

HELHa Don Bosco

COURS DE PROGRAMMATION (2009 - 2010)
THÉORIE & EXERCICES (CODES)
2^{ème} quadrimestre

Notes prises au cours de M. WILFART

Ces notes n'ont pas été approuvées par le prof, elles ne peuvent donc être utilisées comme source source sûr. Des erreurs faites par l'auteur peuvent s'y trouver (et ce notamment au niveau de la théorie et/ou des exercices). Ceci n'est juste qu'un support pouvant éventuellement vous aider dans votre compréhension de la programmation.

THÉORIE & CODES

→ Les fonctions (rappels & nouveautés) :

- On peut créer ses propres fonctions. La fonction « main » est un exemple. Printf(), strlen(), scanf(), system(), getch(), gets(),... sont aussi des fonctions. On peut les réutiliser plusieurs fois.
- Ne pas tout inclure dans une fonction !
- La fonction « main » sert à contenir le code du « main ».
- Une fonction c'est un regroupement de code indépendant.
- Une fonction a un « nom » : choisir des noms évocateurs. Ne pas oublier les () .
- La fonction, on peut la tester indépendamment du reste du code.

Exemple :

```
void triabulle ()
// C'est nous qui choisissons le nom de notre fonction
{
    //bloc de code
}
```

- 3 choses importantes pour une fonction :

1°) Le prototype :

2°) La définition :

On y retrouve 3 choses :

- un nom évocateur
- déclaration entre ()
- le type de retour : void ← à retenir

3°) L'appel :

→ Créer une bonne application :

- Ça équivaut à un cercle concentrique (on va du point central vers le cercle le plus externe. L'interface utilisateur (= l'esthétique) étant la priorité la plus basse (soit le cercle le plus externe) et la fonctionnalité du programme, la priorité la plus haute (soit le centre du cercle, le point de référence)! Dans la fonctionnalité, la fonction est la plus importante (elle doit marcher), ensuite on se préoccupe du code.

- Technique de développement « n » tiers (dépend du nombre de couches) :
 - 1°) UTILISATEUR
 - 2°) TRAITEMENT
 - 3°) DATA
- Avoir la vision modulaire : penser aux cercles concentriques.
- Créer une fonction (ce à quoi il faut faire attention):

1^{ère} étape - ça marche :

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

void MaFonction()
{
    printf("je suis appelee");
}

// Voici la première fonction créée
// Maintenant que faire pour utiliser une fonction ?

void main()
{

    MaFonction();
    system("pause");
}
```

2^{ème} étape - ça marche pas (erreur) :

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

void main()
{
    MaFonction();
    system("pause");
}

void MaFonction()
{
    printf("je suis appelee");
}

// Problème ! Car on demande MaFonction, seulement on ne l'a pas encore
déclarée
```

3^{ème} étape - ça remarque :

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

void MaFonction();
```

```

void main()
{
    MaFonction();
    system("pause");
}

void MaFonction()
{
    printf("je suis appelee");
}
// Ce coup-ci ça remarque car on introduit bien la fonction MaFonction
AVANT !

```

- Compiler une application, 2 étapes :

Édition de lien

- o *Fatal error LINK1120* (par exemple) : 1 externe non résolue → le compilateur ne trouve pas la fonction.
- o Le compilateur C++ est sensible aux minuscules/majuscules : La variable : MaFonction est différente de la variable Mafonction → problème au niveau de l'édition des liens.

```

#include <stdio.h>
#include <stdlib.h>
#include "MesFonctions.h" /*Ce n'est pas une librairie ! C'est un fichier
d'en-tête*/

void main()
{
    MaFonction();
    system("pause");
}

```

dll = librairie (statique) avec des liens dynamiques, dll en mémoire donc chargée une seule fois → c'est plus pratique. On met les fonctions dans des dll et on modifie ensuite les dll.

→ Passage des paramètres

1^{ère} étape - ça marche :

```

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

void Traitement (int a)
{
    printf("donnee recue : %d\n",a);
    /*C'est une variable qui appartient à la fonction*/
}

void main ()
{
    Traitement(10);
    system("pause");
}

```

La variable « a » appartient à la fonction Traitement → c'est une variable locale → elle est déclarée pour une fonction et n'appartient qu'à la fonction « Traitement », ensuite elle est détruite.

2^{ème} étape - ça marche pas :

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

void Traitement (int a)
{
    printf("donnee recue : %d\n",a);
    /*C'est une variable qui appartient à la fonction*/
}
void main ()
{
    Traitement(10);
    printf ("%d\n", a); /*C'est une erreur "a" n'est pas disponible pour
cette fonction*/
    system("pause");
}
```

3^{ème} étape - ça marche toujours pas (à cause du « a » indisponible pour la fonction « void ») :

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

void Traitement (int a)
{
    printf("donnee recue : %d\n",a);
    /*C'est une variable qui appartient a la fonction*/
}
void main ()
{
    int b=10;
    Traitement(b);
    printf ("%d\n", a);
    system("pause");
}
```

- Le retour d'une fonction se traduit par : return

Exemple si on remplace « void » par « int » (Ça marche):

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

int Traitement (int a)
{
    int resultat;
    printf("donnee recue : %d\n",a);
    resultat=a*a;
    return resultat;
}
void main ()
{
    int b=100;
    int y;
```

```

    y=Traitement(b); //c'est une opérande binaire. Traitement est le nom
d'une fonction pas d'une variable
    printf ("%d\n", y);
    printf ("%d\n", Traitement(250));
    system("pause");
}

```

ExProg 21 - Créer une fonction Addition qui reçoit 2 arguments entiers. La fonction retourne le résultat de l'addition de ces deux entiers.

→ Pas de printf ou de scanf dans une fonction de traitement !

CODE :

```

/*Créer une fonction Addition qui reçoit 2 arguments entiers. La fonction
retourne le résultat de l'addition de ces deux entiers:*/

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

int Traitement (int a)
{
    int resultat;
    printf("donnee recue : %d\n",a);
    resultat=a*a;
    return resultat;
}

void main ()
{
    int b=100;
    int y;
    y=Traitement(b); //c'est une opérande binaire. Traitement est le nom
d'une fonction pas d'une variable
    printf ("%d\n", y);
    printf ("%d\n", Traitement(250));
    system("pause");
}

```

→ **Les pointeurs :**

int a ; « a » est une variable
a = 10 ; 1 variable contient une donnée

Pour pouvoir travailler avec les adresses, il faut utiliser les POINTEURS.

`int*b` ; ← « b » est un pointeur (= c'est l'adresse de b)

(Opérateur de : &...)

ExProg 22 - Mettre l'adresse de « a » dans « b » (« b » va contenir une adresse, et « a » va contenir une valeur).

CODE :

```

/* Mettre l'adresse de « a » dans « b » (« b » va contenir une adresse, et
« a » va contenir une valeur).*/

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

void main()
{
    /* 1ère étape pour créer un pointeur */
    int* b;
    int a=500;
    b=&a;
    /* 1ère étape finie */
    /* Maintenant il faut créer une redirection (grace a l'adresse de b
on va pouvoir trouver la valeur de a) */
    /* Quand je mets printf("%d\n", *b): on n'affiche pas le contenu de b
mais de la variable dont l'adresse est dans b/ */
    printf("%d\n", *b);
    *b=256; /* Ca va rien changer, ça sera toujours 500 la réponse */
    system("pause");
}

```

Exercice (non-officiel) :

```

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

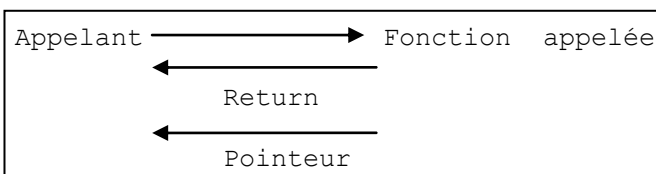
void Traitement(int *var)
{
    *var=*var+10;
}

void main()
{
    int a=10;
    Traitement (&a);
    printf("%d\n", a);
    system("pause");
}

```

Quel sera le résultat ? Que vaudra « a » ?

Réponse : 20, car « a » contient la valeur 10, que « a » se trouve dans *var, et que l'on incrémente de 10 « a » en passant par *var.



On va de l'appelant vers la fonction appelée. Pour le chemin inverse (= le retour (pour renvoyer une réponse)), on peut le faire de deux manières : soit par un return

ExProg 23 : Faire passer l'adresse d'un tableau.**CODE :**

```
/* Faire passer l'adresse d'un tableau. */
#include <stdlib.h>
#include <stdio.h>

void RecupTab(int*tab);
void Recuptab(int tab[]); /* But : faire passer l'adresse d'un tableau. */

void main ()
{
    int TabEntier[]={1,2,3,4,5,6,7};

    RecupTab(&TabEntier[0]);
    RecupTab(TabEntier); /* Ces 2 écritures reviennent au même. */

    system("pause");
}

void RecupTab(int*tab)
{
    printf("%d\n",tab[1]); /* Je peux accéder à n'importe quel élément du
tableau. */
}

void RecupTab2(int tab[])
{
}
```

Remarque sur les pointeurs :

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

void RecupTab(int*tab);
void Recuptab(int tab[]); /* But : faire passer l'adresse d'un tableau. */

void main ()
{
    int TabEntier[]={1,2,3,4,5,6,7};
    // RecupTab(&TabEntier[2]);
    RecupTab(TabEntier);

    system("pause");
}

void RecupTab(int*tab)
{
    printf("%d\n",tab[1]); /* Je peux accéder à n'importe quel élément du
tableau. */
    printf("%d\n",*tab);
    printf("%d\n",(tab+1)); /* Adresse+1 : ça ne va pas, une adresse
reste une adresse, une adresse pointe toujours sur un octet (entier) */
}

void RecupTab2(int tab[])
{
}
```

Sizeof = opérateur (pas besoin d'include) + permet de déterminer la taille totale du tableau en octets.

Exemple :

```
short int a; /* un entier short est codé sur 2 octets */
sizeof (a);
/* 2, va apparaitre à l'écran */
```

Fonctionnement du sizeof dans un tableau :

```
/* On tape: */

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

void main ()
{
    short int tab[20];
    printf("%d\n", sizeof(tab));
    system("pause");
}
/* Qu'est-ce qui va s'afficher à l'écran? */

/* Réponse: 40, car le tableau contient 20 entier de type short, pour
rappel un short est codé sur 2 octes donc ici on fait 20*2 */
```

Faire passer un tableau :

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

void RecupTab(int *tab,int taille);
void RecupTab2(int tab[]);

void main ()
{
    int TabEntier[]={1,2,3,4,5,6,7};
    char tab[255]="bonjour";
    printf("%d\n",strlen(tab));

    //RecupTab(&TabEntier[2];
    RecupTab(TabEntier,7);
    system("pause");
}
void RecupTab(int*tab,int taille) /* Faire passer un tableau numérique =
faire passer la taille */
{
    int i;
    printf("%d\n",tab[1]);
    printf("%d\n",*tab);
    printf("%d\n",*(tab+1));

    for(i=0;i<taille;i++)
    {
        tab[i]++;
        /*(tab+i;
    }
}
```

ExProg 24 - Écrire une fonction devant rechercher l'entier le plus grand dans un tableau d'entiers. Cette valeur sera retrouvée par la fonction.

CODE :

```

/* Écrire une fonction devant rechercher l'entier le plus grand dans un
tableau d'entiers. Cette valeur sera retrouvée par la fonction. */

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

int Rechercher1(int *Tab, int taille);

void main ()
{
    int taille=9;
    int Tab[]={1,2,3,4,5,6,5,2,1};
    printf("%d\n", Rechercher1(Tab,9));
    // Rechercher1(Tab,taille);
    system("pause");
}

int Rechercher1(int *Tab,int taille)
{
    int lpg;
    int i;

    for(i=1,lpg=Tab[0];i<taille;i++)
    {
        if(Tab[i]>lpg)
        {
            lpg=Tab[i];
        }
    }
    return(lpg);
}

```

ExProg 25 - Écrire une fonction permettant de calculer la mensualité à rembourser à un capital emprunté.

La fonction reçoit comme argument :

- k : capital emprunté → capital
- t : taux annuel proportionnel → taux
- n : nombre de mensualités → nbmens

La fonction retourne la mensualité calculée.

$$m = (k * (t/12)) / (1 - (1 + t/12)^{-n})$$

→ float mensualité (int capital, float taux, int nbmens)

Pour calculer une puissance en C/C++ : c'est pas '^', mais il faut passer par une fonction 'pow(x,y)', attention il ne faut pas négliger le 'math.h' comme en-tête précompilatoire.

CODE :

```

/* Écrire une fonction permettant de calculer la mensualité à rembourser à
un capital emprunté. */

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

float mensualite (int capital, float taux, int nbmens);

void main ()
{
    int capital, nbmens;
    float taux;

    /* Mêmes noms mais rien avoir avec les noms de l'autre fonction */

    printf ("Quel est le capital emprunte? ");
    flushall();
    scanf ("%d", &capital);

    printf("Quel est le taux?");
    flushall();
    scanf("%f", &taux);

    printf("Combien de mensualite? ");
    flushall();
    scanf("%d", &nbmens);

    printf("%f\n",mensualite(capital, taux, nbmens));
    /* On affiche m */

    system("pause");
}

float mensualite(int capital, float taux, int nbmens)
{
    float m;
    taux=taux/100;
    m= (capital*(taux/12))/(1-pow((1+(taux/12)), (-nbmens)));
    return m;
}

```

**ExProg 26 - Calculer un taux de remboursement (amélioré :
vérifie si l'utilisateur n'a pas entré des données erronées.)**

CODE :

```

/* Ecrire une fonction permettant de calculer la mensualité à rembourser à
un capital emprunté. */
/* La fonction reçoit comme arguments : K=capital emprunté, t=taux annuel
proportionnel, n=nombre de mensualités. */
/* La fonction retourne la mensualité calculée. */
/*Si les calculs ne sont pas bien faits, on le fait savoir!*/

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

```

```
float mensualite(int capital,float taux,int n);
bool mensualite2 (int capital, float taux, int n, float *m);

void main()
{
    int capital,n;
    float taux;
    //même noms mais différent des variables de la fonction

    printf("Quel est le capital emprunte: ");
    flushall();
    scanf("%d",&capital);

    printf("Quel est le taux?");
    flushall();
    scanf("%f",&taux);

    printf("Combien de mensualite?");
    flushall();
    scanf("%d",&n);

    printf("%f\n",mensualite(capital,taux,n));
    //on affiche m

    system("pause");
}

float mensualite(int capital,float taux,int n)
{
    float m;
    taux=taux/100;
    m=(capital*(taux/12))/(1-pow((1+(taux/12)),(-n)));
    return m;
    //retourner la fonction
}

bool mensualite2 (int capital, float taux, int n, float *m)
{
    if (capital>0 && taux>=0 && n>0)
    {
        taux=taux/100;
        *m=(capital*(taux/12))/(1-pow((1+(taux/12)),(-n)));
        return true;
    }

    else
    {
        return false;
    }
}
```

→ Typedef :

```
Typedef {
// instructions
}ErrorCode ;
```

Exemple concret :

```
typedef enum{
    Pas_Erreur,
    Capital_Erreur,
    Taux_Erreur,
    Periode_Erreur
}ErrorCode;
```

→ Écriture plus facile.

Opérateur logique bit à bit :

```
| : ou
& : et
^ : ou excl
~ : complément
```

Ou		
Et	&&	&
Ou excl		^
Complément	!	~

Applications :

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

void main ()
{
    int a=10;
    int b=20;
    int c=a&&b;

    printf("%d\n",c);

    system("pause");
    /* Quel sera la réponse? */
}
```

La réponse sera 1.

		Réponse
False	False	False
False	True	False
True	False	False
True (10)	True (20)	True (1)

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

void main ()
{
    int a=10;
    int b=20;
    int c=a&&b;
    int d=a&b;

    printf("%d\n",c);
    printf("%d\n",d);

    system("pause");
    /* Quel sera la réponse? */
}
```

}

ExProg 27 - Créer un tableau à 2 dimensions et calculer la moyenne.**CODE :**

```

/* Créer un tableau à 2 dimensions et calculer la moyenne. */

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

float moyenne(float *tab, int ligne);

void main ()
{
    float tab[][2]={{1,10},{4.5,20},{50,50}};
    printf("%f\n",moyenne(&tab[0][0],3));
    system("pause");
}

float moyenne(float *tab, int ligne)
{
    float somme=0;
    int total=0;
    int i;

    for(i=0;i<ligne;i++)
    {
        somme=somme+tab[(i*2)];
        total=total+tab[(i*2)+1];
    }
    return(somme*100/total);
}

```

tab[][2] = tableau à 2 dimensions

**ExProg 28 - Soit une fonction devant calculer le nombre d'entiers présents dans un tableau dont la valeur est comprise dans une fourchette [x,y]
La fonction retournera cette valeur calculée.****CODE :**

```

/* Soit une fonction devant calculer le nombre d'entiers présents dans un
tableau dont la valeur est comprise dans une fourchette [x,y]
La fonction retournera cette valeur calculée. */

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

int entierspresent(int tab[], int x,int y,int ligne);

void main ()
{
    int tab[]={1,2,3,4,5,6,7};
    int taille=7;
    int x=2;
    int y=6;
}

```

```

        /* maintenant récupérer ce que la variable retourne*/

printf("%d\n",entierspresent(tab,x,y,taille));
system("pause");

}
int entierspresent(int tab[], int x,int y,int ligne)
{
    int cpt=0,i;

    for(i=0;i<ligne;i++)
    {
        if(tab[i]>= x && tab[i] <= y)
            cpt++;
        else; /*pas nécessaire*/
    }
    return cpt;
    /*retourner l'entier contenu dans la variable*/
}

```

ExProg 29 - Soit une fonction devant vérifier qu'un tableau d'entiers passé en argument est trié. Un argument permet de renseigner si l'on désire tester le tri est croissant ou décroissant. La fonction retourne un booléen.

→ False → non trié

→ True → trié

Remarque pour les pointeurs, 2 syntaxes :

```

Bool trier (int *tab,...)
{
    Tab[i]=5 ;
    *(tab+i)=5 ;
}
Ne pas les mélanger!

```

CODE :

```

/* Soit une fonction devant vérifier qu'un tableau d'entiers passé en
argument est trié. Un argument permet de renseigner si l'on désire tester
le tri est croissant ou décroissant. La fonction retourne un booléen.
--> False : non trié
--> True : trié */

/* TRI CROISSANT */

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

bool trier (int *tab, int taille, int type);

void main()
{
    int tab []={1,2,3,4,5,6,7,8};
    if (trier(tab,8,0)==true)
    {

```

```
        printf("tableau trie\n");
    }
    else
    {
        printf("tableau non trie /n");
    }
    system("pause");
}

bool trier (int *tab, int taille, int type)
{
    int i;
    for (i=0; i<taille-1; i++)
    {
        if (type ==0)
        {
            if (tab[i] > tab[i+1])
            {
                return false;
            }
        }
        else
        {
            if (tab[i] < tab[i+1])
            {
                return false;
            }
        }
    }
    return true;
}

/* bool trier2 (int *tab, int taille, int type)
{
    int i;
    bool result=true;
    for (i=0; i<taille-1; i++)
    {
        if (type ==0)
        {
            if (tab[i] > tab[i+1])
            {
                result=false;
                break;
            }
        }
        else
        {
            if (tab[i] < tab[i+1])
            {
                result=false;
                break;
            }
        }
    }
    return result;
}*/

/* bool trier3 (int *tab, int taille, int type)
{
    int i;
```

```

bool result=true;
for (i=0; i<taille-1 && result==true; i++)
{
    if (type ==0)
    {
        if (tab[i] > tab[i+1])
        {
            result=false;
        }
    }
    else
    {
        if (tab[i] < tab[i+1])
        {
            result=false;
        }
    }
}
return result;
}*/

```

ExProg 30 - Soit une fonction devant déterminer si un tableau d'entiers passé en argument est trié. La fonction retourne un entier :

- 0 → tab non trié
- 1 → tab trié par ordre croissant
- 2 → tab trié par ordre décroissant

CODE :

```

/* Soit une fonction devant déterminer si un tableau d'entiers passé en
argument est trié. La fonction retourne un entier.
0 : tab non trié
1 : tab trié par ordre croissant
2 : tab trié par ordre décroissant */

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

int Ordre (int *tab, int taille);

void main ()
{
    int tab[]={1,7,6,5,4,3,2,1};

    if (Ordre(tab,8)==1)
    {
        printf("Trie dans l'ordre croissant\n");
    }
    else if (Ordre(tab,8)==2)
    {
        printf("Trie dans l'ordre decroissant\n");
    }
    else
    {
        printf("Non trie\n");
    }
    system("pause");
}

```

```

int Ordre (int *tab, int taille)
{
    int indice;
    int i=0, j=0;

    while(i < taille-1 && tab[i] > tab[i+1])
    {
        i++;
    }

    while(j < taille-1 && tab[j] < tab[j+1])
    {
        j++;
    }

    if(i==taille-1)
    {
        indice=2;
    }
    else if(j==taille-1)
    {
        indice=1;
    }
    else
    {
        indice=0;
    }

    return indice;
}

```

→ Structure & Typedef struct

Branche structure entité étudiant

```

int a ;
etudiant x ;

```

```

/* 1ère solution */
struct etudiant
{
    char nom[50];
    char prenom[50];
};
void main()
{
    struct etudiant x; //étudiant = nom donné à une structure
}

/* 2ème solution */
typedef struct
{
    char nom[50];
    char prenom[50];
}etudiant;

```

```

void main()
{
    etudiant y; /* étudiant devient un type à part entière + plus besoin
du struct */
}

/* 3ème solution : créer un pointeur */
void main()
{
    etudiant tab[20];
    etudiant *ptr;
    etudiant y;
    ptr = &y;
}

/* Rentrer un nom pour étudiant */
void main()
{
    etudiant tab [10];
    etudiant y;

    strcpy(y.nom, ("Durant")); /* Le point est utilisé pour l'élément
de gauche qui est un élément/variable simple. Derrière le point est
renseigné le champs */
    strcpy(tab[1].nom, "Dupond");

    y.mat=10;

    /* Et via un pointeur? */

    ptr =&y;
    printf("matricule:%d",ptr->mot);
}

```

ExProg 31 - Soit une fonction recevant en argument deux tableaux d'entiers (même longueur). La fonction retournera un entier i correspondant au nombre d'entiers identiques dans les deux tableaux aux mêmes positions (est-ce que $A[i] = B[i]$?).

CODE :

```

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

int CompareTab(int *tb1,int *tb2,int taille);

void main ()
{
    int a;
    int tb1[]={1,2,3,4,5,6,7,8}, tb2[]={1,3,2,4,5,6,8,7};

    a=CompareTab(tb1,tb2,8);

    printf("\nNombre de cases identiques : %d\n", a);
    system("pause");
}

int CompareTab(int *tb1,int *tb2,int taille)

```

```

{
    int cpt=0, i;
    int *ptr=&cpt;

    for(i=0;i<taille-1;i++)
    {
        if(tb1[i]==tb2[i])
        {
            cpt++;
        }
    }
    return *ptr;
}

```

ExProg 32 - On a une structure « branche » qui contient les points d'un étudiant dans chaque branche. Ensuite calculer la moyenne de tous les points.

CONCEPT :

```

typedef struct
{
    float points;
    int total;
}branche;

void main()
{
    branche tab[10];
}

```

CODE :

```

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

typedef struct
{
    float points;
    int total;
}branche;

double Moyenne (branche tab[], int taille);

void main ()
{
    branche tab[10];
    int i, nbchamp;

    for(i=0;i<10;i++)
    {
        printf("Introduire les points sous le format '/' : ");
        fflush();
        nbchamp=scanf("%f/%d", &tab[i].points, &tab[i].total);

        if(nbchamp!=2)
        {
            printf("Erreur de format!\n");
            i--;
        }
    }
}

```

```
    printf("La moyenne vaut : %f\n", Moyenne(tab, 10));
    system("pause");
}

double Moyenne (branche tab[], int taille)
{
    float somme=0;
    int total=0, i;

    for(i=0;i<taille;i++)
    {
        somme+= tab[i].points;
        total+= tab[i].total;
    }
    return (somme*20.0)/total;
}
```

ExProg 33 - Structure fiche

- NOM chaine de caractères;
- PRENOM chaine de caractères;
- SUIVANT pointeur sur la structure fiche.

CODE :

```
/* Structure d'une fiche contenant : nom, prénom */

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

struct fiche
{
    char nom[20];
    char prenom[20];
    struct fiche *suivant;
};

void main ()
{
    struct fiche x;
    struct fiche *ptr;
    strcpy(x.nom, "CRUCQ");
    strcpy(x.prenom, "JEREMY");
    x.suivant=NULL;

    ptr=(struct fiche *)malloc(sizeof(struct fiche));

    if(ptr!=NULL)
    {
        strcpy(ptr->nom, "CRUCQ!");
        strcpy(ptr->prenom, "JEREMY!");
        ptr->suivant=NULL;
        x.suivant=ptr;
    }
    else
    {
        printf("Erreur d'allocation\n");
    }
}
```

```

/* Afficher les fiches créées */

ptr=&x;
while(ptr!=NULL)
{
    printf("%s\n",ptr->nom);
    printf("%s\n",ptr->prenom);
    ptr=ptr->suitant;
}
system("pause");
}

```

void* = pointeur générique → il peut pointer sur n'importe quoi.

Void*malloc(size_to_size);

Transtypage = changer le type d'une adresse

Exemple :

```
ptr = (struct fiche *)malloc(
```

ExProg 34 -

- 1) Demander à l'utilisateur le nombre de fiches à créer
- 2) Pour chaque fiche :
 - création dynamique de la flèche
 - demande d'introduction des noms et prénom
 - mise en place des liens.

CODE :

```

/*
1) Demander à l'utilisateur le nombre de fiches à créer
2) Pour chaque fiche :
   - création dynamique de la flèche
   - demande d'introduction des noms et prénom
   - mise en place des liens
*/

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

struct fiche
{
    char nom[20];
    char prenom[20];
    struct fiche *suitant;
};

void main ()
{
    struct fiche x;
    struct fiche *ptr;
    int NombreDeFiche;
    printf("Quel est le nombre de fiches a creer?\n");
    scanf("%d", &NombreDeFiche);
    strcpy(x.nom, "CRUCQ");
    strcpy(x.prenom, "JEREMY");
}

```

```
x.suivant=NULL;

ptr=(struct fiche *)malloc(sizeof(struct fiche));

if(ptr!=NULL)
{
    strcpy(ptr->nom, "CRUCQ!");
    strcpy(ptr->prenom, "JEREMY!");
    ptr->suivant=NULL;
    x.suivant=ptr;
}
else
{
    printf("Erreur d'allocation\n");
}

/* Afficher les fiches créées */

ptr=&x;
while(ptr!=NULL)
{
    printf("%s\n", ptr->nom);
    printf("%s\n", ptr->prenom);
    ptr=ptr->suivant;
}
system("pause");
}
```

→ Les fichiers => données permanentes (sur le HDD).

- o Base de données (2^{ème} année) → Access/SQL Express
 - o Fichier à plat → XML/texte
 - Avantage XML : fichier accompagné d'une feuille de style (fichier facilement convertissable en plusieurs formats).
- Les informations dans un enregistrement s'appellent des champs (field).

→ Gérer des fichiers textes.

On va retrouver :

- o Séparateur d'enregistrement : newline
- o Séparateur de champ : c'est une virgule « , »
- o Les champs
 - 2 modes :
 - longueur variable (ou CSV)
Principe : longueur que l'on veut mais difficilement modifiable.
 - Économie de la place.
 - CSV recommandé si on veut mettre des infos dans un fichier avec la certitude que l'on ne va plus le modifier par la suite. Sinon problème → pertes de données dû à un mauvais décalage et/ou remplacement.

- champs constants
 Principe : longueur limitée → troncature si
 NOM trop long.
 10 caractères prévus pour rentrer un NOM
 Ex : Dupond---- → 4 caractères blancs
 └───┬───> Caractère nul

Fichier haut niveau : **fopen, fclose, fprintf, fscanf, ...**

- o Ouvrir un fichier : **fopen**
- o Fermeture d'un fichier : **fclose** (**fcloseall** → tous les fichiers ouvert)
- o Lecture : **fscanf** **fread**
- o Écriture : **fprintf** **fwrite**
- Accès en mode texte *Accès en mode binaire*
- o Déplacement (= fonctions associées par 2) : **fgetpos** **fsetpos**
 (= récupération d'une position).
- o Récupérer l'endroit dans lequel on se trouve dans un fichier : **fseek** (= se positionner). Le déplacement se fait par une référence
 - « Je veux me déplacer par rapport à l'endroit où je suis. »
 - « Je veux me déplacer de x octets à partir du début du fichier. »
 - Où suis-je ? : **ftell**
 - Revenir au début du fichier : **rewind**
- o Lorsqu'on veut faire un accès en écriture → buffer verrouillé.
- o Lorsqu'on veut écrire dans un fichier à un endroit précis → **fseek** « se positionner de 0 octets par rapport à l'endroit où je suis »

Fopen :

- `Fopen(path=chemin d'accès au fichier , format d'ouverture)`
- `Fopen('`c:\\test.txt' , ...) ;`
`Char * path= `c:\\test.txt;`
- `FILE * fopen (... , ...)`
 → pointeur de descripteur de fichier

Concernant le format :

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

FILE * ptrfile;

void main()
{
    ptr_file=fopen("c:\\test.txt", "format d'ouverture");

    /* Choix du format d'ouverture:
```

```

r : read
w : write
a : append --> permet d'écrire à la fin d'un fichier
+ : accès normal + accès opposé ainsi:
    r+ : lecture/écriture
    w+ : écriture
    a+ : appen/lecture
b : binaire
t : texte

Remarque à propos des formats:

r+ : le fichier doit exister
w+ : le fichier sera créé s'il n'existe pas OU s'il
existe, son contenu sera effacé!!
a+ : le fichier sera créé s'il n'existe pas OU s'il
existe, son contenu est conservé!

Savoir si il y a eu un problème lors de l'ouverture du
fichier

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

FILE * ptrfile;

void main()
{
    ptr_file=fopen("c:\\test.txt", "format
d'ouverture");
    if(ptrfile==NULL)
    {
        printf("Erreur d'accès au fichier\n");
    }
}

TOUJOURS UN CONDITION! */

```

ExProg 35 - Créer un programme permettant de créer un fichier (texte).

CODE :

```

/* Comment créer un fichier texte (Bloc Notes) */

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

FILE* ptrfile; /* ptrfile = Nom de la variable */

void main ()
{
    ptrfile=fopen("test.txt","w+"); /* Si ça aurait été un r+, le fichier
aurait du exister sinon message d'erreur */
    if(ptrfile=NULL)
    {
        printf ("Erreur d'acces au fichier");
    }
    system("pause");
}

```

ExProg 36 - Même chose que l'exercice 35, mais on invite l'utilisateur à entrer un NOM, PRÉNOM,...

→ fread & fwrite : pas de format

```
fwrite(void*ptr, size_t size, size_t count, file*ptr);
```

- o void ici est un pointeur générique : permet de faire passer n'importe quelle adresse.
- o le deuxième « size » : taille d'une instance exprimée en octets.
- o le troisième « size » : compteur d'instance
- o quand on travaille avec des structures, c'est mieux de travailler avec des variables de type « char », car on travaille avec des champs constants.

```
typedef struct
{
    char nom[25];
    char prenom[25];
    char matricule[10];
}identite;
```

?:

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

FILE*ptr; /* conseillé de le mettre hors du main */
void main (void)
{
    typedef struct
    {
        char nom[25];
        char prenom[25];
        char matricule[10];
    }identite;

    identite x;

    strcpy("Dupond",x.nom);
    strcpy("Albert",x.prenom);
    strcpy("1010",x.matricule);

    ptr=fopen("c:\\test.txt","r+");

    identite tab[10];

    fwrite(tab,
           (&tab[0],
            (tab,sizeof(identite)*10,1,ptr);
            sizeof(tab)
            (tab, sizeof(identite),10,ptr);
    )
}
```

CODE :

```
/* Même chose que l'exercice 35, mais on demande à l'utilisateur d'entre un
nom et un prénom autant de fois qu'il le souhaite */

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

FILE* ptrfile; //FILE est une structure!!!

void main ()
{
    char nom[50], prenom[50];
    char choix;
    ptrfile=fopen("test.txt","wn+");

    if(ptrfile==NULL)
    {
        printf("Erreur acces au fichier");
    }
    else
    {
        do
        {
            printf("Voulez-vous introduire un nom et un
prenom?");

            fflush();
            scanf("%c",&choix);
            if(choix=='o')
            {
                printf("Introduire un nom :\n");
                fflush();
                scanf("%s", &nom);
                printf("Introduire un prenom :\n");
                fflush();
                scanf("%s", &prenom);
                fprintf(ptrfile,"%s,%s\n", nom, prenom);
            }
        }
        while(choix=='o');
        fclose(ptrfile);
    }
    system("pause");
}
```

2^{ème} solution pour l'exercice 36 :

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

FILE* ptrfile; //FILE est une structure!!!

void main ()
{
    char nom[50], prenom[50];
    char choix;
    ptrfile=fopen("test.txt","r+");
    if(ptrfile==NULL)
    {
        printf("Erreur acces au fichier");
    }
    else
```

```

    {
        while(!feof(ptrfile))
        {
            fscanf(ptrfile, "%[^,],%[^\\n]\\n", nom, prenom);
            printf("Nom : %s\\nPrenom : %s\\n", nom, prenom);
            //On demande de ne pas recuperer les virgules et les \\n
du fichier.
        }
        fclose(ptrfile);
    }
    system("pause");
}

```

ExProg 37 - Demander le nombre de fiche signalétique (nom, prénom) à introduire. Introduire chacune des fiches. Transférer les fiches dans un fichier. !! Utilisation de l'allocation dynamique !!

CODE :

```

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

typedef struct
{
    char nom[25];
    char prenom[25];
    char matricule[10];
}identite;
FILE*ptr;

void main()
{
    int nbfiche=0;
    int i=0;
    identite *x;
    printf("Combien de fiche signaletique voulez-vous enter?\\n");
    fflush();
    scanf("%d",&nbfiche);
    x=(identite*)malloc(sizeof(identite)*nbfiche);

    while(i<nbfiche)
    {
        printf("Veuillez introduire le NOM:\\n");
        scanf("%s",x[i].nom);
        printf("Veuillez introduire le PRENOM:\\n");
        scanf("%s",x[i].prenom);
        printf("Veuillez introduire le MATRICULE:\\n");
        scanf("%s",x[i].matricule);
        i++;
    }
    ptr=fopen("fiche.txt", "w");
    if(ptr!=NULL)
    {
        fwrite(x, sizeof(identite), nbfiche, ptr);
        fclose(ptr);
    }
    else

```

```

    {
        printf("Erreur d'ouverture du fichier!\n");
    }

    system("pause");
    free(x);
}

```

2^{ème} solution pour l'exercice 37 :

CODE :

```

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

FILE* ptrfile; //FILE est une structure!!!

void main ()
{
    char nom[50], prenom[50];
    char choix;
    ptrfile=fopen("test.txt","r+");

    if(ptrfile==NULL)
    {
        printf("Erreur acces au fichier");
    }
    else
    {
        do
        {
            printf("Voulez-vous introduire un nom et un prenom?");
            fflush();
            scanf("%c",&choix);
            if(choix=='o')
            {
                printf("Introduire un nom :\n");
                fflush();
                scanf("%s", &nom);
                printf("Introduire un prenom :\n");
                fflush();
                scanf("%s", &prenom);
                fprintf(ptrfile,"%-20.20s%,%-20.20s\n", nom,
prenom);
            }
        }
        while(choix=='o');
        fclose(ptrfile);
    }
    system("pause");
}

```

→ **static** : permet à une variable locale de ne pas être détruite après utilisation (on la transforme en variable globale).

→ **Effacer des données contenues dans une structure (nom, prénom, matricule):**

A		A
B		B
C → à supprimer	=>	D
D		E
E		

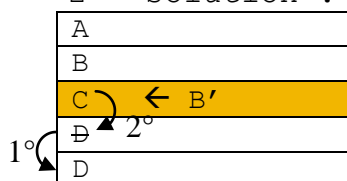
- o Problème : La taille de fichier se voit diminué d'une structure (la C). Donc on va imaginer que le programme ne va pas lire la fiche C (il la passe), il laisse un « trou », et il passe directement à la fiche D.
- o => besoin d'une variable « temporaire » supplémentaire.

→ Insérer une fiche :

1^{ère} solution :

A	⇨ tmp	A	
B (ajouter B')		B	
C		B'	
D		C	
		D	

2^{ème} solution :



3^{ème} solution :

	→ index	index	index
1°) A (110 oct)		2°)	2°)
2°) B		4°) (ajout 5°))	4°)
3°) C		3°)	3°) → 5
4°) D		1°)	3°)
5°) E			1°)

ExProg 38 - Effacer une fiche.

CODE :

```

/* Effacer une fiche */

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

int Comptage_Fiches(FILE *ptr_file);

typedef struct
{
    char nom[50];
    char prenom[50];

```

```
    char matricule[10];
}fiche;

FILE * ptr_file;
FILE * ptr_tmp;

void main()
{
    fiche MaFiche;
    int compte;
    int Effacer;
    int i;

    ptr_file=fopen("fiches.txt","a+");

    if (ptr_file == NULL)
    {
        printf("erreur d acces au fichier");
        exit;
    }

    compte=Comptage_Fiches(ptr_file);
    printf("nbre de fiches: %d\n",compte);
    printf("introduire la fiche a effacer");
    scanf("%d",&Effacer);

    if (Effacer <= compte)
    {
        ptr_tmp=fopen("tmp.txt","w");
        if(ptr_tmp==NULL)
        {
            printf("erreur de creation du fichier\n");
        }
        else
        {
            rewind(ptr_file);
            for(i=1;i<=compte;i++)
            {
                fread(&MaFiche,sizeof(fiche),1,ptr_file);
                if(i==Effacer) continue;
                fwrite(&MaFiche,sizeof(fiche),1,ptr_tmp);
            }
            fclose(ptr_tmp);
            fclose(ptr_file);
            system("del fiches.txt");
            system("rename tmp.txt fiches.txt");

            ptr_file=fopen("fiches.txt","r");
        }
    }

    fclose(ptr_file);
    system("pause");
}

int Comptage_Fiches(FILE *ptr_file)
{
    int taille,nbre_fiche;
    fseek(ptr_file,0,SEEK_END);
    taille=ftell(ptr_file);
```

```
    nbre_fiche=taille/sizeof(fiche);
    return nbre_fiche;
}
```

ExProg 39 - Insérer une fiche.

CODE :

```
/* Insérer une fiche */

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

int Comptage_Fiches(FILE *ptr_file);

typedef struct
{
    char nom[50];
    char prenom[50];
    char matricule[10];
}fiche;

FILE * ptr_file;
FILE * ptr_tmp;

void main()
{
    fiche MaFiche;
    int compte;
    int inserer;
    int i=0;

    ptr_file=fopen("fiches.txt","a+");

    if (ptr_file == NULL)
    {
        printf("erreur d acces au fichier");
        exit;
    }

    compte=Comptage_Fiches(ptr_file);
    printf("nbre de fiches: %d\n",compte);
    printf("Introduire la fiche a inserer");
    scanf("%d",&inserer);

    if (inserer <= compte)
    {
        ptr_tmp=fopen("tmp.txt","w");
        if(ptr_tmp==NULL)
        {
            printf("Erreur de creation du fichier\n");
        }
        else
        {
            rewind(ptr_file);

            while(i<compte)
            {
                if(i==inserer)
```

```
        {
            printf("Introduire NOM : ");
            scanf("%s",MaFiche.nom);
            printf("Introduire PRENOM : ");
            scanf("%s",MaFiche.prenom);
            printf("Introduire MATRICULE : ");
            scanf("%s",MaFiche.matricule);
            inserer=i-1;
        }
    else
    {
        fread(&MaFiche,sizeof(fiche),1,ptr_file); /* 1
car une fiche à lire */
        i++;
    }
    fwrite(&MaFiche,sizeof(fiche),1,ptr_tmp);
}
fclose(ptr_tmp);
fclose(ptr_file);
system("del fiches.txt");
system("rename tmp.txt fiches.txt");

ptr_file=fopen("fiches.txt","r");
}

}
fclose(ptr_file);
system("pause");
}

int Comptage_Fiches(FILE *ptr_file)
{
    int taille,nbre_fiche;
    fseek(ptr_file,0,SEEK_END);
    taille=ftell(ptr_file);
    nbre_fiche=taille/sizeof(fiche);
    return nbre_fiche;
}
```

→ Fonctions de conversion

o Ascii → numérique

- atoi : entiers
- atol : idem
- sscanf : libre (%d, %ld, %f, %lf)
- atof : double, flottant (pas de correspondance entiers → char)

o Numérique → ascii

- itoa : entiers
- ltoa : idem
- sprintf : libre

- fcvt : double, flottant
- ecvt : idem

→ Les fonctions de gestion des chaînes de caractères

- o strcpy : copier une chaîne dans une autre.
- o strcat : concaténer une chaîne avec une autre.
- o strncpy : n caractères sont concaténés.
- o strcmp : comparaison de 2 chaînes.
- o strncmp : comparaison des n premiers caractères dans une chaîne.
- o toupper : conversion d'un caractère en majuscule.
- o tolower : conversion d'un caractère en minuscule.
- o strstr : rechercher une chaîne dans une autre.
- o strtok : rechercher des chaînes de caractères séparées par un caractère de séparation ('\|'\' ;').

ExProg 40 - Conversions de chaînes <-> entiers :

CODE :

```

/* Conversion - Les fonctions de conversion */
/* chaîne de caractères --> numérique (int --> long) */
/*                               <--          (float --> double) */

/* La fonction atoi  --> récupérer un entier */
/* La fonction atol  --> récupérer un long */
/* La fonction sprintf --> */
/* La fonction sscanf --> écriture dans une chaîne de caractères */

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

void main()
{
    char *tab="125";
    int result;
    result=atoi(tab);
    printf("%d\n",result);

    sscanf(tab,"%d",&result);
    printf("%d\n",result);

    system("pause");
}

```

ExProg 41 - Convertir un « float » en un « char ».

CODE :

```

/* Convertir un float en un char */

#include <stdio.h>

```

```
#include <stdlib.h>

void main ()
{
    char *tab; /* pointeur non-initialisé */
    char tab2[255];
    int posdec;
    int signe;
    float x =+12.752F;

    sprintf(tab2,"%0.3f",x); /* 3 chiffres après la virgule, le surplus
est coupé */

    printf("%s\n",tab2);

    tab=fcvt(x,3,&posdec,&signe);

    system("pause");
}
```

ExProg 42 - EXERCICE RÉCAPITULATIF SUR LES FICHES & LES STRUCTURES ! IMPORTANT ! (Ajouter une fiche, Effacer une fiche, voir la première fiche, voir la dernière fiche, voir fiche précédente, voir fiche suivante :

CODE :

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

typedef struct{
    char nom[255];
    char prenom[255];
    int efface;
}fiche;

void ajouter (FILE* ptr_ajout, fiche*x);
void first (FILE* ptr_first);
void last (FILE* ptr_last);
int compte(FILE * ptr_count);
void next(FILE* ptr_next);
void before(FILE* ptr_before);
int NumFiche(FILE * ptr_count);

FILE* ptr_file;

void main()
{
    int choix;
    fiche x;
    ptr_file=fopen("fiche.txt","a+");
    if(ptr_file==NULL)
    {
        printf("Erreur ouverture du fichier");
        exit;
    }
    do
    {
```

```
        system("cls");
        if(fread(&x, sizeof(fiche), 1, ptr_file)==1)
        {
            printf("Nom:\t%s\n", x.nom);
            printf("Prenom:\t%s\n", x.prenom);
        }
        else
        {
            printf("Nom:\n");
            printf("Prenom:\n");
        }
        printf("\n\n\n");
        printf("1: suivant, 2: precedent, 3: premier, 4: dernier, 5:
effacer, 6: ajouter, 0: quitter\n");
        scanf("%d", &choix);
        switch(choix)
        {
            case 1:
                next(ptr_file);
                break;
            case 2:
                before(ptr_file);
                break;
            case 3:
                first (ptr_file);
                break;
            case 4:
                last(ptr_file);
                break;
            case 5:
                break;
            case 6:
                x.efface=0;
                printf("Entrez le nom");
                scanf("%s", x.nom);

                printf("Entrez le prenom");
                scanf("%s", x.prenom);
                ajouter(ptr_file, &x);
                break;
            case 0:
                printf("On quitte");
                break;
            default:
                break;
        }
    }
    while(choix!=0);
    fclose(ptr_file);

    system("pause");
}

void ajouter (FILE* ptr_ajout, fiche*x)
{
    fseek(ptr_ajout, 0L, SEEK_END);
    fwrite(x, sizeof(fiche), 1, ptr_ajout);
    fseek(ptr_ajout, -sizeof(fiche), SEEK_END);
}

void first (FILE* ptr_first)
```

```
{
    fiche tmp;
    fseek(ptr_first, 0L, SEEK_SET);
    while(!feof(ptr_first))
    {
        if (fread(&tmp, sizeof(fiche), 1, ptr_first)==1)
        {
            if (tmp.efface==0)
            {
                fseek(ptr_first, -sizeof(fiche), SEEK_CUR);
                return;
            }
        }
    }
}

void last (FILE* ptr_last)
{
    fiche tmp;
    int i;
    int taille = compte(ptr_last);

    for(i=1; i<=taille; i++)
    {
        fseek(ptr_last, -i*sizeof(fiche), SEEK_END);
        fread(&tmp, sizeof(fiche), 1, ptr_last);

        if(tmp.efface == 0)
        {
            fseek(ptr_last, -sizeof(fiche), SEEK_CUR);
            return ;
        }
    }
}

int compte(FILE * ptr_count)
{
    int nombre;
    fseek(ptr_count, 0L, SEEK_END);
    nombre = ftell(ptr_count)/sizeof(fiche);

    return nombre;
}

void next(FILE* ptr_next)
{
    fpos_t position;
    fiche tmp;
    fgetpos(ptr_next, &position);

    while(!feof(ptr_next))
    {
        if(fread(&tmp, sizeof(fiche), 1, ptr_next)==1)
        {
            if(tmp.efface==0)
            {
                fseek(ptr_next, -sizeof(fiche), SEEK_CUR);
                return;
            }
        }
    }
}
```

```

    }
    }
    fsetpos(ptr_next, &position);
    fseek(ptr_next, -sizeof(fiche), SEEK_CUR);
}

void before(FILE* ptr_before)
{
    fiche tmp;
    int i;
    fpos_t position;
    fgetpos(ptr_before, &position);
    int posfiche = NumFiche(ptr_before);

    for(i=1; i<posfiche; i++)
    {
        fseek(ptr_before, -2*sizeof(fiche), SEEK_CUR);
        fread(&tmp, sizeof(fiche), 1, ptr_before);

        if(tmp.efface == 0)
        {
            fseek(ptr_before, -sizeof(fiche), SEEK_CUR);
            return ;
        }

    }
    fsetpos(ptr_before, &position);
    fseek(ptr_before, -sizeof(fiche), SEEK_CUR);
}

int NumFiche(FILE * ptr_count)
{
    int nombre;
    nombre = ftell(ptr_count)/sizeof(fiche);

    return nombre;
}

```

ExProg 43 - Écrire un programme permettant d'afficher dans l'ordre renseigné par l'index tous les enregistrements. (fichiers de structures + fichier index)

CODE :

```

/* Écrire un programme permettant d'afficher
dans l'ordre renseigné par l'index tous
les enregistrements. (fichiers de
structures + fichier index) */

/* PAS FAIT EN CLASSE --> MANQUE DE TEMPS */

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

void main ()
{
    system("pause");
}

```

