Table des matières

[Langage C 2](#_Toc434934585)

[Base 2](#_Toc434934586)

[Les caractères spéciaux 2](#_Toc434934587)

[Pour un commentaire court 2](#_Toc434934588)

[Votre commentaire est long 2](#_Toc434934589)

[Variables 4](#_Toc434934590)

[Les Opérations 6](#_Toc434934591)

[La bibliothèque mathématique 6](#_Toc434934592)

[Les conditions 8](#_Toc434934593)

[Construire un switch 10](#_Toc434934594)

[Les boucles 11](#_Toc434934595)

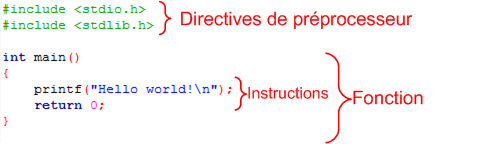
[La Boucle While 11](#_Toc434934596)

[La boucle do… while 12](#_Toc434934597)

[La boucle for. 13](#_Toc434934598)

# Langage C

# Base



printf est une instruction. Elle commande à l'ordinateur : « Affiche-moi ce message à l'écran ».

La console de Windows ne gère pas les accents.

# Les caractères spéciaux

* \n : retour à la ligne (= « Entrée ») ;
* \t : tabulation.

## Pour un commentaire court

Un double slash (//) suivi de votre commentaire. Par exemple :

// Ceci est un commentaire

## Votre commentaire est long

début du commentaire : tapez un slash suivi d'une étoile (/\*) ;

fin du commentaire : tapez une étoile suivie d'un slash (\*/).

Vous écrirez donc par exemple :

/\* Ceci est

un commentaire

sur plusieurs lignes \*/

Variables

Voici les principaux types de variables existant en langage C :

| **Nom du type** | **Minimum** | **Maximum** |
| --- | --- | --- |
| signed char | -127 | 127 |
| int | -32 767 | 32 767 |
| long | -2 147 483 647 | 2 147 483 647 |
| float | -1 x1037 | 1 x1037 |
| double | -1 x1037 | 1 x1037 |

|  |  |
| --- | --- |
| unsigned char | 0 à 255 |
| unsigned int | 0 à 65 535 |
| unsigned long | 0 à 4 294 967 295 |

Les trois premiers types (signed char, int, long) permettent de stocker des nombres entiers : 1, 2, 3, 4…  
Les deux derniers (float, double) permettent de stocker des nombres décimaux (appelés nombres flottants) : 13.8, 16.911…

Attention avec les nombres décimaux ! utilisé le point. Vous ne devez donc pas écrire 54,9 mais plutôt 54.9 !

unsigned sont des types qui ont le défaut de ne pas pouvoir stocker de nombres négatifs, mais l'avantage de pouvoir stocker des nombres deux fois plus grands

Pour un nombre entier, on utilisera le plus souvent int ;

Pour un nombre flottant, on utilisera généralement double.

si vous avez plusieurs variables du même type à déclarer Il vous suffit de séparer les différents noms de variables par des virgules sur la même ligne :

int nombreDeVies, niveau, ageDuJoueur;.

Cela créera trois variables int appelées nombreDeVies, niveau et ageDuJoueur.

Exemple de déclaration de constante :

const *int* NOMBRE\_DE\_VIES\_INITIALES = 5;

Ce « symbole spécial » '% suivi d'une lettre (dans mon exemple, la lettre 'd'). Cette lettre permet d'indiquer ce que l'on doit afficher. 'd' signifie que l'on souhaite afficher un int.

| **Format** | **Type attendu** |
| --- | --- |
| "%d" | int |
| "%ld" | long |
| "%f" | float |
| "%f" | double |

printf("Il vous reste %d vies", nombreDeVies);

Le %d sera remplacé par la variable indiquée après la virgule, à savoir nombreDeVies.

(Veillez à bien indiquer vos variables dans le bon ordre. Le premier %d sera remplacé par la première variable (nombreDeVies), et le second %d par la seconde variable (niveau). Si vous vous trompez d'ordre, votre phrase ne voudra plus rien dire.)

*int* main(*int* argc, *char* \*argv[])

{

*int* nombreDeVies = 5, niveau = 1;

printf("Vous avez %d vies et vous etes au niveau n° %d\n", nombreDeVies, niveau);

return 0;

}

Lorsque celui-ci arrive à un scanf, il se met en pause et attend que l'utilisateur entre un nombre. Ce nombre sera stocké dans la variable poids.

*int* main(*int* argc, *char* \*argv[])

{

*int* age = 0; // On initialise la variable à 0

printf("Quel age avez-vous ? ");

scanf("%d", &age); // On demande d'entrer l'âge avec scanf

printf("Ah ! Vous avez donc %d ans !\n\n", age);

return 0;

}

Quel age avez-vous ? 20

Ah ! Vous avez donc 20 ans !

# Les Opérations

| **Opération** | **Signe** |
| --- | --- |
| Addition | + |
| Soustraction | - |
| Multiplication | \* |
| Division | / |
| Modulo | % |

## La bibliothèque mathématique

#include <math.h>

Vous avez déjà utilisé les fonctions printf et scanf de la bibliothèque stdio.h.  
Il faut savoir qu'il existe une autre bibliothèque, appelée math.h, qui contient de nombreuses fonctions mathématiques toutes prêtes.

#### fabs

Cette fonction retourne la valeur absolue d'un nombre  
La valeur absolue d'un nombre est sa valeur positive :

* si vous donnez -53 à la fonction, elle vous renvoie 53 ;
* si vous donnez 53 à la fonction, elle vous renvoie 53.

En bref, elle renvoie toujours l'équivalent positif du nombre que vous lui donnez.

*double* absolu = 0, nombre = -27;

absolu = fabs(nombre); // absolu vaudra 27

Cette fonction renvoie un double, donc votre variable absolu doit être de type double.

(Il existe aussi une fonction similaire appelée abs, située cette fois dans stdlib.h.  
La fonction abs marche de la même manière, sauf qu'elle utilise des entiers (int). Elle renvoie donc un nombre entier de type int et non un double comme fabs.)

#### ceil

On arrondit en fait toujours au nombre entier supérieur.  
Par exemple, si on lui donne 26.512, la fonction renvoie 27.

Cette fonction s'utilise de la même manière et renvoie un double :

*double* dessus = 0, nombre = 52.71;

dessus = ceil(nombre); // dessus vaudra 53

#### floor

C'est l'inverse de la fonction précédente : elle renvoie le nombre directement en dessous.  
Si vous lui donnez 37.91, la fonction floor vous renverra donc 37.

#### pow

Cette fonction permet de calculer la puissance d'un nombre. Vous devez lui indiquer deux valeurs : le nombre et la puissance à laquelle vous voulez l'élever. Voici le schéma de la fonction :

pow(nombre, puissance);

Par exemple, « 2 puissance 3 » (que l'on écrit habituellement 2^3 sur un ordinateur), c'est le calcul 2 \* 2 \* 2, ce qui fait 8 :

*double* resultat = 0, nombre = 2;

resultat = pow(nombre, 3); // resultat vaudra 2^3 = 8

Vous pouvez donc utiliser cette fonction pour calculer des carrés. Il suffit d'indiquer une puissance de 2.

#### sqrt

Cette fonction calcule la racine carrée d'un nombre. Elle renvoie un double.

*double* resultat = 0, nombre = 100;

resultat = sqrt(nombre); // resultat vaudra 10

#### sin, cos, tan

Ce sont les trois fameuses fonctions utilisées en trigonométrie.  
Le fonctionnement est le même, ces fonctions renvoient un double.

Ces fonctions attendent une valeur en **radians**.

#### asin, acos, atan

Ce sont les fonctions arc sinus, arc cosinus et arc tangente, d'autres fonctions de trigonométrie.  
Elles s'utilisent de la même manière et renvoient un double.

#### exp

Cette fonction calcule l'exponentielle d'un nombre. Elle renvoie un double (oui, oui, elle aussi).

#### log

Cette fonction calcule le logarithme népérien d'un nombre (que l'on note aussi « ln »).

#### log10

Cette fonction calcule le logarithme base 10 d'un nombre.

L'**incrémentation** est l'opération qui consiste à ajouter 1 à une variable. On écrit variable++.

La **décrémentation** est l'opération inverse on retire 1 à une variable. On écrit donc variable--.

# Les conditions

#### Quelques symboles à connaître

| **Symbole** | **Signification** |
| --- | --- |
| == | est égal à |
| > | est supérieur à |
| < | est inférieur à |
| >= | est supérieur ou égal à |
| <= | est inférieur ou égal à |
| != | est différent de |

Cela nous donne donc à écrire :

if (/\* Votre condition \*/)

{

// Instructions à exécuter si la condition est vraie

}

Petit exemple :

if (age >= 18) // Si l'âge est supérieur ou égal à 18

{

printf ("Vous etes majeur !");

}

else // Sinon...

{

printf ("Ah c'est bete, vous etes mineur !");

}

Les mots-clés if, else if, else signifient respectivement « si », « sinon si », « sinon ». On peut écrire autant de else if que l'on désire.

#### Plusieurs conditions à la fois

| **Symbole** | **Signification** |
| --- | --- |
| && | ET |
| || | OU |
| ! | NON |

if (age > 30 || argent > 100000)

{

printf("Bienvenue chez PicsouBanque !");

}

else

{

printf("Hors de ma vue, miserable !");

}

Ce test n'est valide que si la personne a plus de 30 ans ou si elle possède plus de 100 000 euros !

if (!(age < 18))

Cela pourrait se traduire par « si la personne n'est pas mineure ». Si on avait enlevé le ! devant, cela aurait signifié l'inverse : « si la personne est mineure ».

Un if est une condition, et on ne met de point-virgule qu'à la fin d'une instruction et non d'une condition.

Un **booléen** est une variable qui peut avoir deux états : vrai (1) ou faux (0) (toute valeur différente de 0 est en fait considérée comme « vraie »). On utilise des int pour stocker des booléens car ce ne sont en fait rien d'autre que des nombres.

if (1)

{

printf("C'est vrai");

}

else

{

printf("C'est faux");

}

Résultat :

C'est vrai

Retenez bien ceci : on dit qu'une variable à laquelle on fait prendre les valeurs 0 et 1 est **un booléen**.

## Construire un switch

Les informaticiens détestent faire des choses répétitives.

Alors, pour éviter d'avoir à faire des répétitions comme ça quand on teste la valeur d'une seule et même variable, ils ont inventé une autre structure que le if... else. Cette structure particulière s'appelle switch

switch (age)

{

case 2:

printf("Salut bebe !");

break;

case 6:

printf("Salut gamin !");

break;

case 12:

printf("Salut jeune !");

break;

case 16:

printf("Salut ado !");

break;

case 18:

printf("Salut adulte !");

break;

case 68:

printf("Salut papy !");

break;

default:

printf("Je n'ai aucune phrase de prete pour ton age ");

break;

}

Voici un code qui fait exactement la même chose que le code précédent, mais écrit cette fois sous forme ternaire :

age = (majeur) ? 18 : 17;

Le point d'interrogation permet de dire « est-ce que tu es majeur ? ». Si oui, alors on met la valeur 18 dans age. Sinon (le deux-points : signifie else ici), on met la valeur 17.

# Les boucles

## La Boucle While

while (/\* Condition \*/)

{

// Instructions à répéter

}

C'est aussi simple que cela. while signifie « Tant que ». On dit donc à l'ordinateur « Tant que la condition est vraie, répète les instructions entre accolades ».

Ce code répète 10 fois l'affichage de « Salut les Zeros ! ».

1. Au départ, on a une variable compteur initialisée à 0. Elle vaut donc 0 au début du programme.
2. La boucle while ordonne la répétition TANT QUE compteur est inférieur à 10. Commecompteur vaut 0 au départ, on rentre dans la boucle.
3. On affiche la phrase « Salut les Zeros ! » via un printf.
4. On **incrémente** la valeur de la variable compteur, grâce à compteur++;. compteur valait 0, elle vaut maintenant 1.
5. On arrive à la fin de la boucle (accolade fermante) : on repart donc au début, au niveau du while. On refait le test du while : « *Est-ce que compteur est toujours inférieure à 10 ?* ». Ben oui,compteur vaut 1 ! Donc on recommence les instructions de la boucle.

Et ainsi de suite… compteur va valoir progressivement 0,   
1, 2, 3, …, 8, 9, et 10. Lorsque compteur vaut 10, la condition compteur < 10 est fausse. Comme l'instruction est fausse, on sort de la boucle.

On pourrait d'ailleurs voir que la variable compteur augmente au fur et à mesure dans la boucle, en l'affichant dans le printf :

*int* compteur = 0;

while (compteur < 10)

{

printf("La variable compteur vaut %d\n", compteur);

compteur++;

}

La variable compteur vaut 0

La variable compteur vaut 1

La variable compteur vaut 2

... etc

Pour arrêter un tel programme sous Windows, vous n'avez pas d'autre choix que de fermer la console en cliquant sur la croix en haut à droite. Sous Linux, faites Ctrl + C.

## La boucle do… while

La seule chose qui change en fait par rapport à while, c'est la position de la condition. Au lieu d'être au début de la boucle, la condition est à la fin :

*int* compteur = 0;

do

{

printf("Salut les Zeros !\n");

compteur++;

} while (compteur < 10);

C'est très simple : la boucle while pourrait très bien ne jamais être exécutée si la condition est fausse dès le départ. Par exemple, si on avait initialisé le compteur à 50, la condition aurait été fausse dès le début et on ne serait jamais rentré dans la boucle.  
Pour la boucle do… while, c'est différent : **cette boucle s'exécutera toujours au moins une fois**. En effet, le test se fait à la fin comme vous pouvez le voir. Si on initialise compteur à 50, la boucle s'exécutera une fois.

Il est donc parfois utile de faire des boucles de ce type, pour s'assurer que l'on rentre au moins une fois dans la boucle

## La boucle for.

il est dans certains cas utile d'avoir un autre système de boucle plus « condensé », plus rapide à écrire.

Comme je vous le disais, les boucles for sont juste une autre façon de faire une boucle while.  
Voici un exemple de boucle while que nous avons vu tout à l'heure :

*int* compteur = 0;

while (compteur < 10)

{

printf("Salut les Zeros !\n");

compteur++;

}

Voici maintenant l'équivalent en boucle for :

*int* compteur;

for (compteur = 0 ; compteur < 10 ; compteur++)

{

printf("Salut les Zeros !\n");

}

Quelles différences ?

* Vous noterez que l'on n'a pas initialisé la variable compteur à 0 dès sa déclaration (mais on aurait pu le faire).
* Il y a beaucoup de choses entre les parenthèses après le for

Intéressons-nous à ce qui se trouve entre les parenthèses. Il y a trois instructions condensées, chacune séparée par un point-virgule.

* La première est **l'initialisation** : cette première instruction est utilisée pour préparer notre variablecompteur. Dans notre cas, on initialise la variable à 0.
* La seconde est **la condition** : c'est la condition qui dit si la boucle doit être répétée ou non. Tant que la condition est vraie, la boucle for continue.
* Enfin, il y a **l'incrémentation**  pour mettre à jour la variable compteur, on peut aussi faire une décrémentation (variable--) ou encore n'importe quelle autre opération (variable += 2; pour avancer de 2 en 2 par exemple).

## Exercice tirer un nombre au sort

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

*int* main ( *int* argc, *char*\*\* argv )

{

*int* nombreMystere = 0, nombreEntre = 0;

const *int* MAX = 100, MIN = 1;

// Génération du nombre aléatoire

srand(time(NULL));

nombreMystere = (rand() % (MAX - MIN + 1)) + MIN;

/\* La boucle du programme. Elle se répète tant que l'utilisateur n'a pas trouvé le nombre mystère \*/

do

{

// On demande le nombre

printf("Quel est le nombre ? ");

scanf("%d", &nombreEntre);

// On compare le nombre entré avec le nombre mystère

if (nombreMystere > nombreEntre)

printf("C'est plus !\n\n");

else if (nombreMystere < nombreEntre)

printf("C'est moins !\n\n");

else

printf ("Bravo, vous avez trouve le nombre mystere !!!\n\n");

} while (nombreEntre != nombreMystere);

return 0;

}

# Les fonctions

Voyons un exemple pratique sans plus tarder : la fameuse fonction triple dont je vous parle depuis tout à l'heure. On va dire que cette fonction reçoit un nombre entier de type int et qu'elle renvoie un nombre entier aussi de type int. Cette fonction calcule le triple du nombre qu'on lui donne :

*int* triple(*int* nombre)

{

*int* resultat = 0;

resultat = 3 \* nombre; // On multiplie le nombre fourni par 3

return resultat; // On retourne la variable resultat qui vaut le triple de nombre

}

# La programmation modulaire

#### En résumé

* Un programme contient de nombreux fichiers .c. En règle générale, chaque fichier .c a un petit frère du même nom ayant l'extension .h (qui signifie **header**). Le .c contient les fonctions tandis que le .h contient les**prototypes**, c'est-à-dire la signature de ces fonctions.
* Le contenu des fichiers .h est inclus en haut des .c par un programme appelé **préprocesseur**.
* Les .c sont transformés en fichiers .o binaires par le **compilateur**.
* Les .o sont assemblés en un exécutable (.exe) par le **linker**, aussi appelé **éditeur de liens**.
* Une variable déclarée dans une fonction n'est pas accessible dans une autre fonction. On parle de **portée des variables**.

<https://openclassrooms.com/courses/apprenez-a-programmer-en-c/la-programmation-modulaire>

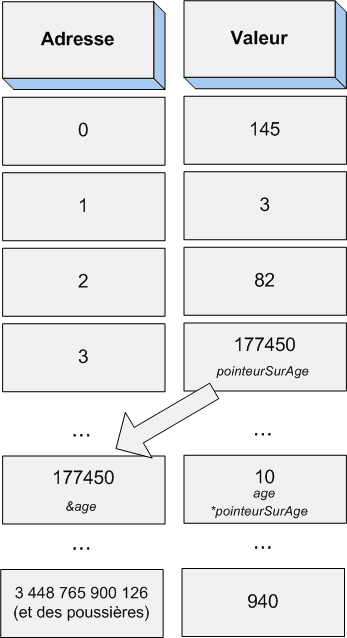
## **À L'ASSAUT DES POINTEURS**

#### À retenir absolument

Voici ce qu'il faut avoir compris et ce qu'il faut retenir pour la suite de ce chapitre :

* sur une variable, comme la variable age :
  + age signifie : « Je veux la valeur de la variable age »,
  + &age signifie : « Je veux l'adresse à laquelle se trouve la variable age » ;
* sur un pointeur, comme pointeurSurAge :
  + pointeurSurAge signifie : « Je veux la valeur de pointeurSurAge » (cette valeur étant une adresse),
  + \*pointeurSurAge signifie : « Je veux la valeur de la variable qui se trouve à l'adresse contenue dans pointeurSurAge ».

Contentez-vous de bien retenir ces quatre points. Faites des tests et vérifiez que ça marche.  
Le schéma de la fig. suivante devrait bien vous aider à situer chacun de ces éléments.



Attention à ne pas confondre les différentes significations de l'étoile ! Lorsque vous déclarez un pointeur, l'étoile sert juste à indiquer qu'on veut créer un pointeur : int \*pointeurSurAge;.  
En revanche, lorsqu'ensuite vous utilisez votre pointeur en écrivant printf("%d", \*pointeurSurAge);, cela ne signifie pas « Je veux créer un pointeur » mais : « Je veux la valeur de la variable sur laquelle pointe mon pointeurSurAge ».

<https://openclassrooms.com/courses/apprenez-a-programmer-en-c/a-l-assaut-des-pointeurs>

***Chapitre important à connaitre***

## **LES TABLEAUX**

Pour commencer, nous allons voir comment définir un tableau de 4 int :

*int* tableau[4];

Voilà, c'est tout. Il suffit donc de rajouter entre crochets le nombre de cases que vous voulez mettre dans votre tableau. Il n'y a pas de limite (à part peut-être la taille de votre mémoire, quand même).

Maintenant, comment accéder à chaque case du tableau ? C'est simple, il faut écriretableau[numeroDeLaCase].

Attention : un tableau commence à l'indice n° 0 ! Notre tableau de 4 int a donc les indices 0, 1, 2 et 3. Il n'y a pas d'indice 4 dans un tableau de 4 cases ! C'est une source d'erreurs très courantes, souvenez-vous-en.

Si je veux mettre dans mon tableau les mêmes valeurs que celles indiquées sur la fig. suivante, je devrai donc écrire :

*int* tableau[4];

tableau[0] = 10;

tableau[1] = 23;

tableau[2] = 505;

tableau[3] = 8;

## **LES CHAÎNES DE CARACTÈRES**

#### En résumé

* Un ordinateur ne sait pas manipuler du texte, il ne connaît que les nombres. Pour régler le problème, on associe à chaque lettre de l'alphabet un nombre correspondant dans une table appelée la **table ASCII**.
* Le type char est utilisé pour stocker une et une seule lettre. Il stocke en réalité un nombre mais ce nombre est automatiquement traduit par l'ordinateur à l'affichage.
* Pour créer un mot ou une phrase, on doit construire une **chaîne de caractères**. Pour cela, on utilise un **tableau de char**.
* Toute chaîne de caractère se termine par un caractère spécial appelé \0 qui signifie « fin de chaîne ».
* Il existe de nombreuses fonctions toutes prêtes de manipulation des chaînes dans la **bibliothèque string**. Il faut inclure string.h pour pouvoir les utiliser.

## **LE PRÉPROCESSEUR**

En résumé

* Le préprocesseur est un programme qui analyse votre code source et y effectue des modifications avant la compilation.
* L'instruction de préprocesseur #include insère le contenu d'un autre fichier.
* L'instruction #define définit une constante de préprocesseur. Elle permet de remplacer un mot-clé par une valeur dans le code source.
* Les macros sont des morceaux de code tout prêts définis à l'aide d'un #define. Ils peuvent accepter des paramètres.
* Il est possible d'écrire des conditions en langage préprocesseur pour choisir ce qui sera compilé. On utilise notamment les mots-clés #if, #elif et #endif.
* Pour éviter qu'un fichier .h ne soit inclus un nombre infini de fois, on le protège à l'aide d'une combinaison de constantes de préprocesseur et de conditions. Tous vos futurs fichiers .h devront être protégés de cette manière.

## **CRÉEZ VOS PROPRES TYPES DE VARIABLES**

#### https://openclassrooms.com/courses/apprenez-a-programmer-en-c/creez-vos-propres-types-de-variables

#### En résumé

* Une structure est un type de variable personnalisé que vous pouvez créer et utiliser dans vos programmes. C'est à vous de la définir, contrairement aux types de base tels que int et double que l'on retrouve dans tous les programmes.
* Une structure est composée de « sous-variables » qui sont en général des variables de type de base comme int etdouble, mais aussi des tableaux.
* On accède à un des composants de la structure en séparant le nom de la variable et la composante d'un point :joueur.prenom.
* Si on manipule un pointeur de structure et qu'on veut accéder à une des composantes, on utilise une flèche à la place du point : pointeurJoueur->prenom.
* Une énumération est un type de variable personnalisé qui peut seulement prendre une des valeurs que vous prédéfinissez : FAIBLE, MOYEN ou FORT par exemple.

## **LIRE ET ÉCRIRE DANS DES FICHIERS**

Toutes les bibliothèques que je vous ai fait utiliser jusqu'ici (stdlib.h, stdio.h, math.h, string.h…)

#### fopen : ouverture du fichier

#### fclose : fermer le fichier

FILE\* fopen(const *char*\* nomDuFichier, const *char*\* modeOuverture);

Nous allons donc créer un pointeur de FILE au début de notre fonction (par exemple la fonction main) :

*int* main(*int* argc, *char* \*argv[])

{

FILE\* fichier = NULL;

fichier = fopen("test.txt", "r+");

if (fichier != NULL)

{

// On peut lire et écrire dans le fichier

// ...

fclose(fichier); // On ferme le fichier qui a été ouvert

}

else

{

// On affiche un message d'erreur si on veut

printf("Impossible d'ouvrir le fichier test.txt");

}

return 0;

}

Voici les modes d'ouverture possibles.

* **"r" : lecture seule**. Vous pourrez lire le contenu du fichier, mais pas y écrire. *Le fichier doit avoir été créé au préalable.*
* **"w" : écriture seule**. Vous pourrez écrire dans le fichier, mais pas lire son contenu. *Si le fichier n'existe pas, il sera créé.*
* **"a" : mode d'ajout**. Vous écrirez dans le fichier, en partant de la fin du fichier. Vous ajouterez donc du texte à la fin du fichier. *Si le fichier n'existe pas, il sera créé.*
* **"r+" : lecture et écriture**. Vous pourrez lire et écrire dans le fichier. *Le fichier doit avoir été créé au préalable.*
* **"w+" : lecture et écriture, avec suppression du contenu au préalable**. Le fichier est donc d'abord vidé de son contenu, vous pouvez y écrire, et le lire ensuite. *Si le fichier n'existe pas, il sera créé.*
* **"a+" : ajout en lecture / écriture à la fin**. Vous écrivez et lisez du texte à partir de la fin du fichier. *Si le fichier n'existe pas, il sera créé.*

#### Écrire dans le fichier

* fputc : écrit un caractère dans le fichier (UN SEUL caractère à la fois) ;

*int* fputc(*int* caractere, FILE\* pointeurSurFichier);

* fputs : écrit une chaîne dans le fichier ;

*char*\* fputs(const *char*\* chaine, FILE\* pointeurSurFichier);

* fprintf : écrit une chaîne « formatée » dans le fichier, fonctionnement quasi-identique à printf.

#### Lire dans un fichier

* fgetc : lit un caractère ;

*int* fgetc(FILE\* pointeurDeFichier);

* fgets : lit une chaîne ;

*char*\* fgets(*char*\* chaine, *int* nbreDeCaracteresALire, FILE\* pointeurSurFichier);

* fscanf : lit une chaîne formatée.

Il existe trois fonctions à connaître :

* ftell : indique à quelle position vous êtes actuellement dans le fichier ;

*long* ftell(FILE\* pointeurSurFichier);

* fseek : positionne le curseur à un endroit précis ;

fseek(fichier, 2, SEEK\_SET);

* rewind : remet le curseur au début du fichier (c'est équivalent à demander à la fonction fseek de positionner le curseur au début).

La fonction fseek permet de déplacer le curseur d'un certain nombre de caractères (indiqué par deplacement) à partir de la position indiquée par origine.

* Le nombre deplacement peut être un nombre positif (pour se déplacer en avant), nul (= 0) ou négatif (pour se déplacer en arrière).
* Quant au nombre origine, vous pouvez mettre comme valeur l'une des trois constantes (généralement desdefine) listées ci-dessous :
  + SEEK\_SET : indique le début du fichier ;
  + SEEK\_CUR : indique la position actuelle du curseur ;
  + SEEK\_END : indique la fin du fichier.

#### rename : renommer un fichier

*int* rename(const *char*\* ancienNom, const *char*\* nouveauNom);

#### remove : supprimer un fichier

*int* remove(const *char*\* fichierASupprimer);

## **L'ALLOCATION DYNAMIQUE**

#### En résumé

* Une variable occupe plus ou moins d'espace en mémoire en fonction de son type.
* On peut connaître le nombre d'octets occupés par un type à l'aide de l'opérateur sizeof().
* L'allocation dynamique consiste à réserver manuellement de l'espace en mémoire pour une variable ou un tableau.
* L'allocation est effectuée avec malloc() et il ne faut surtout pas oublier de libérer la mémoire avecfree() dès qu'on n'en a plus besoin.
* L'allocation dynamique permet notamment de créer un tableau dont la taille est déterminée par une variable au moment de l'exécution.