

Types Construits / Unions

- Unions
 - Cas où on doit représenter des données de types différents
 - Un seul de ces types de données est utile pour une instance d'objets donnée
 - Les Unions permettent l'utilisation d'un même espace mémoire pour des données de type différents à des moments différents
 - Syntaxe proche de celle de la déclaration d'une structure

Types Construits / Unions

Syntaxe	Exemple
<pre>union nomUnion{ type1 champ1 ; type2 champ2 ; } ;</pre>	<p>nomUnion ⇔ étiquette de l'union</p> <p>union indique une déclaration de l'union constituée d'une liste de déclaration entre accolades</p> <p>La déclaration d'une union ne réserve pas d'espace mémoire</p> <p>⇔ l'allocation se fera au moment de la définition d'une variable correspondant à ce modèle d'union</p> <p>⇔ réservation de mémoire sera faite comme pour une variable classique</p> <p>⇔ espace mémoire réservé = espace mémoire nécessaire pour stocker le plus grand des champs appartenant à l'union.</p>

Types Construits / Unions

Syntaxe	Exemple
<pre>union Cat{ int iCat ; char *chCat ; } ;</pre>	<p>la catégorie peut être soit de type int soit de type char *</p> <p>Un objet de type union est composé d'un seul champ choisi parmi ceux du modèle union</p> <p>Un objet de type structure est composé de la totalité des champs définis dans la structure</p> <p>L'accès aux champs se fait de la même façon que pour les structures</p>

Types Construits / Unions

Union comme champ de structure

Une Union peut être utilisée comme champ d'une structure

```
typedef struct produit {  
    int      reference, typeCat;  
    union cat{  
        int    iCat ;  
        char  *chCat ;  
    } cat ;  
    char  *nom, *provenance ;  
}produit;
```

typeCat = 0 si la catégorie de l'objet est représentée par un entier

typeCat = 1 si la catégorie de l'objet est représentée par une chaîne de caractères

On utilisera soit le champ **iCat** soit le champ **chCat** de l'union cat

Types Construits / Unions

Union comme champ d'une structure / illustration

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

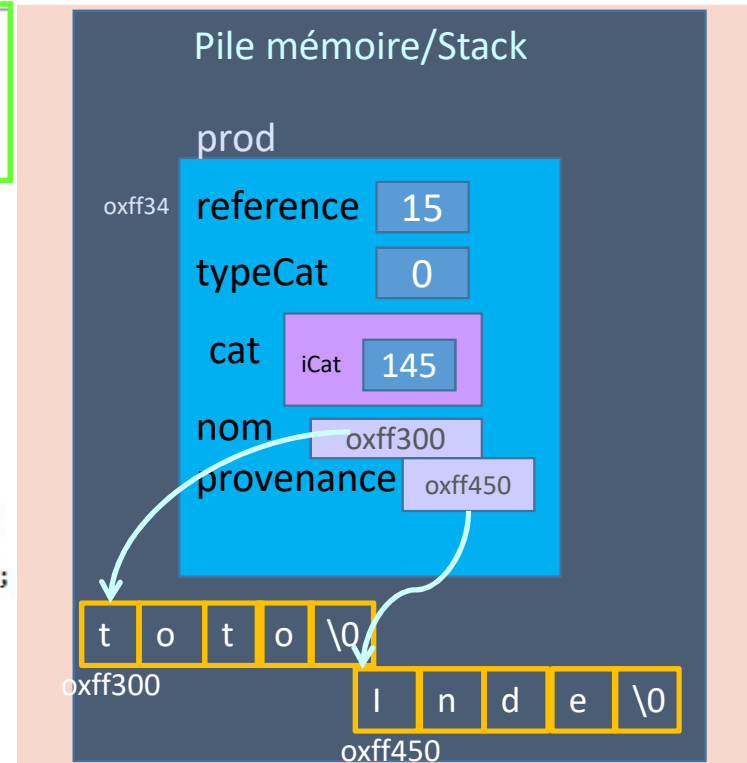
typedef struct produit {
    int    reference, typeCat ;
    union  cat{
        int    iCat ;
        char    *chCat ;
    } cat ;
    char    *nom, *provenance ;
} produit;

void afficheProduit(produit p){
    printf("nom du produit: %s\n", p.nom);
    printf("provenance du produit: %s\n", p.provenance);
    printf("référence du produit: %d\n", p.reference);
    switch(p.typeCat ) {
        case 0: printf("Catégorie du produit : %d\n", p.cat.iCat);
                break;
        case 1: printf("Catégorie du produit: %s\n ", p.cat.chCat);
                break;
    }
}

int main () {
    produit prod={15,0,{145,},"toto","Inde"};

    afficheProduit(prod);
    return EXIT_SUCCESS ;
}
```

nom du produit: toto
provenance du produit: Inde
référence du produit: 15
Catégorie du produit : 145



Types Construits / Unions

Union comme champ d'une structure / illustration

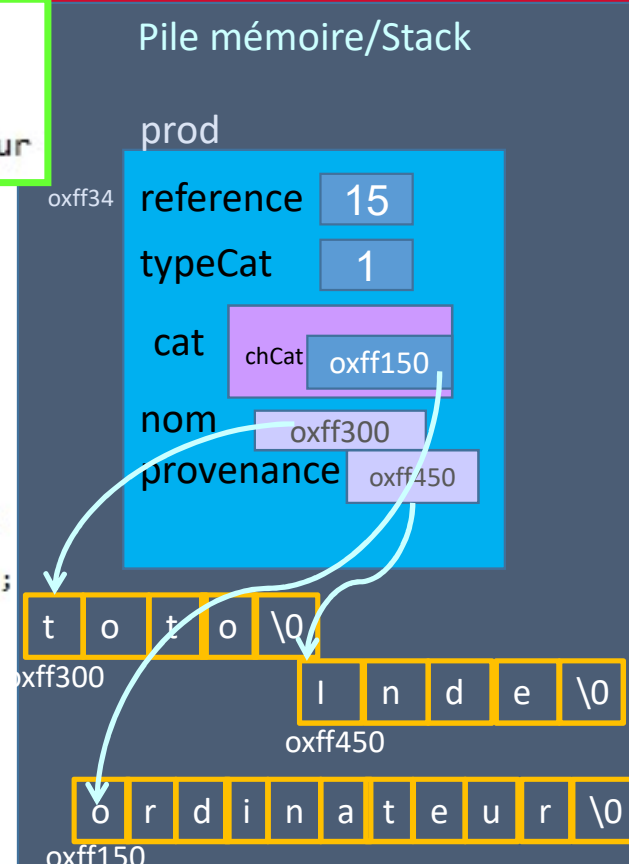
```
#include <stdio.h>
#include <stdlib.h>

typedef struct produit {
    int    reference, typeCat;
    union  cat{
        int    iCat ;
        char    *chCat ;
    } cat ;
    char    *nom, *provenance ;
} produit;

void afficheProduit(produit p){
    printf("nom du produit: %s\n", p.nom);
    printf("provenance du produit: %s\n", p.provenance);
    printf("référence du produit: %d\n", p.reference);
    switch(p.typeCat ) {
        case 0: printf("Catégorie du produit : %d\n", p.cat.iCat);
                break;
        case 1: printf("Catégorie du produit: %s\n ", p.cat.chCat);
                break;
    }
}

int main () {
    produit prod={15,1,{}, "toto", "Inde"};
    prod.cat.chCat="ordinateur";
    afficheProduit(prod);
    return EXIT_SUCCESS ;
}
```

nom du produit: toto
provenance du produit: Inde
référence du produit: 15
Catégorie du produit: ordinateur



Types Construits / Enumérations

- Enumérations
 - Une déclaration d'énumération permet de définir un modèle d'entiers dont les valeurs sont notées par des constantes symboliques
 - Permettent d'exprimer des valeurs constantes de type entier en associant ces valeurs à des noms
 - La 1^{ère} constante symbolique est associée à la valeur 0
 - La 2^{ème} constante symbolique est associée à la valeur 1
 -
 - Facilite la compréhension et la maintenance du programme

Types Construits / Enumérations

Syntaxe	
<code>enum nomEnum{ liste de symboles } ;</code>	<p>nomEnum ⇔ étiquette de l'énumération</p> <p>enum indique une déclaration d'énumération constituée d'une liste de symboles entre accolades</p> <ul style="list-style-type: none">⇔ les noms doivent être différents les uns des autres, et différents de noms de variables visibles. <p>Les valeurs sont représentées par des entiers comptés à partir de 0 si rien n'est précisé</p> <ul style="list-style-type: none">⇔ le compilateur se charge de convertir les symboles en valeurs entières <p>Mais on peut imposer des valeurs à certains symboles en le précisant par une initialisation des symboles dans la déclaration du type énuméré</p>

Types Construits / Enumérations

Syntaxe	
<pre>enum couleurRVB{ROUGE ,VERT, BLEU} ;</pre>	ROUGE ⇔ 0 VERT ⇔ 1 BLEU ⇔ 2
<pre>enum autresCouleurs{ JAUNE=4, ORANGE=6, VIOLET=8 };</pre>	JAUNE ⇔ 4 ORANGE ⇔ 6 VIOLET ⇔ 8
<pre>enum couleursNGris{ BLANC=9, GRIS, NOIR=14 };</pre>	BLANC ⇔ 9 GRIS ⇔ 10 ⇔ BLANC+1 NOIR ⇔ 14

Types Construits / Enumérations

Syntaxe	
<pre>typedef enum boolean{ FALSE, TRUE }boolean; boolean b ; b=TRUE; printf("Valeur de b : %d\n", b) ; while(b!=FALSE){ if(test()==3) b = FALSE; }</pre>	<p>Enfin les outils pour définir le type booléen qui n'existe pas en Langage C !</p> <p>Définition d'une variable b de type boolean</p> <p>Initialisation avec la valeur TRUE (\Leftrightarrow 1)</p> <p>Valeur de b: 1</p> <p>On suppose qu'on dispose d'une fonction test qui renvoie un entier</p> <p>Valeur de b: 0</p>

Types Construits / Enumérations

Syntaxe	
<pre>typedef enum boolean{ TRUE=1, FALSE=0 }boolean; boolean b ; b=FALSE; printf("Valeur de b : %d\n", b) ; while(b!=TRUE){ if(test()==2) b = TRUE; }</pre>	<p>Enfin les outils pour définir le type booléen qui n'existe pas en Langage C !</p> <p>Définition d'une variable b de type boolean</p> <p>Initialisation avec la valeur FALSE (\Leftrightarrow 0)</p> <p>Valeur de b: 0</p>

Types Construits / Enumérations

Enumérations/ illustration

```
typedef enum couleur{VERT,ORANGE,ROUGE}    couleurFeu;

couleurFeu    Feu;

....
Feu = ORANGE ;
printf("Couleur du feu : %d\n", Feu) ;

switch(Feu){
    case VERT:    printf("Passez\n");
                  break;
    case ORANGE:  printf("Ralentissez\n");
                  break;
    case ROUGE:   printf("Arrêtez vous\n");
                  break;
}
```

Couleur du feu : 1
Ralentissez

COURS 8

Programmation impérative

Entrées- sorties sur fichiers

- FILE
- Séquence Type
- Opérations d'écriture dans fichier
- Opérations de lecture à partir de fichier

SOMMAIRE

- Informations pratiques
- Introduction
- Éléments de base
 - Programmer en Langage C – Compilation
 - Structure d'un programme / Règles d'écritures
 - Types de base
 - Constantes/Variables
 - Opérateurs
 - Instructions de contrôle
 - Pointeurs
 - Tableaux
- Fonctions
- Chaînes de caractères
- Pointeurs- Tableaux-Fonctions
- Types Construits
- Entrées – Sorties sur Fichiers
- Compilation séparée
- Implémentation de Types Abstraits de Données

Entrées – Sorties sur Fichiers

- Les fichiers permettent de conserver de façon permanente des informations dans la mémoire de masse d'un ordinateur.
- Ce sont des objets qui sont manipulés de manière linéaire, du début jusqu'à la fin.
- `FILE` est le type d'un fichier en Langage C
 - Défini dans la librairie `<stdio.h>`
 - ex de déclaration
`FILE *fich;`

Entrées – Sorties sur Fichiers

- Séquence type d'utilisation de fichiers
 1. Ouverture d'un fichier
 2. Séquence d'opérations sur le fichier (lire, écrire, ajouter)
 3. Fermeture du fichier

Entrées – Sorties sur Fichiers/ Ouverture

fopen

```
FILE *fopen(char *chemin, char *modeAcces);
```

Renvoie NULL si erreur lors de la tentative d'ouverture du fichier spécifié dans **chemin**

Modes d'accès pour les fichiers de type **texte**

- **"r"** ⇔ lecture

positionnement en début de fichier s'il existe

- **"w"** ⇔ écriture

positionnement en début de fichier s'il existe ou création

- **"a"** ⇔ ajout

positionnement en fin de fichier ou création

Modes d'accès pour les fichiers de type **binaire**

- **"rb"** ⇔ lecture

positionnement en début de fichier s'il existe

- **"wb"** ⇔ écriture

positionnement en début de fichier s'il existe ou création

- **"ab"** ⇔ ajout

positionnement en fin de fichier ou création

Entrées – Sorties sur Fichiers/ Ouverture

fopen	
<pre>FILE *fopen(char *chemin, char *modeAcces);</pre>	
<pre>FILE *fich ; fich=fopen("d :\\tmp\\toto.txt", "r");</pre>	Ouverture en mode lecture sous windows du fichier toto.txt situé dans le répertoire tmp de la partition d du disque
<pre>FILE *fich ; fich=fopen("/users/toto.txt", "r");</pre>	Ouverture en mode lecture sous Unix du fichier toto.txt situé dans le répertoire users situé sous la racine
<pre>FILE *fich ; fich = fopen("toto.txt", "r") ;</pre>	Ouverture en mode lecture sous windows ou Unix du fichier toto .txt dans le répertoire courant

Entrées – Sorties sur Fichiers/ Fermeture

fopen	
<pre>int fclose (FILE * fic);</pre>	<p>ferme le fichier pointé par fic</p> <p>Si fermeture correcte fclose renvoie 0 Sinon EOF</p> <p>EOF - End Of File ⇔ constante définie comme la fin de fichier,</p> <p>EOF est insérée à la fin du fichier lors de la fermeture du fichier</p>

Entrées – Sorties sur Fichiers/ Fermeture

Séquence Type Ouverture / Fermeture de Fichier

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

int main () {
    FILE *Fic = NULL ;
    char NomFichier[200];
    char code;

    do{
        printf("Entrez le chemin du fichier à ouvrir?\n");
        fflush(stdout);
        scanf("%s", NomFichier);
        Fic = fopen(NomFichier, "w");
        if(Fic == NULL){
            printf("Erreur à l'ouverture du fichier %s\n", NomFichier) ;
        }
    }while(Fic==NULL);
    printf("Le fichier %s correctement ouvert en mode écriture\n", NomFichier);
    //..... traitement .....

    code=fclose(Fic);
    if(code==EOF)
        printf("Le fichier %s n'a pu être fermé correctement\n", NomFichier);
    else
        printf("Le fichier %s a correctement été fermé\n", NomFichier);
    return EXIT_SUCCESS ;
}
```

Entrez le chemin du fichier à ouvrir?

toto.txt

Le fichier toto.txt correctement ouvert en mode écriture

Le fichier toto.txt a correctement été fermé

Entrées – Sorties sur Fichiers

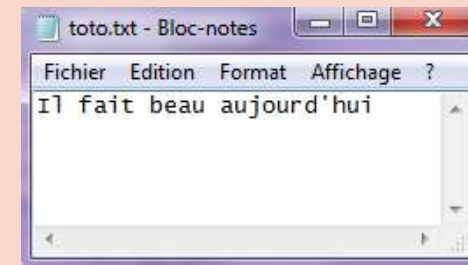
Opérations Lecture/écriture	
<code>char fgetc(FILE *fic);</code>	Opération de lecture d'un caractère dans le fichier fic renvoie le caractère suivant de la position courante du marqueur du fichier pointé par fic ou EOF si la fin du fichier atteinte.
<code>char fputc(char c, FILE *fic);</code>	Opération d'écriture d'un caractère c dans le fichier fic retourne le caractère c écrit ou EOF en cas d'erreur.

Entrées – Sorties sur Fichiers/ Fermeture

Exemple de programme de lecture de caractères à partir d'un fichier

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

int main () {
    FILE *Fic = NULL ;
    char  NomFichier[200];
    char code, c;
    //ouverture du fichier
    do{
        printf("Entrez le chemin du fichier à ouvrir?\n");
        fflush(stdout);
        scanf("%s", NomFichier);
        Fic = fopen(NomFichier, "r");
        if(Fic == NULL){
            printf("Erreur à l'ouverture du fichier %s\n", NomFichier) ;
        }
    }while(Fic==NULL);
    //Opérations de traitement sur le fichier
    while((c = fgetc(Fic)) != EOF)
        printf("%c",c) ;
    //fermeture du fichier
    code=fclose(Fic);
    if(code==EOF)
        printf("\nLe fichier %s n'a pu être fermé correctement\n", NomFichier);
    else
        printf("\nLe fichier %s a correctement été fermé\n", NomFichier);
    return EXIT_SUCCESS ;
}
```



```
Entrez le chemin du fichier à ouvrir?
toto.txt
Il fait beau aujourd'hui
Le fichier toto.txt a correctement été fermé
```

Entrées – Sorties sur Fichiers

Opérations Lecture avec formatage

```
int fscanf(FILE *fic, char * format, arguments);
```

Lit des données dans un **fichier texte** ouvert en mode **lecture**, selon le format spécifié dans la chaîne de caractères **format** et les stocke dans les variables précisées dans la liste **arguments**.

Retourne le nombre de valeurs correctement lues.

Lecture de deux entiers séparés par un espace, puis 1 caractère retour à la ligne.

```
FILE *fic ;  
int    a,b ;  
//séquence ouverture de fichier  
fscanf(fic, "%d %d\n", &a, &b) ;
```



Entrées – Sorties sur Fichiers

Opérations écriture avec formatage

```
int fprintf(FILE *fic, char *format, arguments);
```



écrit les données dans la liste d'**arguments** dans un **fichier texte** ouvert en mode **écriture**, selon le format spécifié dans la chaîne de caractères

format

Retourne le nombre de valeurs correctement écrites.

Ecriture de deux entiers séparés par un espace, puis 1 caractère retour à la ligne.

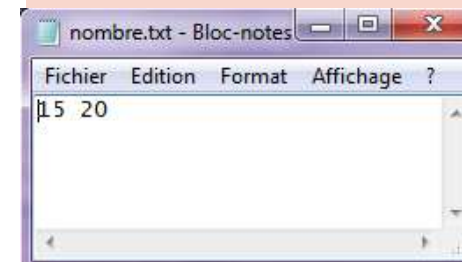
```
FILE *fic ;  
int    a=10,b=15 ;  
//séquence ouverture de fichier  
fprintf(fic, "%d %d\n", a, b) ;
```


Entrées – Sorties sur Fichiers

Exemple de programme d'écriture formatée d'entiers dans un fichier texte

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

int main () {
    FILE *fic = NULL ;
    char    NomFichier[200];
    int a=15,b=20;
    char code;
    //ouverture du fichier
    do{
        printf("Entrez le chemin du fichier à ouvrir?\n");
        fflush(stdout);
        scanf("%s", NomFichier);
        fic = fopen(NomFichier, "w");
        if(fic == NULL){
            printf("Erreur à l'ouverture du fichier %s\n", NomFichier) ;
        }
    }while(fic==NULL);
    //Opérations de traitement sur le fichier
    fprintf(fic,"%d %d\n",a,b);
    //fermeture du fichier
    code=fclose(fic);
    if(code==EOF)
        printf("\nLe fichier %s n'a pu être fermé correctement\n", NomFichier);
    else
        printf("\nLe fichier %s a correctement été fermé\n", NomFichier);
    return EXIT_SUCCESS ;
}
```

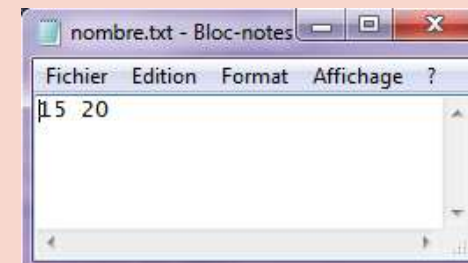


Entrées – Sorties sur Fichiers

Exemple de programme de lecture formatée d'entiers dans un fichier texte

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

int main () {
    FILE *fic = NULL ;
    char  NomFichier[200];
    int a,b;
    char code;
    //ouverture du fichier
    do{
        printf("Entrez le chemin du fichier à ouvrir?\n");
        fflush(stdout);
        scanf("%s", NomFichier);
        fic = fopen(NomFichier, "r");
        if(fic == NULL){
            printf("Erreur à l'ouverture du fichier %s\n", NomFichier) ;
        }
    }while(fic==NULL);
    //Opérations de traitement sur le fichier
    fscanf(fic,"%d %d\n",&a,&b);
    printf("a: %d b:%d\n", a, b);
    //fermeture du fichier
    code=fclose(fic);
    if(code==EOF)
        printf("\nLe fichier %s n'a pu être fermé correctement\n", NomFichier);
    else
        printf("\nLe fichier %s a correctement été fermé\n", NomFichier);
    return EXIT_SUCCESS ;
}
```



```
Entrez le chemin du fichier à ouvrir?
nombre.txt
a: 15 b:20

Le fichier nombre.txt a correctement été fermé
```

Entrées – Sorties sur Fichiers

Opérations Lecture en bloc

```
int fread(type * ptr, int tailleObj, int nbObjet, FILE *fic);
```



Lit en bloc des données dans un **fichier binaire** pointé par **fic**, au maximum **nbObjet** de taille **tailleObj** et les place dans le tableau **ptr**.

Retourne le nombre de valeurs correctement lues.

Entrées – Sorties sur Fichiers

Opérations Ecriture en bloc

```
int fwrite(type * ptr, int tailleObj, int nbObjet, FILE *fic);
```



Ecrit en bloc **nbObjet** données de taille **tailleObj** du tableau **ptr**, dans un **fichier binaire** pointé par **fic**.

Retourne le nombre de valeurs correctement écrites.

Entrées – Sorties sur Fichiers

Exemple de programme d'écriture en bloc dans un fichier binaire

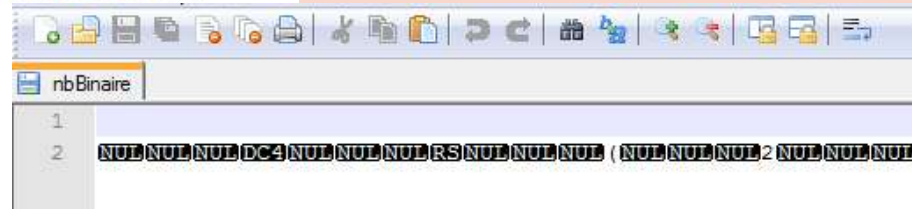
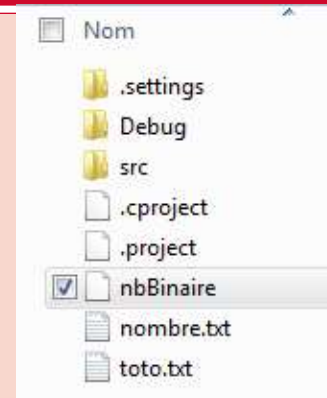
```
#include <stdio.h>
#include <stdlib.h>

#define NB 5

int main () {
    FILE *fic = NULL ;
    char    NomFichier[200];

    int tab[NB]={10,20,30,40,50};
    char code;
    //ouverture du fichier
    do{
        printf("Entrez le chemin du fichier à ouvrir?\n");
        fflush(stdout);
        scanf("%s", NomFichier);
        fic = fopen(NomFichier, "wb");
        if(fic == NULL){
            printf("Erreur à l'ouverture du fichier %s\n", NomFichier) ;
        }
    }while(fic==NULL);
    //opérations de traitement sur le fichier
    fwrite(tab,sizeof(int),NB, fic);

    //fermeture du fichier
    code=fclose(fic);
    if(code==EOF)
        printf("\nLe fichier %s n'a pu être fermé correctement\n", NomFichier);
    else
        printf("\nLe fichier %s a correctement été fermé\n", NomFichier);
    afficheTab(tab, NB);
    return EXIT_SUCCESS ;
}
```



Entrées – Sorties sur Fichiers

Exemple de programme de lecture en bloc dans un fichier binaire

```
int main () {
    FILE *fic = NULL ;
    char    NomFichier[200];

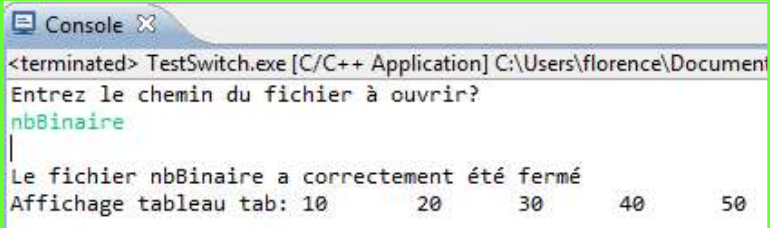
    int tab[NB];
    char code;
    //ouverture du fichier
    do{
        printf("Entrez le chemin du fichier à ouvrir?\n");
        fflush(stdout);
        scanf("%s", NomFichier);
        fic = fopen(NomFichier, "rb");
        if(fic == NULL){
            printf("Erreur à l'ouverture du fichier %s\n", NomFichier) ;
        }
    }while(fic==NULL);
    //opérations de traitement sur le fichier
    fread(tab,sizeof(int),NB, fic);

    //fermeture du fichier
    code=fclose(fic);
    if(code==EOF)
        printf("\nLe fichier %s n'a pu être fermé correctement\n", NomFichier);
    else
        printf("\nLe fichier %s a correctement été fermé\n", NomFichier);
    afficheTab(tab, NB);
    return EXIT_SUCCESS ;
}

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

#define NB 5

void afficheTab(int t[], int nb){
    int i;
    printf("Affichage tableau tab: ");
    for(i=0;i<nb;i++)
        printf("%d\t",t[i]);
    printf("\n");
}
```



```
Console X
<terminated> TestSwitch.exe [C/C++ Application] C:\Users\florence\Document
Entrez le chemin du fichier à ouvrir?
nbBinaire
Le fichier nbBinaire a correctement été fermé
Affichage tableau tab: 10    20    30    40    50
```

Entrées – Sorties sur Fichiers

Opérations diverses

```
int fseek(FILE *fic, long offset, int mode);
```



positionnement du marqueur de fichier

fseek fait pointer le pointeur de fichier, associé au flux *fic*, *offset* octets *plus loin*.

mode peut prendre les valeurs 0, 1, 2 selon que le déplacement doit être :

- 0 : par rapport au début du fichier
- 1 : par rapport à la position courante
- 2 : par rapport à la fin du fichier.

Entrées – Sorties sur Fichiers

fopen	
<code>long int ftell(FILE *fic);</code>	retourne la position du marqueur de fichier ftell et fseek sont surtout utiles pour gérer des fichiers avec des structures
<code>void rewind(FILE *fic);</code>	met le marqueur en début de fichier
<code>int remove(char * fileName);</code>	destruction du fichier de nom <code>fileName</code>
<code>int rename(char * oldFileName, char * newFileName);</code>	change le nom du fichier de nom <code>oldfile</code> par le nom <code>newFileName</code>