture
    if ((f_out = fopen(SORTIE, "w")) == NULL) {
        fprintf(stderr, "\nErreur: Impossible d'écrire dans %s\n", SORTIE);
        fclose(f_in); // Fermer le fichier d'entrée déjà ouvert
        return 1;
    }
    // Recopie du contenu de ENTREE dans SORTIE
    while ((c = fgetc(f_in)) != EOF) {
        fputc(c, f_out);
    }
    // Fermeture des fichiers avec vérification des erreurs
    if (fclose(f_in) != 0) {
        fprintf(stderr, "\nErreur: Impossible de fermer %s\n", ENTREE);
    }
    if (fclose(f_out) != 0) {
        fprintf(stderr, "\nErreur: Impossible de fermer %s\n", SORTIE);
    }
    getchar();
    return 0;
}
```

# Positionnement dans un fichier : fseek, rewind et ftell

- fseek
  - Les différentes fonctions d'entrées-sorties permettent d'accéder à un fichier en mode séquentiel : les données du fichier sont lues ou écrites les unes à la suite des autres.
  - Il est également possible d'accéder à un fichier en mode direct, c'est-à-dire que l'on peut se positionner à n'importe quel endroit du fichier.
  - La fonction fseek permet de se positionner à un endroit précis.

# Positionnement dans un fichier : fseek, rewind et ftell

- fseek
- a pour prototype :
  - int fseek(FILE *pf, long deplacement, int origine);
  - La variable deplacement détermine la nouvelle position dans le fichier. Il s'agit d'un déplacement relatif par rapport à origine, compté en nombre d'octets.
  - La variable origine peut prendre trois valeurs :
    1. SEEK_SET (égale à 0) : début du fichier ;
    2. SEEK_CUR (égale à 1) : position courante ;
    3. SEEK_END (égale à 2) : fin du fichier.

# Positionnement dans un fichier : fseek, rewind et ftell

- rewind
  - int rewind(FILE *flot);
  - permet de se positionner au début du fichier. Elle est équivalente à fseek(flot,0, SEEK_SET) ;
- ftell
  - long ftell(FILE *flot);
  - retourne la position courante dans le fichier (en nombre d'octets depuis l'origine).

# Les Structures

# 3-Les structures

- Une structure est une suite finie d'objets de types différents. Qui sont regroupé au sein d'une même entité.
- Contrairement aux tableaux, les différents éléments d'une structure n'occupent pas nécessairement des zones contiguës en mémoire.
- Chaque élément de la structure, appelé **membre** ou **champ**, est désigné par un identificateur.

```
struct enreg {
    int  numero;
    int qte;
    float  prix;
};
```

# Les structures

- L'utilisation pratique d'une structure se déroule de la façon suivante :
  1. On commence par déclarer la structure elle-même. Le modèle général de cette déclaration est le suivant :

```
struct nom_structure {
        type_1 membre_1 ;
        type_2 membre_2 ;
        ...
       type_n membre_n ;
    } ;
```

# Les structures

2. Pour déclarer un objet de type structure correspondant au modèle précédent, on utilise la syntaxe :

struct nom_structure identificateur_objet ;

Ou bien, si le modèle n'a pas encore été déclaré au préalable :

```
struct nom_structure {
        type_1 membre_1 ;
        type_2 membre_2 ;
         ...
        type_n membre_n ;
       } identificateur_objet ;
```

3. On accède aux différents membres d'une structure grâce à l'opérateur membre de structure, noté ".". Le i-ème membre de objet est désigné par l'expression :

identificateur_objet.membre_i

# Les structures

- Chaque champ d'une structure peut être manipulé comme n'importe quelle variable de type correspondant. Voici quelques exemples utilisant la structure ***enreg:***

```
struct enreg art1,art2;        /* déclaration*/
art1.numero=15;                /*affectation*/
printf(''%f'',art1.prix);      /* affiche la valeur du champ prix*/
scanf(''%f'',&art2.prix);      /*lit une valeur qui sera affecté
                                  au champ prix de la structure art2 */
art1.numero++;        /* incrémente de 1 la valeur du champ
                         numero de la structure art1*/
```

# Les structures

- Il possible d'affecter à une structure le contenu d'une structure définie à partir du **même modèle**. Par exemple nous pouvons écrire:
  - art1=art2;
- Une telle affectation remplace :
  - art1.numero = art2.numero;
  - art1.qte = art2.qte;
  - art1.prix = art2.prix;
- Initialisation d'une structure au moment de sa déclaration :
  - struct enreg art1={100,290,3000};

# Les structures

- structure comportant des tableaux:
  - struct personne{ char nom[30];
    char prenom[20];
    float heures[31];
    } employe, courant;

  */*heures: nbre d'heures du travail dans chaque jours du mois*/*

    - employe.heures[4]; */*5ème éléments du tableau heures*/*
    - employe.nom[0]; */* premier caractère du champ nom*/*
  - Initialisation:
    struct personne courant={''Dupont'', ''julea''}

# Les structures

- Tableaux de structures

```
struct point { char nom;
               int  x;
               int  y;
             };
struct point courbe[50];
```

  - Cette structure sert à représenter un point d'un plan qui serait défini par son nom(caractère) et ses deux coordonnées.
  - courbe[i].nom;  courbe[i].x; courbe[i].y; */*représente le nom du point de rang i, la valeur du champ x et la valeur du champ y*/*
  - Initialisation :
    - struct point courbe[50]={{'A',10,25},{'M',13,34},{'R',2,6}};

# Les structures

- Structures comportant d'autres structures:
  - Supposant que l'intérieur de la structure employe et courant nous ayant besoin d'introduire deux dates: la date d'embauche et la date d'entrée. Si ces dates sont elles-mêmes des structures comportant trois champs correspondant au jour et à l'année, nous pouvons alors procéder aux déclarations suivantes :
    - struct date { int jour;
      int mois;
      int annee;};
    - struct personne{ char nom[30];
      char prenom[20];
      float heures[31];
      struct date date_embauche;
      struct date date_poste;
      } employe, courant;

# Les structures

- La notation :

  - employe.date_embauche.annee

  Représente l'année d'embauche correspondant à la structure employe. Il s'agit d'une valeur de type int.

  - courant.date_embauche

  Représente la date d'embauche de la structure courant, et nous pouvons faire l'affectation:

  - courant.date_embauche=employe.date_poste;

# Les structures

- Exercice:
  - Ecrire un programme qui lit au clavier des informations dans un tableau de structure du type point défini comme suit:

```
struct  point{
                    int num;
                    float x;
                    float y;
                  }
```

Le nombre d'éléments du tableau sera fixé par une instruction #define.

Affiche à l'écran l'ensemble des informations précédentes?