

# ALGORITHMIQUE AVANCE LANGAGE C

TCSRIT LICENCE 1



KONAN HYACINTHE



## A- Introduction aux Langages de programmation

L'ordinateur est une machine non intelligente, qui sait exécuter des opérations très simples :

- Opérations arithmétiques : addition, soustraction, multiplication...
- Opérations de comparaisons.
- Lecture de valeurs et affichage de résultats.

Se pose alors la question très simple : où réside l'importance de l'ordinateur ? La réponse est qu'il peut être programmé : c'est-à-dire qu'on peut lui donner, à l'avance, la séquence (suite ordonnée) des instructions (ordres) à effectuer l'une après l'autre. La grande puissance de l'ordinateur est sa **rapidité**. Mais, c'est à l'homme (**programmeur**) de tout faire et de tout prévoir, l'ordinateur ne fait qu'exécuter des ordres codés en binaire (langage machine). C'est pour cela que des langages dits « évolués » ont été mis au point pour faciliter la programmation, exemples : FORTRAN, COBOL, BASIC, PASCAL...et bien évidemment, le **langage C** qui favorise (tout comme le Pascal) une approche méthodique et disciplinée, on parle de méthode « **structurée** », ce langage a vu le jour dans les Laboratoires BELL en 1978, et il a été créé par KERNIGHAM et RITCHIE pour développer le système d'exploitation **UNIX**.

Le C est un langage compilé, c'est-à-dire qu'il faut passer par les étapes suivantes :

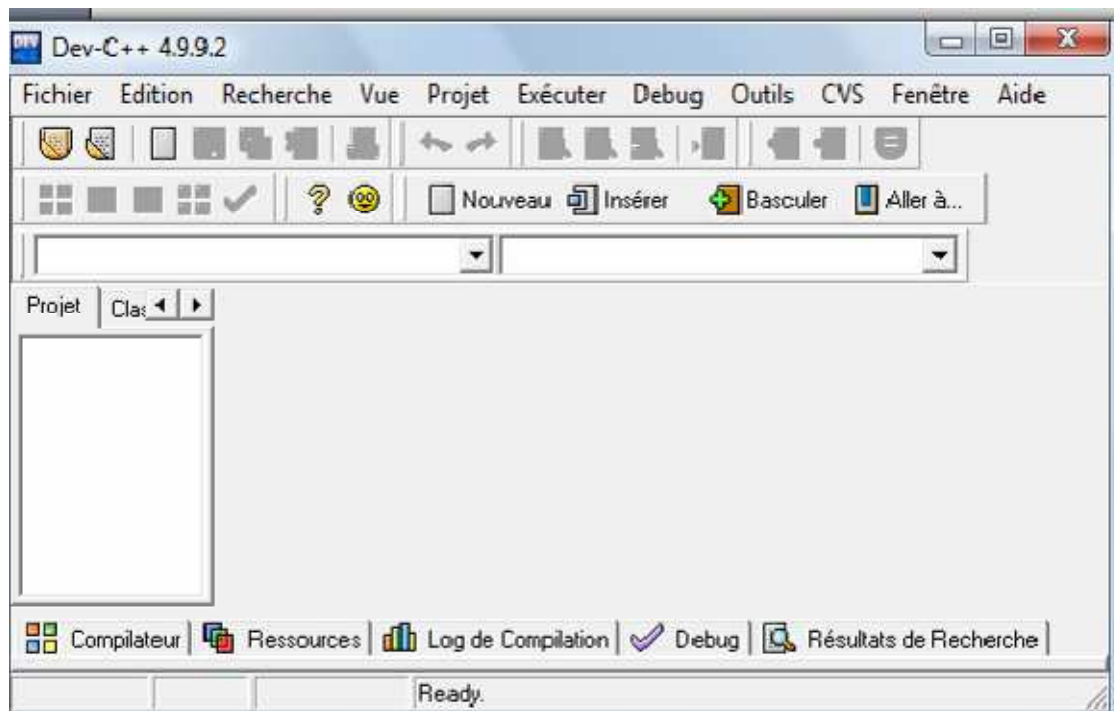
1. Entrer le programme ou le code source dans l'ordinateur à l'aide d'un **EDITEUR** (DEV C++ par exemple qui est disponible gratuitement sur Internet).
2. Traduire ce programme en langage machine (codes binaires compréhensibles par l'ordinateur), c'est la **compilation**.
3. **Exécuter** le programme et visualiser le résultat.

## B- Premier programme en C

### Remarque :


Tous les programmes ont été écrits à l'aide de l'éditeur (ou plutôt **EDI** Environnement de Développement Intégré) **DEV C++** (ver 4.9.9.2) qui est téléchargeable gratuitement de l'Internet ([www.bloodshed.net/dev/devcpp.html](http://www.bloodshed.net/dev/devcpp.html)), la taille du fichier < 9 Mo.


Une fois installée sur votre ordinateur, vous pouvez lancer l'application DEV C++, à partir du bouton *Démarrer*, *Programmes*, *Bloodshed Dev C++* et enfin *DEV C++* pour obtenir l'interface suivante :

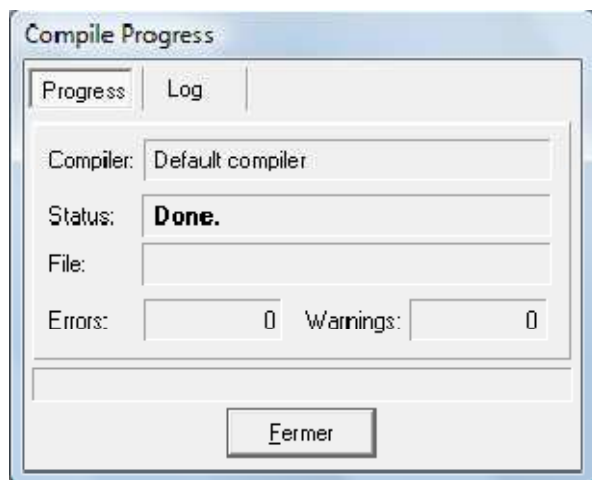



Pour écrire votre premier programme en C, vous choisissez le menu *Fichier, Nouveau ; Fichier Source*. Tapez au clavier les instructions suivantes :

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    // Ceci est un commentaire
    // Déclaration de la variable entière a
    int a;
    // Affectation de la valeur 10 à la variable a
    a=10;
    // Affichage de la variable a
    printf("%d\n",a);
    // Affichage d'un texte
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

A présent, il est temps d'enregistrer le fichier source (code) à l'aide du menu *Fichier, ensuite Sauvegarder* (ou plus simplement le bouton  ou encore CTRL S), nommez-le **exemple1** (et choisissez l'emplacement en parcourant l'arborescence de votre disque).

Maintenant, vous devez compiler votre programme à l'aide du bouton  de la barre d'outils (ou CTRL F9). Si tout se passe bien, c'est à dire pas d'erreurs, vous obtenez :



Dernière étape, il ne vous reste plus qu'à exécuter votre programme ; à l'aide du bouton  de la barre d'outils (ou CTRL F10) :



### Commentaires :

- 1- Dans le programme précédent, il y a des directives du style : **#include** permettant d'inclure les bibliothèques, c'est-à-dire, les fichiers déclarant les **fonctions standard**, par exemple :
  - **stdio** (**S**Tan**D**ard **I**nput **O**utput) : qui fait le lien entre le programme et la console (clavier/écran). Dans notre cas, c'est la fonction `printf` (affichage à l'écran).
  - **stdlib** : qui permet, entre autre, la gestion dynamique de la mémoire (`malloc` et `free` - voir plus loin).
- 2- La fonction principale du programme appelée « **main** » qui représente le point d'entrée du programme où débutera son exécution.
- 3- Le bloc **d'instructions** délimité par les accolades { ... } et comportant :
  - Des commentaires : `// ...commentaire` ou `/* commentaire */`.
  - Des déclarations de variables : dans notre cas, `a` est une variable entière. Une variable représente une case mémoire au niveau de la RAM de l'ordinateur que l'on se réserve pour notre programme.
  - Chaque instruction se termine par ;
  - Des affectations, dans notre cas : `a = 10` ; signifie que la valeur 10 est stockée (en mémoire) au niveau de la variable `a`.
  - Des affichages de variables ou de texte au niveau de la console à l'aide de `printf`. Le `\n` signifie que l'on retourne à la ligne après affichage.
  - La dernière instruction : `system("pause")` ; qui à l'origine est une commande du MS-DOS/Windows (pause) permettant l'affichage du message « Appuyez sur une touche pour continuer » et met le système en attente de la frappe d'une touche du clavier, sinon, la console d'exécution se ferme automatiquement à la fin de l'exécution d'un programme ne laissant guère le temps de visualiser les éventuels résultats affichés. Cette dernière instruction peut être remplacée par `getch()` ; invitant l'utilisateur à taper une touche et mettant ainsi la console en attente !

## C- Syntaxe du Langage C

### 1) Les Types de données de C

Les variables existantes en C sont de 2 types : scalaires (entiers, réels, caractères) et agrégés (combinaison de scalaires, comme les tableaux, les matrices...).

#### Variable de type scalaire

C'est une variable ne contenant qu'une seule valeur sur laquelle on peut faire un calcul arithmétique. Il existe 3 types de base : **char**, **int** et **float**.

Une variable de type `char` est codée sur 1 octet (8 bits).

Une variable de type `int` ou **short** `int` est codée sur 16 bits et **long** `int` sur 32 bits.

Une variable de type `float` représente un réel flottant, c'est-à-dire un nombre stocké en deux parties, une mantisse et un exposant. La taille de la mantisse définit le nombre de chiffres significatifs, alors que la taille de l'exposant définit le plus grand nombre acceptable par la machine. `float` est codé sur 32 bits et il existe le réel double précision de type **double** codé sur 64 bits.

En résumé :

Type	Taille (en bits)	Plage de valeurs
<code>char</code>	8	-128 à +127
<code>short (short int)</code>	16	-32768 à 32767
<code>long (long int)</code>	32	-2.147.483.648 à 2.147.483.647
<code>float</code>	32	-3.4e38 à 3.4e38 (7 chiffres significatifs)
<code>double (long float)</code>	64	-1.7e308 à 1.7e308 (15 chiffres significatifs)

#### Les conversions :

- D'abord implicites, c'est-à-dire automatiques et toujours de la plus petite à la plus grande précision :

`short int` → `int` → `long int`

`float` → `double` → `long double`

- Ensuite explicites, par exemple :

`int a ; float b ;`

On convertit à l'aide de : `(float)a` ou `float(a)` et `(int)b` ou `int(b)`

## 2) Déclaration et stockage de variables, constantes...

Une variable **doit** être définie, dans un programme, par une déclaration : on indique le nom que l'on désire lui donner, son type (int, float, char...) pour que le compilateur sache quelle taille en mémoire il doit lui réserver et les opérateurs qui peuvent lui être associés, mais aussi comment elle doit être gérée (visibilité, durée de vie,...).

### Les constantes

**Exemple** : #define PI 3.14 ou const float pi=3.14;

```
#include <stdio.h>
#include <stdlib.h>
#define PI 3.14 /* PI est une constante=3,14 */
// on peut aussi utiliser : const float PI=3.14;
int main ()
{
    float P,R;
    printf("Entrez la valeur du rayon : ");
    scanf("%f", &R);
    P=2*PI*R;
    printf("Le perimetre du cerle est : %f\n",P);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

### La Déclaration de variables simples

**Exemple** :

```
int i ; /* la variable i est déclarée comme étant entière */
int compteur = 0 ; /* on déclare et on initialise la variable */
float x, y ;
double VOLUME ; /* Attention, le C fait la différence entre minuscule et majuscule */
char touche_clavier ;
```

### Déclarations locales

Dans tout bloc d'instructions {...}, avant la première instruction, on peut déclarer des variables. Elles seront alors "**locales** au bloc d'instructions" : elles n'existent qu'à l'intérieur du bloc (on parle alors de la **portée** de ces variables qui est limitée au bloc ainsi que de la **durée de vie** de ces mêmes variables qui sont en fait créées à l'entrée du bloc et libérées à sa sortie).

### Déclarations globales

Une déclaration faite à l'extérieur d'un bloc d'instructions {...}, (en général en début du fichier) est dite **globale**. La variable est stockée en mémoire statique, sa durée de vie est celle du programme. Elle est visible de sa déclaration jusqu'à la fin du fichier.

### Déclaration de type

De nouveaux types de variables peuvent être déclarés à l'aide de **typedef**. Cela permet de donner un nom à un type donné, mais ne crée aucune variable.

### Exemple :

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    typedef float prix;
    typedef float quantite;
    typedef float total;
    prix a, b;
    quantite x, y;
    total z, t;
    printf("Entrez les prix a et b : ");
    scanf("%f %f", &a, &b);
    printf("Entrez les quantites x et y : ");
    scanf("%f %f", &x, &y);
    z = a * x;
    t = b * y;
    printf("Le total1 est : %f et Le total2 est : %f\n", z, t);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

```
Entrez les prix a et b : 10 20
Entrez les quantites x et y : 2 4
Le total1 est : 20.000000 et Le total2 est : 80.000000
Fin du travail
Appuyez sur une touche pour continuer...
```

### 3) Les fonctions d'entrée/sortie les plus utilisées en C

Comme le langage C a été créé pour être indépendant du matériel sur lequel il est implanté, les entrées sorties, bien que nécessaires à tout programme, ne font pas partie intégrante du langage, c'est pour cela qu'on fait appel à des **bibliothèques** de fonctions de base, standardisées au niveau des compilateurs. En voici les bibliothèques principales :

- Entrée et sortie de données : **<stdio.h>**
- Traitement de chaînes de caractères : **<string.h>**
- Fonctions d'aide générales - gestion de la mémoire : **<stdlib.h>**
- Fonctions arithmétiques : **<math.h>** ...

Comme on vient de voir, la bibliothèque standard **<stdio.h>** contient un ensemble de fonctions qui assurent la communication entre l'ordinateur et l'extérieur. En voici les plus importantes :

#### **printf()**

La fonction printf est utilisée pour **afficher** ou transférer du texte, ou des valeurs de variables vers le fichier de sortie standard stdout (par défaut l'écran).

### Exemple1 :

Ecrivez la suite d'instructions dans un fichier source « exemple1 » :

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int a = 2;
    int b = 4;
    printf("%d fois %d est %d\n", a, b, a*b);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

```
2 fois 4 est 8
Fin du travail
Appuyez sur une touche pour continuer...
```

### Exemple2 :

Taper dans un fichier source « exemple2 » le code ci-dessous, constatez que les numéros de lignes ont été affichés ! (Outils, Options de l'Editeur et Affichage...) :

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    float x = 10.1234;
    double y = 20.1234567;
    printf("%f\n", x);
    printf("%f\n", y);
    printf("%e\n", x);
    printf("%e\n", y);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

```
10.123400
20.123457
1.012340e+001
2.012346e+001
Fin du travail
Appuyez sur une touche pour continuer...
```

En Résumé, voici les spécificateurs de format de printf :

SYMBOLE	TYPE	IMPRESSION COMME
%d ou %i	int	entier relatif
%u	int	entier naturel (unsigned)
%c	int	caractère
%x	int	Entier en notation hexadécimale
%f	double	rationnel en notation décimale
%e ou %E	double	rationnel en notation scientifique (exponentielle)
%s	char*	chaîne de caractères

### scanf()

La fonction scanf est la fonction symétrique à printf, mais au lieu d'afficher, elle permet de **lire** des variables; elle fonctionne pratiquement comme printf, mais en sens inverse. Cette fonction reçoit des données à partir du fichier d'entrée standard stdin (le clavier par défaut), ces données sont mémorisées aux adresses indiquées par &.

### Exemple :

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int jour, mois, annee;
    scanf("%d %d %d", &jour, &mois, &annee);
    printf("le jour est: %d le mois est: %d l'annee est: %d\n", jour, mois, annee);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

A l'exécution :

```
3 9 2009
le jour est: 3 le mois est: 9 l'annee est: 2009
Fin du travail
Appuyez sur une touche pour continuer... _
```

### putchar()

La commande putchar ('a') **affiche** le caractère « a » à l'écran (stdout), l'argument de cette fonction est bien évidemment un caractère.

### Exemple :

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    char a = 225;
    char b = 'e';
    putchar('x'); /* afficher la lettre x */
    putchar('\n'); /* retour à la ligne */
    putchar('?'); /* afficher le symbole ? */
    putchar('\n');
    putchar(65); /* afficher le symbole avec */
    /* le code 65 (ASCII: 'A') */
    putchar('\n');
    putchar(a); /* afficher la lettre avec */
    /* le code 225 (ASCII: 'ß') */
    putchar('\n');
    putchar(b); /* affiche le contenu de la var b 'e' */
    putchar('\n');
    printf("\nFin du travail\n"); /* 2 retours à la ligne! */
    system("pause");
    return 0;
}
```

```
x
?
A
ß
e

Fin du travail
Appuyez sur une touche pour continuer...
```

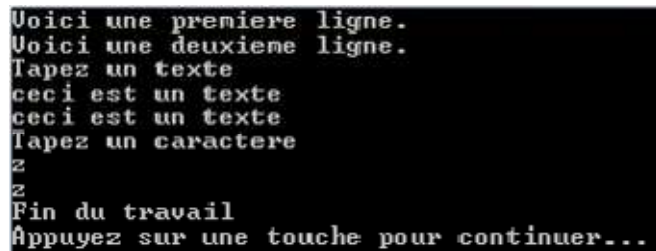
### getchar() - puts(chaine) et gets(chaine)

- La fonction getchar, plus utilisée que putchar, permet de **lire** le prochain caractère du fichier d'entrée standard stdin.
- La fonction puts(chaine) affiche, à l'écran, la chaîne de caractères « chaîne » puis positionne le curseur en début de ligne suivante.
- La fonction gets(chaine) lecture d'une chaîne sur stdin. Tous les caractères peuvent être entrés, y compris les blancs. La saisie est terminée par un retour chariot.



**Exemple :**

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{ char a ;
    char TEXTE[] = "Voici une premiere ligne.";
    puts(TEXTE);
    puts("Voici une deuxieme ligne.");
    printf("Tapez un texte\n");
    gets(TEXTE);
    puts(TEXTE);
    //printf("Tapez un caractere\n");
    a = getchar();
    putchar(a);
    putchar('\n');
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```



```
Voici une premiere ligne.
Voici une deuxieme ligne.
Tapez un texte
ceci est un texte
ceci est un texte
Tapez un caractere
z
z
Fin du travail
Appuyez sur une touche pour continuer...
```

#### 4) Expressions, Opérateurs (arithmétiques, relationnels, affectations...)

##### Affectation

On sait faire déjà : variable = expression ;

Exemple : i = 3 ; c = 'z' ;

Affectations multiples : a = b = c = 0 ;

Il existe en C une manière optimale d'affectation ne faisant évaluer la variable qu'une seule fois, par exemple : i = i+1 (2 fois !) peut s'écrire i += 1 (une seule fois).

**Exemple :**

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int i; i=4; i+=1;
    printf("i = %d\n",i);
    i-=1;
    printf("i = %d\n",i);
    i*=2;
    printf("i = %d\n",i);
    i++; // très utile dans les boucles for
    printf("i = %d\n",i);
    i--;
    printf("i = %d\n",i);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

```
i = 5
i = 4
i = 8
i = 9
i = 8
Fin du travail
Appuyez sur une touche pour continuer...
```

Voici en résumé :

Les pré incrémentations - décrémentations :

$++i \Leftrightarrow i+=1 \Leftrightarrow i = i+1$

$--i \Leftrightarrow i-=1 \Leftrightarrow i = i-1$

Dans ces cas, la valeur de l'expression est la valeur de la variable après l'opération.

Les post incrémentations - décrémentations :

$i++ \Leftrightarrow i+=1 \Leftrightarrow i = i+1$

$i-- \Leftrightarrow i-=1 \Leftrightarrow i = i-1$

Dans ces cas, la valeur de l'expression est la valeur de la variable avant l'opération.

### Expressions

Une expression est un calcul qui donne une valeur résultat.

Exemple :  $2 + 4$ .

Une expression comporte des variables, des appels de fonction et des constantes combinés entre eux par des opérateurs.

Exemple : `Ma_Valeur*cos(Mes_Angle*PI/90)`.

Pour former une expression, les opérateurs possibles sont très nombreux :

#### Opérateurs arithmétiques

Ces opérateurs s'appliquent à des valeurs entières ou réelles. Ces opérateurs sont :

**+** (addition), **-** (soustraction), **\*** (produit), **/** (division), **%** (reste de la division).

#### Opérateurs relationnels

#### Comparaisons :

Ces opérateurs sont : **=** (égalité), **!=** (différent), **<**, **>**, **<=**, **>=**. Des deux côtés du signe opératoire, il faut deux opérandes de même type (sinon, transformation implicite). **Attention**, le compilateur ne prévient pas on met **=** (affectation) au lieu de **==** !!!

#### Logique booléenne

Le résultat est toujours 0 (faux) ou 1 (vrai).

Ces opérateurs sont : **&&** (ET), **||** (OU) et **!** (NON). (on verra l'utilisation plus loin).

Voici le tableau des priorités :

Priorité 1 (la plus forte):	()
Priorité 2:	! ++ --
Priorité 3:	* / %
Priorité 4:	+ -
Priorité 5:	< <= > >=
Priorité 6:	== !=
Priorité 7:	&&
Priorité 8:	
Priorité 9 (la plus faible):	= += -= *= /= %=

**Exemple :** « Résolution de l'équation du 1er degré :  $ax + b = 0$  »

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    float a, b, x;
    printf("Entrez les valeurs de a et b : ");
    scanf("%f %f",&a,&b);
    if (a!= 0) {
        x=-b/a;
        printf("le resultat est : %f\n",x);
    }
    else
        if (b==0) printf("Tout reel est solution\n");
        else printf("Pas de solution\n");
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

```
Entrez les valeurs de a et b : -2 4
le resultat est : 2.000000
Fin du travail
Appuyez sur une touche pour continuer...
```

```
Entrez les valeurs de a et b : 0 0
Tout reel est solution
Fin du travail
Appuyez sur une touche pour continuer...
```

```
Entrez les valeurs de a et b : 0 5
Pas de solution
Fin du travail
Appuyez sur une touche pour continuer...
```

### Fonctions mathématiques standard

Ces fonctions sont prédéfinies dans la bibliothèque standard **<math.h>**, un programme faisant appel à ces fonctions doit contenir la ligne : **#include <math.h>**

En voici la liste :

Commande C	Explication
exp(X)	fonction exponentielle
log(X)	logarithme naturel
log10(X)	logarithme à base 10
pow(X,Y)	X exposant Y
sqrt(X)	racine carrée de X
fabs(X)	valeur absolue de X
floor(X)	arrondir en moins
ceil(X)	arrondir en plus
fmod(X,Y)	reste rationnel de X/Y (même signe que X)
sin(X) cos(X) tan(X)	sinus, cosinus, tangente de X
asin(X) acos(X) atan(X)	arcsin(X), arccos(X), arctan(X)
sinh(X) cosh(X) tanh(X)	sinus, cosinus, tangente hyperboliques de X

### Exemple1 :

Tapez le programme suivant :

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main ()
{
    double A, B, RES;
    /* Saisie de A et B */
    printf("Introduire la valeur pour A : ");
    scanf("%lf", &A);
    printf("Introduire la valeur pour B : ");
    scanf("%lf", &B); RES = A*A;
    /* Affichage du résultat */
    printf("Le carré de A est %f \n", RES);
    /* Calcul */
    RES = B*B;
    /* Affichage du résultat */
    printf("Le carré de B est %f \n", RES);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

```
Introduire la valeur pour A : 5
Introduire la valeur pour B : 10
Le carré de A est 25.000000
Le carré de B est 100.000000
Fin du travail
Appuyez sur une touche pour continuer...
```

### Exemple2 :

Modifiez le programme précédent de façon à ce qu'il affiche :

- $A^B$  ( $A$  à la puissance  $B$ ),
- L'hypoténuse d'un triangle rectangle de côtés  $A$  et  $B$ ,
- La tangente de  $A$  en n'utilisant que les fonctions sin et cos,
- La valeur arrondie (en moins) de  $A/B$ ,
- La valeur arrondie (en moins) à trois positions derrière la virgule de  $A/B$ .

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main ()
{
    double A, B, RES;
    printf("Introduire les valeurs de A et B : ");
    scanf("%lf %lf", &A, &B); RES = pow(A,B);
    printf("A exposant B est %e \n", RES); // pour l'affichage decimal : %f
    RES = sqrt(pow(A,2)+pow(B,2));
    printf("hypotenuse du triangle rectangle est %f \n", RES); RES = sin(A)/cos(A);
    printf("La tangente de A est %f \n", RES); RES = floor(A/B);
    printf("La valeur arrondie en moins de A/B est %f \n", RES);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

```
Introduire les valeurs de A et B : 23 ?
A exposant B est 3.404825e+009
hypotenuse du triangle rectangle est 24.041631
La tangente de A est 1.588153
La valeur arrondie en moins de A/B est 3.000000
Fin du travail
Appuyez sur une touche pour continuer...
```

## 5) Structures de contrôle

### Boucles ou structures répétitives

Les boucles sont très importantes en programmation puisqu'elles permettent de répéter plusieurs fois un bloc d'instructions, et c'est ainsi que l'ordinateur révèle toute sa puissance et sa rapidité.

Il existe, en C, 3 types de boucles :

#### 1- Boucle for (pour)

Cette boucle est utile quand on connaît à l'avance le nombre d'itérations à effectuer. Sa structure est la suivante :

**for (expression initiale; expr\_condition; expr\_incrémentation) instruction**

#### Exemple1 :

Ecrivez un programme qui affiche les nombres de 1 à 10 :

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int i;
    for (i=1; i<=10; i++) // i++ ou i=i+1
    { // { : est facultative comme il n'y a qu'une seule instruction!
        printf("%d ",i);
    }
    printf("\nFin du travail\n");
    system("pause");
    return 0;
}
```

```
1 2 3 4 5 6 7 8 9 10
Fin du travail
Appuyez sur une touche pour continuer...
```

Modifiez le programme précédent pour qu'il affiche les chiffres à la verticale.

#### 2- While (tant que)

**while (expression) instruction ;**

Tant que l'expression ou la condition est vraie ( $\neq 0$ ), on effectue l'instruction, soit une seule instruction (terminée par ;), soit un bloc d'instructions (entre {}). La condition est au moins évaluée une fois. Tant qu'elle est vraie, on effectue l'instruction, dès qu'elle est fausse, on passe à l'instruction suivante (si elle est fausse dès le début, l'instruction n'est jamais effectuée, on ne rentre jamais dans la boucle).

#### Exemple :

Ecrivez un programme qui permet de **saisir** un entier N au clavier et qui **calcule** et **affiche** la somme :

$S = 1 + 2 + \dots + N$ .

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int N, S, i=1;
    printf("Saisir le nombre N : ");
    scanf("%d",&N);
    S=0;
    while (i<=N)
    {
        S=S+i;
        i++;
    }
    printf("La somme S = %d\n",S);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

```
Saisir le nombre N : 4
La somme S = 10
Fin du travail
Appuyez sur une touche pour continuer...
```

### 3- Do While (faire tant que)

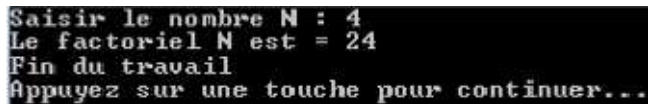
**do** instruction **while** (expression) ;

Cette structure est comme le while, mais à la différence fondamentale que l'instruction est au moins faite une fois.

#### Exemple1 :

Ecrivez un programme qui permet de **saisir** un entier N au clavier et qui **calcule** et **affiche** le factoriel N (N !) :  
 produit :  $P = 1 \times 2 \times \dots \times N$ .

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int N, P, i=1;
    printf("Saisir le nombre N : ");
    scanf("%d",&N);
    P=1;
    do
    {
        P=P*i;
        i++;
    } while (i<=N);
    printf("Le factoriel N est = %d\n",P);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

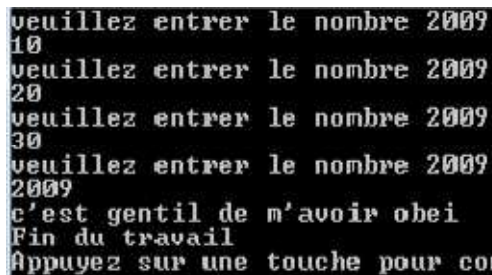


```
Saisir le nombre N : 4
Le factoriel N est = 24
Fin du travail
Appuyez sur une touche pour continuer...
```

#### Exemple2 :

Tapez le programme suivant :

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int a;
    do
    {
        puts("veuillez entrer le nombre 2009");
        scanf("%d",&a);
    }
    while (a!=2009);
    puts("c'est gentil de m'avoir obei");
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```



```
veuillez entrer le nombre 2009
10
veuillez entrer le nombre 2009
20
veuillez entrer le nombre 2009
30
veuillez entrer le nombre 2009
2009
c'est gentil de m'avoir obei
Fin du travail
Appuyez sur une touche pour continuer...
```

Exercice récapitulatif sur les boucles :

Ecrivez un programme en C qui **lit** N valeurs « **réelles** » saisies au clavier et qui **calcule** et **affiche** la **somme** S et la **moyenne** M de ces valeurs.

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int i,N;
    float S,M,V;
    S=0;
    printf("entrez le nombre de valeurs N : ");
    scanf("%d",&N);
    printf("entrez le N valeurs : ");
    for (i=1;i<=N;i++)
    {
        scanf("%f",&V);
        S=S+V;
    }
    M=S/N;
    printf("la somme des N valeurs est S = %f\n",S);
    printf("la moyenne des N valeurs est M = %f\n",M);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

```
entrez le nombre de valeurs N : 5
entrez le N valeurs : 1 2 3 4 5
la somme des N valeurs est S = 15.000000
la moyenne des N valeurs est M = 3.000000
Fin du travail
Appuyez sur une touche pour continuer...
```

### Structures alternatives

On a souvent besoin de n'effectuer certaines instructions que si des **conditions** sont remplies. On dispose pour cela du **IF** et du **SWITCH**.

#### 1- If - Else (Si - Sinon)

**if** (expression) instruction ; l'instruction peut être simple ou un bloc { ... }.

**if** (expression) instruction1 **else** instruction2 ;

#### Exemple1 :

Ecrivez un programme en C qui **saisi** (ou lit) 2 valeurs au clavier, les **compare** et **affiche** la plus grande valeur.

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    float A,B,MAX;
    printf("Entrez les 2 valeurs A et B : ");
    scanf("%f %f",&A,&B);
    if (A>B) MAX = A;
    else MAX = B;
    printf("La plus grande des 2 valeurs est : %f\n",MAX);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

```
Entrez les 2 valeurs A et B : 10 20
La plus grande des 2 valeurs est : 20.000000
Fin du travail
Appuyez sur une touche pour continuer...
```

### Exemple2 :

Ecrivez un programme qui **lit** 2 valeurs A et B, les **échange** si A > B de manière à les **afficher** dans l'ordre croissant.

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    float A,B,AUX;
    printf("Entrez les 2 valeurs A et B : ");
    scanf("%f %f",&A,&B);
    if (A>B)
    {
        AUX=B;
        B=A;
        A=AUX; // Echange de A et B
    }
    printf("La valeurs trieés : %f %f\n",A,B);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```



```
Entrez les 2 valeurs A et B : 8 4
La valeurs trieés : 4.000000 8.000000
Fin du travail
Appuyez sur une touche pour continuer...
```

### Switch - Case (brancher - dans le cas)

**switch** (expression\_entière)

```
{
    case cstel:instructions
    case cstel2:instructions
    .....
    case cstelN:instructions
    default :instructions
}
```

### Exemple :

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int a;
    printf("Entrez la valeur de a : ");
    scanf("%d",&a);
    switch(a)
    {
        case 1:printf("a vaut 1\n");
        break;
        case 2:printf("a vaut 2\n");
        break;
        case 3:printf("a vaut 3\n");
        break;
        default:printf("a non compris entre 1 et 3\n");
        break;
    }
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```



```
Entrez la valeur de a : 2
a vaut 2
Fin du travail
Appuyez sur une touche pour continuer...
```

```
Entrez la valeur de a : 10
a non compris entre 1 et 3
Fin du travail
Appuyez sur une touche pour continuer...
```

## 6) Structures de données avancées

### Tableaux et matrices

#### a- Tableaux ou vecteurs :

Déclaration : type nom [nombre\_d'éléments];

Exemple : int tab[10];

Ceci réserve en mémoire un espace pouvant contenir 10 entiers. Le premier est tab[0], jusqu'à tab[9] !!! (Source d'erreurs non signalée par le compilateur !).

On peut aussi déclarer un tableau par typedef :

typedef float vecteur[3] ;

vecteur x, y, z ;

On peut aussi initialiser un tableau lors de sa déclaration :

int T[4] = { 10,20,30,40} ;

Comme on peut ne pas spécifier la dimension du tableau :

int Y[ ] = {3,5,7} ;

#### Exemple :

Dans cet exemple, on va **sommer 2 vecteurs** et **afficher** le vecteur résultat :

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int i;
    typedef int vecteur[4];
    vecteur U={ 1,2,3,4},V={ 1,2,3,4},W;
    for(i=0;i<=3;i++)
    {
        W[i]=U[i]+V[i];
        printf("%d ",W[i]);
    }
    printf("\nFin du travail\n");
    system("pause");
    return 0;
}
```

```
2 4 6 8
Fin du travail
Appuyez sur une touche pour continuer...
```

## b- Matrices :

Déclaration : type nom [nombre\_lignes] [nombre\_colonnes];

Exemple : float M[3][4] ;

### Exemple :

Dans cet exemple, on va **sommer 2 matrices** et **afficher** le résultat :

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int i,j;
    typedef int matrice[2][2]; //2 lignes 2 colonnes
    matrice A={1,2,3,4},B={5,6,7,8},C;//1,2 1ère ligne, 3,4 2ème ligne
    for(i=0;i<=1;i++)
    {
        for(j=0;j<=1;j++)
        {
            C[i][j]=A[i][j]+B[i][j];
            printf("%5d",C[i][j]); //de l'espace pour afficher
        }
        printf("\n"); printf("\n");
    }
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

```

  6    8
 10   12
Fin du travail
Appuyez sur une touche pour continuer...
```

## Pointeurs

Les pointeurs sont des variables contenant des **adresses**.

Exemple1 commenté :

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int i=17,j;
    int *p; //déclaration du pointeur
    p=&i; //p pointe sur l'adresse de i
    j=*p; //c'est la même chose que j = i;
    printf("j = %d\n",j);
    printf("p = %d\n",*p);
    *p=22;
    printf("p = %d\n",*p);
    printf("i = %d\n",i);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

```

j = 17
p = 17
p = 22
i = 22
Fin du travail
Appuyez sur une touche pour continuer...
```

## Exemple2 commenté :

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int A = 1, B = 2;
    int *P1;
    P1=&A;
    printf("%d\n",P1);
    printf("%d\n",A);
    printf("%d\n",*P1);
    *P1=3;
    printf("%d\n",A);
    P1=&B;
    printf("%d\n",*P1);
    (*P1)++;
    printf("%d\n",B);
    system("pause");
    return 0;
}
```

```
2293572
1
1
3
2
3
Appuyez sur une touche pour continuer...
```

### Allocation dynamique d'une variable et libération de la mémoire

Quand une variable est déclarée dans un programme, Le système d'exploitation se charge d'allouer (ou réserver) automatiquement de l'espace mémoire suffisant pour cette variable. En revanche, On peut demander manuellement de l'espace mémoire au S.E., c'est ce qu'on appelle de l'**allocation dynamique de mémoire**.

### Taille des variables

On a déjà vu que selon le type de variable (char, int, double, float...), on a besoin de plus ou moins de mémoire. Exemple, un « char » tient sur 1 octet, un « int » sur 4 octets...

Pour vérifier cela, on dispose d'une fonction en C : sizeof(type de données).

Exemple :

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    printf("char : %d octets\n", sizeof(char));
    printf("int : %d octets\n", sizeof(int));
    printf("long : %d octets\n", sizeof(long));
    printf("float : %d octets\n", sizeof(float));
    printf("double : %d octets\n", sizeof(double));
    system("pause");
    return 0;
}
```

```
char : 1 octets
int : 4 octets
long : 4 octets
double : 4 octets
double : 8 octets
Appuyez sur une touche pour continuer...
```

Peut-on afficher la taille d'un type personnalisé qu'on a créé, un tableau par exemple ? Cela est possible grâce toujours à sizeof(tableau) :

### Exemple :

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int T[3]; // 3 x taille de int = 12 octets
    printf("Le tableau T de 3 int : %d octets\n", sizeof(T));
    system("pause");
    return 0;
}
```

```
Le tableau T de 3 int : 12 octets
Appuyez sur une touche pour continuer... _
```

Donc, en suivant ce même principe, la déclaration suivante :

int tableau[100]; fait occuper 4 x 100 = 400 octets en mémoire, même si le tableau est vide !

### Allocation dynamique de la mémoire

Pour pouvoir allouer de façon dynamique la mémoire, on a besoin d'inclure la bibliothèque <stdlib.h> qui contient 2 fonctions :

- **malloc** ("Memory ALLOcation", c'est-à-dire "Allocation de mémoire") : demande au système d'exploitation la permission d'utiliser de la mémoire.
- **free** ("Libérer") : permet d'indiquer au S.E. que l'on n'a plus besoin de la mémoire qu'on avait demandée. La place en mémoire est libérée, un autre programme peut maintenant s'en servir au besoin.

### Exemple :

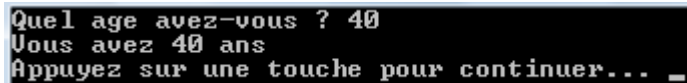
On va écrire un programme en C qui demande l'âge à l'utilisateur et qui va le lui afficher. La variable utilisée va être allouée manuellement (ou dynamiquement) au lieu d'automatiquement par le S.E.

```
#include<stdio.h>
#include<stdlib.h> // nécessaire pour utiliser malloc et free
int main()
{
    int *memoireAllouee = NULL; // On crée un pointeur sur int car malloc renvoie void* (un pointeur)
    memoireAllouee = malloc(sizeof(int)); // Allocation de la mémoire
    //test du pointeur : si l'allocation a marché, le pointeur contient une adresse,
    // sinon il contient l'adresse NULL
    if (memoireAllouee == NULL) // Si l'allocation a échoué
    {
        exit(0); // On arrête immédiatement le programme
    }
    // Utilisation de la mémoire
    printf("Quel age avez-vous ? ");
    scanf("%d", memoireAllouee);
    printf("Vous avez %d ans\n", *memoireAllouee);
    free(memoireAllouee); // Libération de mémoire
    system("pause");
    return 0;
}
```

```
Quel age avez-vous ? 40
Vous avez 40 ans
Appuyez sur une touche pour continuer... _
```

Si on opte pour l'allocation automatique de la mémoire (# à dynamique), le programme précédent est équivalent à :

```
#include<stdio.h>
#include<stdlib.h>// Facultatif
int main()
{
    int age = 0; // Allocation de la mémoire (automatique) par le S.E
    // Utilisation de la mémoire
    printf("Quel age avez-vous ? ");
    scanf("%d",&age);
    printf("Vous avez %d ans\n",age);
    system("pause");
    return 0;
} // Libération de la mémoire (automatique à la fin de la fonction) par le S.E
```



### Allocation dynamique d'un tableau

La vraie utilité de l'allocation dynamique est pour créer un tableau dont on ne connaît pas la taille avant l'exécution du programme.

Exemple :

On veut écrire un programme qui stocke l'âge de tous les amis de l'utilisateur dans un tableau, on peut utiliser : `int ageAmis[15]`; le problème est que l'on ne sait pas à l'avance le nombre d'amis de l'utilisateur, on ne le saura qu'à l'exécution du programme.

Voici une solution :

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int nombreDAmis = 0, i = 0;
    int* ageAmis = NULL; // Ce pointeur va servir de tableau après l'appel du malloc
    // On demande le nombre d'amis à l'utilisateur
    printf("Combien d'amis avez-vous ? ");
    scanf("%d", &nombreDAmis);
    if (nombreDAmis > 0) // Il faut qu'il ait au moins un ami
    {
        ageAmis = malloc(nombreDAmis * sizeof(int)); // On alloue de la mémoire pour le tableau
        if (ageAmis == NULL) // On vérifie si l'allocation a marché ou pas
        {
            exit(0); // On arrête tout
        }
        // On demande l'âge des amis un à un
        for (i = 0 ; i < nombreDAmis ; i++)
        {
            printf("Quel age a l'ami numero %d ? ", i + 1);
            scanf("%d", &ageAmis[i]);
        }
        // On affiche les âges stockés un à un
        printf("\n\nVos amis ont les ages suivants :\n");
        for (i = 0 ; i < nombreDAmis ; i++) { printf("%d ans\n", ageAmis[i]); }
        // On libère la mémoire allouée avec malloc, on n'en a plus besoin
        free(ageAmis);
    }
    system("pause");
    return 0;
}
```

```
Combien d'amis avez-vous ? 3
Quel age a l'ami numero 1 ? 35
Quel age a l'ami numero 2 ? 40
Quel age a l'ami numero 3 ? 45

Vos amis ont les ages suivants :
35 ans
40 ans
45 ans
Appuyez sur une touche pour continuer...
```

### Structures

Jusque là, nous avons toujours utilisés des variables contenant les mêmes types de données (tableaux ou matrices d'entiers ou réels...), les structures sont des variables composées de plusieurs variables (ou **CHAMPS**) de types différents :

#### Exemple :

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    struct point
    {
        int x;
        int y; //x et y sont les champs de la struct point
    };
    struct point a, b; //déclaration de variables de type point
    a.x=1;
    a.y=2;
    b.x=3;
    b.y=4;
    printf("la valeur de l'abscisse de a est : %d\n",a.x);
    printf("la valeur de l'ordonnee de b est : %d\n",b.y);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

```
la valeur de l'abscisse de a est : 1
la valeur de l'ordonnee de b est : 4
Fin du travail
Appuyez sur une touche pour continuer...
```

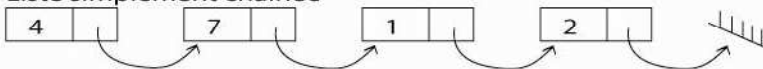
### Structures et pointeurs – Listes chaînées

La manière la plus simple et intuitive de créer un programme utilisant des conteneurs est les tableaux. Lorsqu'on crée un tableau, les éléments de celui-ci sont placés de façon « structurée » en mémoire. Pour pouvoir créer ce tableau, il faut connaître sa taille. Pour supprimer un élément au milieu du tableau, il faut recopier les éléments temporairement, ré-allouer de la mémoire pour le tableau, puis le remplir à partir de l'élément supprimé. Bref, ce sont beaucoup de manipulations coûteuses en ressources. Une liste chaînée est différente dans le sens où les éléments de la liste sont répartis dans la mémoire et reliés entre eux par des pointeurs. On peut ajouter et enlever des éléments d'une liste chaînée à n'importe quel endroit, à n'importe quel instant, sans devoir recréer la liste toute entière :

Tableau standard

4	7	1	2
---	---	---	---

Liste simplement chaînée



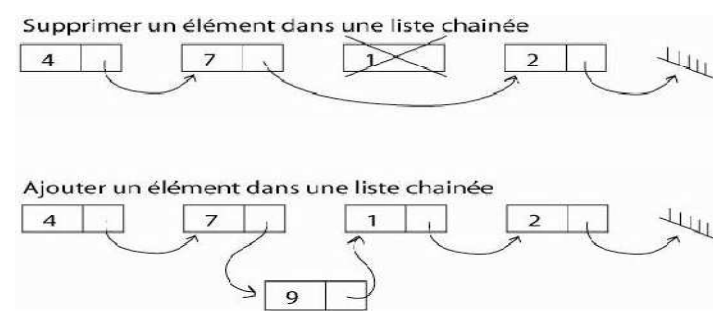
On a sur le schéma ci-dessus la représentation que l'on pourrait faire d'un tableau et d'une liste chaînée. Chacune de ces représentations possède ses avantages et inconvénients. C'est lors de l'écriture d'un programme qu'on doit se poser la question de savoir laquelle des deux méthodes est la plus intéressante.

- Dans un tableau, la taille est connue, l'adresse du premier élément aussi. Lorsqu'on déclare un tableau, la variable contiendra l'adresse du premier élément du tableau.
- Comme le stockage est contigu, et la taille de chacun des éléments connue, il est possible d'atteindre directement la case i d'un tableau.
- Pour déclarer un tableau, il faut connaître sa taille.
- Pour supprimer ou ajouter un élément à un tableau, il faut créer un nouveau tableau et supprimer l'ancien...
- Dans une liste chaînée, la taille est inconnue au départ, la liste peut avoir autant d'éléments que la mémoire le permet.
- Il est en revanche impossible d'accéder directement à l'élément i de la liste chaînée. Pour ce faire, il faut traverser les i-1 éléments précédents de la liste.
- Pour déclarer une liste chaînée, il suffit de créer le pointeur qui va pointer sur le premier élément de la liste chaînée, aucune taille n'est donc à spécifier.
- Il est possible d'ajouter, de supprimer, d'intervir des éléments d'une liste chaînée sans avoir à recréer la liste en entier, mais en manipulant simplement leurs pointeurs.

Chaque élément d'une liste chaînée est composé de deux parties :

- la valeur qu'on veut stocker,
- l'adresse de l'élément suivant, s'il existe. S'il n'y a plus d'élément suivant, alors l'adresse sera NULL, et désignera le bout de la chaîne.

Voilà deux schémas pour expliquer comment se passent l'ajout et la suppression d'un élément d'une liste chaînée. Remarquons le symbole en bout de chaîne qui signifie que l'adresse de l'élément suivant ne pointe sur rien, c'est-à-dire sur NULL.



### Exemple :

On considère l'exemple suivant de parcours et d'affichage d'une liste chaînée (qui se termine par 0) :

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    struct elem
    {
        int val;//valeur entière
        elem *suiv;//pointeur sur elem
    };

    typedef elem *ptrelem;//pointeur sur elem
    ptrelem L=NULL;//L est la liste, vide au départ
    int x;
    scanf("%d",&x);
    while (x!=0)
    {
        L=new elem;//allocation
        L->val=x;//on met x dedans
        printf(" La valeur du pointeur : %d\n",L->val);
        L=L->suiv;
        scanf("%d",&x);
    }
    printf("\nFin du travail\n"); system("pause");
    return 0;
}
```

```
Tapez la suite 0 pour finir :
1
La valeur du pointeur : 1
2
La valeur du pointeur : 2
3
La valeur du pointeur : 3
0
Fin du travail
Appuyez sur une touche pour continuer...
```

## 7) Fonctions

Une fonction est définie par son entête, suivie d'un bloc d'instructions

**Déclaration : type\_retourné nom\_fonction(liste\_arguments ou paramètres).**

Exemple : `int carre (int x) { ... }` : `carre` est une fonction qui retourne un entier et qui a comme paramètre un entier.

Si la fonction ne retourne rien on indique : **void**. (c'est l'équivalent des procédures en PASCAL).

Exemples : - `float nom_fonction(int a, float b) {bloc d'instructions}`

- `void nom_fonction() {bloc d'instructions}` : fonction sans paramètre et ne retournant rien (void).

### Exemple 1:

```
#include <stdio.h>
#include <stdlib.h>
void debut() // debut retourne rien
{ printf("message de debut\n");
}
void fin(int a) // a paramètre formel
{ printf("valeur finale = %d\n",a);
}
int carre(int x)
{
    int v;
    v=x*x;
    return v; // carre retourne un entier
}
int main ()
{
    int i,total;
    debut();
    total=0;
    for(i=1;i<=2;i++)
        total=total+carre(i);
    fin(total);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

```
message de debut
valeur finale = 5
Fin du travail
Appuyez sur une touche pour continuer...
```

### Passage des paramètres d'une fonction

**1- Par référence** : L'argument ou (paramètre effectif) remplace le paramètre formel et toute modification de la valeur du paramètre (à l'intérieur de la fonction) **modifie** la valeur de l'argument. Il suffit de mettre **&** devant le nom du paramètre pour que le passage soit par référence. Si le paramètre formel est un tableau le passage se fait toujours par référence.

**2- Par valeur** : l'argument est recopié dans une variable locale et le paramètre désigne cette variable. Toute modification de la valeur du paramètre modifie la variable locale. **L'argument n'est pas modifié.**



## Exemple2 :

```
#include <stdio.h>
#include <stdlib.h>
void compte1(int i)//passage par valeur
{
    i++;
    printf("La valeur du parametre de compte1 : %d\n",i);
}
void compte2(int &j)//passage par référence
{
    j++;
    printf("La valeur du parametre de compte2 : %d\n",j);
}
void compte3(int t[3]) //tableau donc par référence
{
    t[1]++;
    printf("La valeur du parametre de compte3 : %d\n",t[1]);
}
int main ()
{
    int a,b,r[3];
    a=b=r[1]=1;//initialisations multiples
    compte1(a);
    printf("La valeur de l'argument de compte1 : %d\n",a);
    compte2(b);
    printf("La valeur de l'argument de compte2 : %d\n",b);
    compte3(r);
    printf("La valeur de l'argument de compte3 : %d\n",r[1]);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

```
La valeur du parametre de compte1 : 2
La valeur de l'argument de compte1 : 1
La valeur du parametre de compte2 : 2
La valeur de l'argument de compte2 : 2
La valeur du parametre de compte3 : 2
La valeur de l'argument de compte3 : 2
Fin du travail
Appuyez sur une touche pour continuer...
```

## 8) Fichiers de données

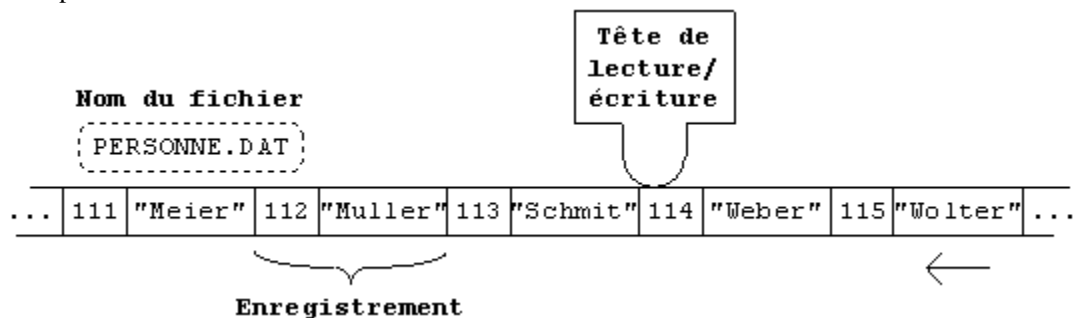
### Notion de fichier

Un fichier (file en anglais) est un ensemble structuré de données stocké en général sur un support externe (clé USB, disque dur, disque optique, bande magnétique, ...).

Fichier séquentiel.

Dans des fichiers séquentiels, les enregistrements sont mémorisés consécutivement dans l'ordre de leur entrée et peuvent seulement être lus dans cet ordre. Si on a besoin d'un enregistrement précis dans un fichier séquentiel, il faut lire tous les enregistrements qui le précèdent, en commençant par le premier.

Exemple :



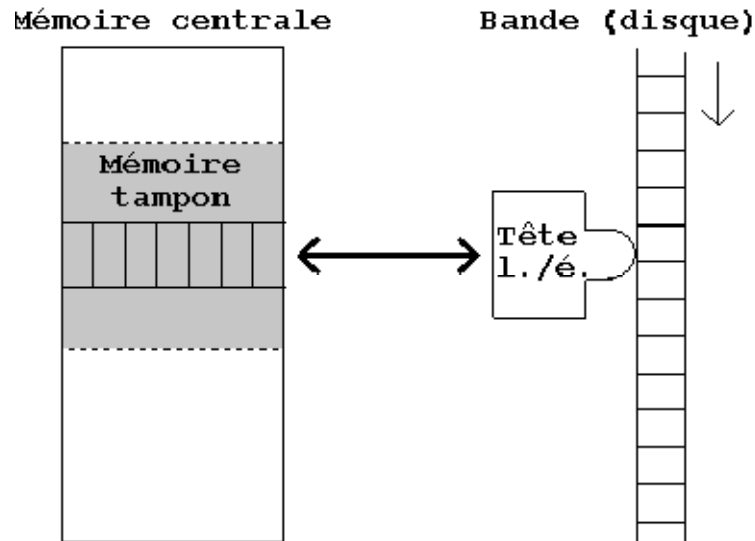
Il existe 2 fichiers spéciaux qui sont définis par défaut pour tous les programmes :

- stdin le fichier d'entrée standard (clavier)
- stdout le fichier de sortie standard (l'écran).

Cela signifie que les programmes lisent leurs données au clavier et écrivent les résultats sur l'écran.

### **La mémoire tampon (buffer)**

Pour des raisons d'efficacité, les accès à un fichier se font par l'intermédiaire d'une mémoire tampon (buffer en anglais). La mémoire tampon est une zone de la mémoire centrale de l'ordinateur réservée à un ou plusieurs enregistrements du fichier. L'utilisation de la mémoire tampon a l'effet de réduire le nombre d'accès à un périphérique de stockage d'une part et le nombre des mouvements de la tête de lecture/écriture d'autre part :



### **Ouverture et fermeture de fichiers**

L'ouverture et la fermeture de fichiers se font à l'aide des fonctions `fopen` et `fclose` définies dans la bibliothèque standard `<stdio>`.

La commande `fopen` permet d'ouvrir des fichiers en écriture ou en lecture :

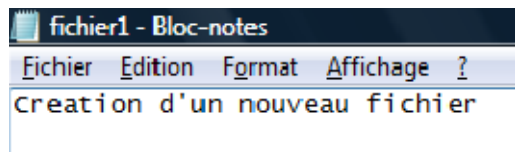
**`fopen (<Nom> , "w" ) : écriture, fopen (<Nom> , "r" ) : lecture.`**

Ouverture en écriture :

- Dans le cas de la création d'un nouveau fichier, le nom du fichier est ajouté au répertoire de stockage.
- Si un fichier existant est ouvert en écriture, alors son contenu est perdu.
- Si un fichier non existant est ouvert en écriture, alors il est créé automatiquement. Si la création du fichier est impossible alors `fopen` indique une erreur en retournant la valeur zéro.

### Exemple1 :

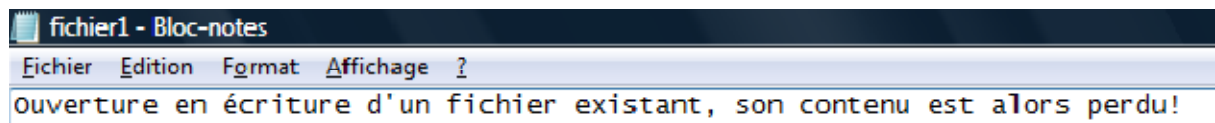
L'exemple suivant permet la création du fichier : « fichier1.txt » à la racine du disque « d : », et insère le texte : « Création d'un nouveau fichier » dans ce fichier :



```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    FILE *index;//déclaration de fichiers
    int x;
    index = fopen("d:\\fichier1.txt","w");//ouverture en écriture
    if (index == NULL) printf("Probleme d'ouverture\n");
    else printf("Pas de probleme d'ouverture\n");
    fputs("Creation d'un nouveau fichier", index);
    fclose(index);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

### Exemple 2 :

Cet exemple nous montre que l'ouverture en écriture d'un fichier existant a pour effet de faire perdre son contenu : Voici le contenu du fichier précédent ouvert en écriture :



```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    FILE *index;//déclaratio de fichiers
    int x;
    index = fopen("d:\\fichier1.txt","w");//ouverture en écriture
    if (index == NULL) printf("Probleme d'ouverture\n");
    else printf("Pas de probleme d'ouverture\n");
    fputs("Ouverture en écriture d'un fichier existant, son contenu est alors perdu!", index);
    fclose(index);
    printf("Fin du travail\n");
    system("pause");
    return 0;
}
```

### Exemple 3 :

Le programme suivant contrôle si l'ouverture du fichier a été réalisée avec succès, d'abord l'ouverture en écriture :

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    FILE *index; /* pointeur sur FILE */
    char nom_fichier[30]; /* nom du fichier */
    do
    {
        printf("Entrez le nom du fichier : ");
        scanf("%s", nom_fichier);
        index = fopen(nom_fichier, "w");
        if (!index)
            printf("\aERREUR: Impossible d'ouvrir le fichier: %s.\n", nom_fichier);
    } while (!index);
    fclose(index);
    system("pause");
    return 0;
}
```

### Exemple 4 :

Ensuite l'ouverture en lecture :

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    FILE *index; /* pointeur sur FILE */
    char nom_fichier[30]; /* nom du fichier */
    do
    {
        printf("Entrez le nom du fichier : ");
        scanf("%s", nom_fichier);
        index = fopen(nom_fichier, "r");
        if (!index)
            printf("\aERREUR: Impossible d'ouvrir le fichier: %s.\n", nom_fichier);
    } while (!index);
    fclose(index);
    system("pause");
    return 0;
}
```

### Lire et écrire dans des fichiers séquentiels

Les fichiers que nous manipulons ici sont des fichiers texte, cela signifie que toutes les informations dans les fichiers sont mémorisées sous forme de chaînes de caractères et sont organisées en lignes. Même les valeurs numériques (types int, float, double, ...) sont stockées comme chaînes de caractères.

Pour l'écriture et la lecture des fichiers, nous allons utiliser les fonctions standard fprintf, fscanf, fputc et fgetc qui correspondent à printf, scanf, putchar et getchar...

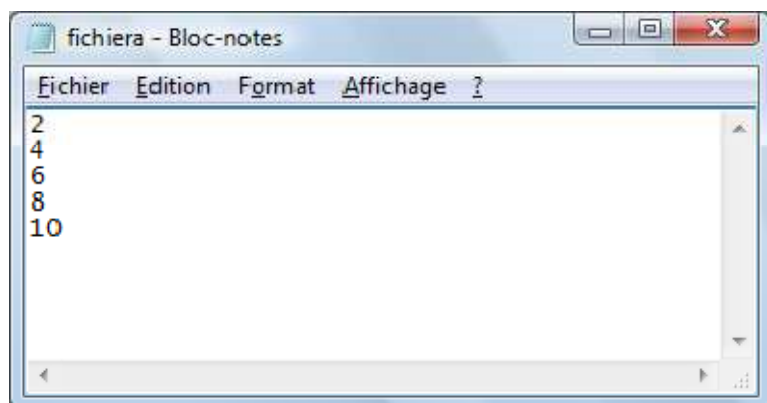
### Exemple 5 :

Ecrire un programme qui ouvre un fichier en écriture « w » et qui affiche dedans 5 entiers saisis au clavier. On utilisera la fonction : fprintf.

```
// Programme qui écrit 5 entiers dans un fichier ouvert en écriture
#include <stdio.h>
#include <stdlib.h>
main()
{
    FILE *index; /* pointeur sur FILE */
    char nom_fichier[30]; /* nom du fichier */
    int n;
    do
    {
        printf("Entrez le nom du fichier : ");
        scanf("%s", nom_fichier);
        index = fopen(nom_fichier, "w");
        if (!index)
            printf("\aERREUR: Impossible d'ouvrir le fichier: %s.\n", nom_fichier);
    } while (!index);
    printf("Saisir 5 entiers : ");
    for (int i=1; i<=5; i++)
    {
        scanf("%d", &n);
        fprintf(index, "%d\n", n);
    }
    fclose(index);
    system("pause");
    return 0;
}
```



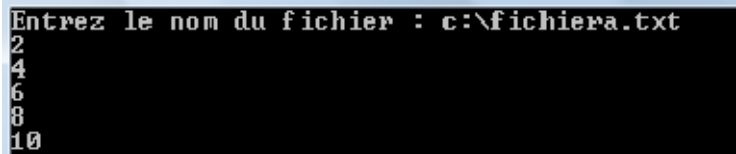
```
Entrez le nom du fichier : c:\fichier.txt
Saisir 5 entiers : 2
4
6
8
10
Appuyez sur une touche pour continuer...
```



### Exemple 6 :

Ecrire un programme qui ouvre, en lecture, le fichier créé par le programme précédent et qui lit et affiche les 5 entiers insérés dans ce fichier.

```
// Programme qui ouvre, en lecture, le fichier créé par le programme précédent
//et qui lit et affiche les 5 entiers insérés dans ce fichier
#include <stdio.h>
#include <stdlib.h>
main()
{
    FILE *index; /* pointeur sur FILE */
    char nom_fichier[30]; /* nom du fichier */
    int n;
    do
    {
        printf("Entrez le nom du fichier : ");
        scanf("%s", nom_fichier);
        index = fopen(nom_fichier, "r");
        if (!index)
            printf("\aERREUR: Impossible d'ouvrir le fichier: %s.\n", nom_fichier);
    } while (!index);
    for (int i=1; i<=5; i++)
    {
        fscanf(index, "%d", &n);
        printf("%d\n", n);
    }
    fclose(index);
    system("pause");
    return 0;
}
```



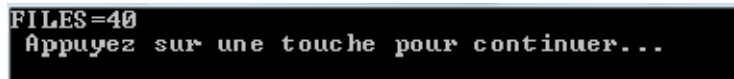
```
Entrez le nom du fichier : c:\fichier1.txt
2
4
6
8
10
```

### Exemple 7 :

Le programme suivant lit et affiche le fichier "C:\CONFIG.SYS" en le parcourant caractère par caractère:

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    FILE *FP;
    FP = fopen("C:\\CONFIG.SYS ", "r");
    if (!FP)
    {
        printf("Impossible d'ouvrir le fichier\n");
        exit(-1);
    }
    while (!feof(FP))
        putchar(fgetc(FP));
    fclose(FP);
    system("pause")
    return 0;
}
```

Dans une chaîne de caractères constante, il faut indiquer le symbole '\' (back-slash) par '\\', pour qu'il ne soit pas confondu avec le début d'une séquence d'échappement (p.ex: \n, \t, \a, ...).



```
FILES=40
Appuyez sur une touche pour continuer...
```

**FIN**