langage C

- langage C
  - types
  - exemple programme c
  - Declaration de variables
  - Afficher le contenu d'une variable
  - Récupérez une saisie
  - Calculs
    - Les raccourcis
  - Les conditions
    - Comparaison
    - ET, OU, ...
  - expression ternaire
  - Les Boucles
  - prototype
  - structure d'un projet
    - exemple
  - La compilation
  - Portée des variables et fonctions
  - Les pointeurs
  - Les tableaux
  - Les chaînes de caractères
  - Directive de préprocesseur
    - include
    - define
    - les constantes prédefinies par le processeur
    - Les macros
    - Les conditions
    - ifdef et ifndef
  - structures et enumerations
    - Structures
    - enumeration
  - Les fichiers
    - renommer et supprimer un fichier
  - Allocation dynamique
    - sizeof
    - Allocation manuelle de la mémoire (malloc et free)
  - Mots-clés
  - Regles d'ecritures:
  - Notes importantes

types

Nom du type	Minimum	Maximum

Nom du type	Minimum	Maximum
signed char	-127	127
int	-32 767	32 767
long	-2 147 483 647	2 147 483 647
float	-1×10^{37}	1×10^{37}
double	-1×10^{37}	1×10^{37}

Pour les types entiers (signed char, int, long,...) il existe d'autres types dits "unsigned":

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

difference entre float et double:

float	précision de 6 décimales
double	précision de 15 décimales

Il existe aussi le type spéciale size_t signifiant qu'une fonction renvoie un nombre correpondant à une taille

exemple programme c

```c
/*
Ci-dessous, ce sont des directives de préprocesseur.
Ces lignes permettent d'ajouter des fichiers au projet,
fichiers que l'on appelle bibliothèques.
Grâce à ces bibliothèques, on disposera de fonctions toutes prêtes pour
afficher
par exemple un message à l'écran.
*/

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

/*
Ci-dessous, vous avez la fonction principale du programme, appelée main.
C'est par cette fonction que tous les programmes commencent.
Ici, ma fonction se contente d'afficher Bonjour à l'écran.
*/

int main()
{
    printf("Bonjour!\n"); // instruction pour afficher 'Bonjour!' à l'ecran
```

```
    return 0; // renvoie le nombre 0 puis s'arrete
}
```

Declaration de variables

```
int nb; // creer une variable nb de type int
double note; //creer une variable note de type double
```

pour creer une constante, il faut utiliser le mot "const":

```
const int nb; // creer une constante nb de type int
```

Afficher le contenu d'une variable

```
int nb, niveau;
printf("Il vous reste %d vies, niveau %d", nb, niveau);
```

Format	Type attendu
%c	char
%s	string(chaine de caractère)
%d	int
%u	unsigned int
%ld	long
%f	float
%f	double

Récupérez une saisie

```
int age = 0;
scanf("%d", &age);
```

Calculs

Opération	Signe
Addition	+

Opération	Signe
Soustraction	-
Multiplication	*
Division	/
Modulo	%

pour élever une variable a à une puissance b:

```
#include <math.h>
int a=5, b=2;
pow(a,b);
```

Les raccourcis

```
int nb;
nb++; // incrémentation
nb--; // décrémentation
nb*=2; // multiplication par 2
nb+=4; // nb = nb+4
nb-=4; // nb = nb-4
nb/=4; // nb = nb/4
nb%=4 // nb = nb%4
```

Les conditions

condition if..else:

```
if (/* condition */)
{
    // Instructions à exécuter si la condition est vraie
}
else if(/* condition */)
{
    // Instructions à exécuter si la condition est vraie
}
else{
    // Instructions à exécuter si les deux conditions précédentes ne sont
pas vraies
}
```

condition switch:

```c
int age;
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 reponse pour ton age");
  break;
}
```

Comparaison

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

ET, OU, ...

Symbole	Signification
&&	ET
||	OU
!	NON

expression ternaire

```c
int age=5, majeur;
if (age>18)
{
    majeur = 1;
}
else{
    majeur =0;
}
/* cette condition peut s'ecrire sous une forme ternaire: */
majeur=(age>18)? 1:0;
```

Les Boucles

Boucle while et do..while:

```c
while (/* Condition */)
{
    // Instructions à répéter
}
/* Ou */
do{
    // Instructions à répéter
}
while (/* Condition */);
```

Boucle for:

```c
int compteur;

for (compteur = 0 ; compteur < 10 ; compteur++)
{
    printf("%d",compteur);
}
```

Les instructions:

- Break: permet de quitter la boucle
- continue: permet de passer à l'iteration suivante de la boucle sans faire les instruction suivante de la boucle.

prototype

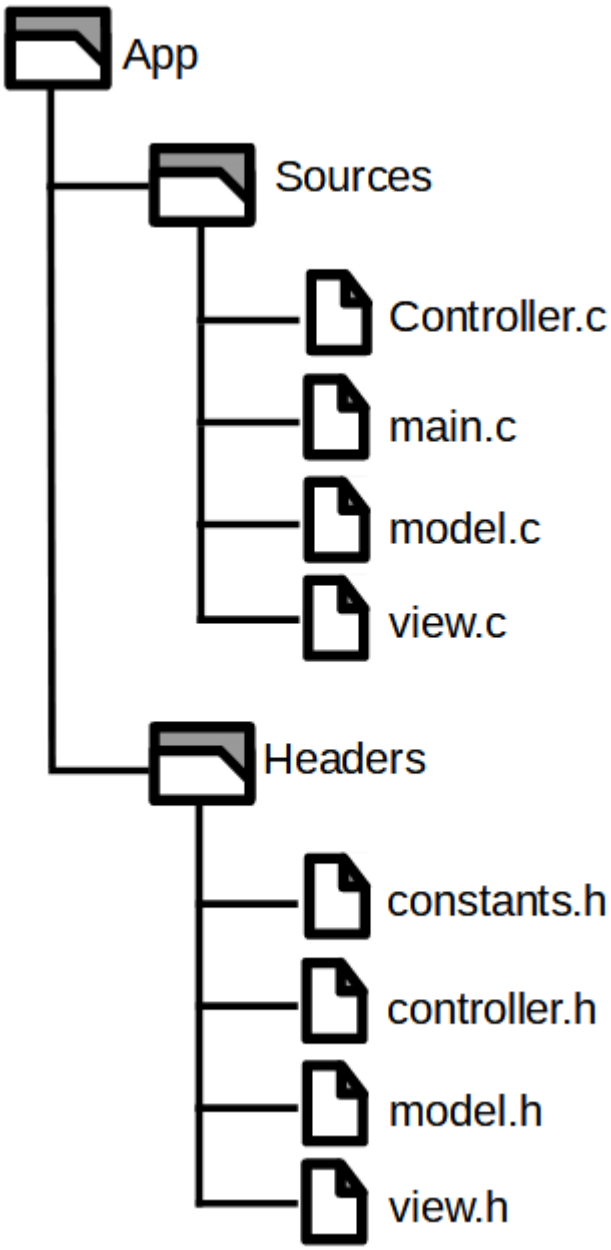
Lorsque l'on programme en C, les differentes fonction que l'on veut utiliser dans la fonction main doivent êtres déclkarées avant la fonction main. Pour pouvoir mettre les fonctions ou l'on veut, on va utiliser un prototype. Exemple:

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

double aireRectangle(double largeur, double hauteur); // On copie le nom de
la fonction avec ses paramètres en haut du programme.

int main()
{
    printf("l'aire d'un rectangle de largeur 5 et de hauteur 10 est %.1f",
aireRectangle(5,10));
    return 0;
}

double aireRectangle(double largeur, double hauteur)
{
    return largeur * hauteur;
}
```

Un prototype permet de dire à l'ordinateur qu'il existe une fonction qui prend tels paramètres en entrée et renvoie une sortie du type que l'on indique.

structure d'un projet

exemple



Dans cette exemple le fichier "view.c" contient les fonctions concernant l'interface graphique Le fichier "controller.c" contient les fonctions concernant la logique de l'application.

2 types de fichiers:

- .h: fichiers header, contiennent les prototypes des fonctions
- .c: fichiers source, contiennent les fonctions

Dans le fichier "view.c":

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

On retrouve les bibliothèques "stdio.h" et "stdlib.h" entre <>. Ce sont des bibliothèques déja installés. On retrouve ensuite le fichier header "view.h" entre "". Il est placé dans le repertoire du projet. On retrouve dans

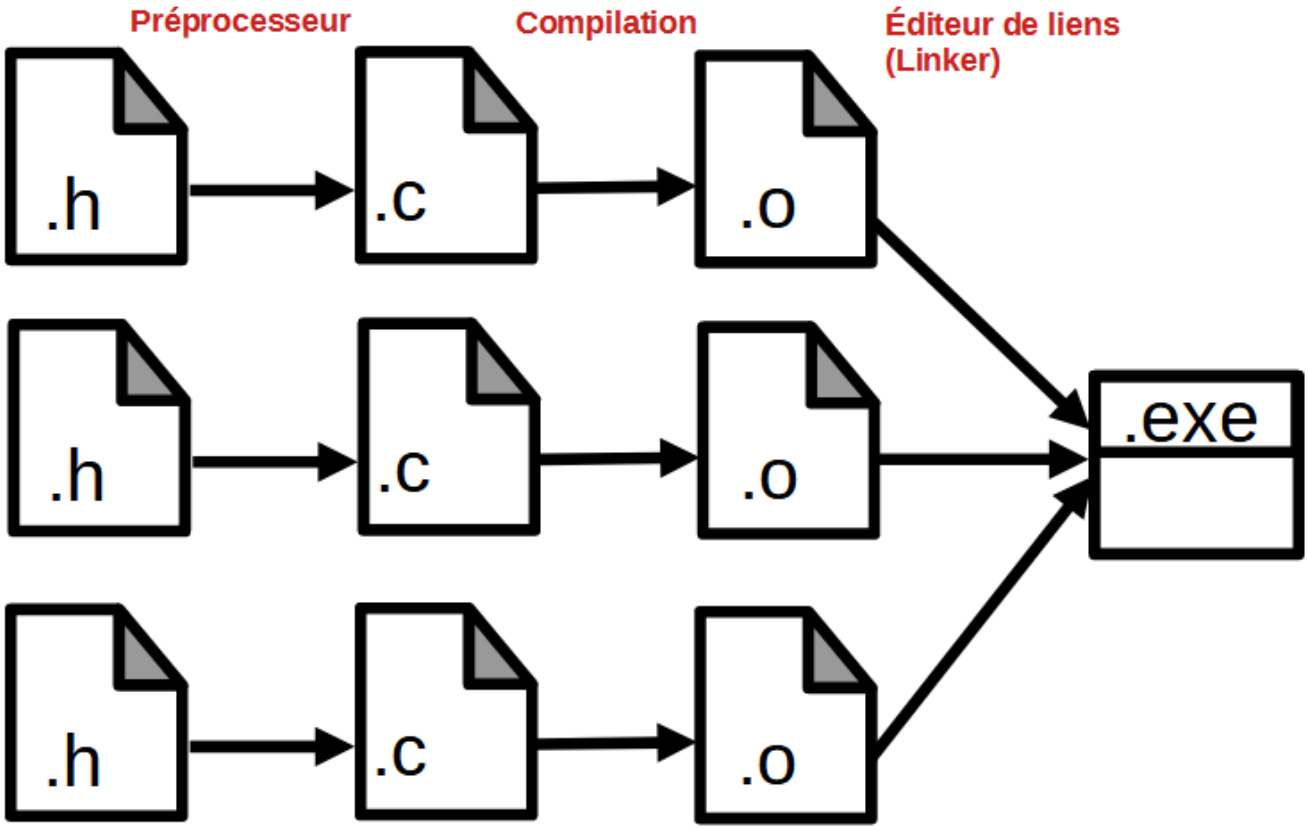
ce fichier les prototypes des fonctions du fichier "view.c".
ATTENTION: lors de la compilation, il faut compiler tous les fichiers .c. Par exemple:

```
gcc *.c -o start.exe
```

La compilation

La compilation d'un programme se fait en plusieurs étapes:

- **préprocesseur**: programme démarrant avant la compilation. Il execute les instructions spéciales données par les directives commençant par un # (#include,...). Il remplace par exemple les directives "include" par les contenus des fichiers .h.
- **compilation**: génere des fichiers en code binaire compréhensible par l'ordinateur à partir des fichiers source (.c). Le fichier résultant est un fichier .o ou .obj en fonction du compilateur.
- **éditeur de liens (le linker)**: il assemble tous les fichiers .o dans un seul fichier executable (.exe sous windows).



Portée des variables et fonctions

Variable prope à une fonction:

```
int fonction(){
  int nb; //la variable ne peut être utilisé que dans cette fonction
}
```

Variable global accessible dans tout le projet:

```
int resultat; //Déclaration de variable globale
void fonction(){
  int nb=5;
  resultat = nb
}
```

Variable global accessible seulement dans le fichier:

```
static int resultat; //Déclaration de variable globale
```

Si l'on met le mot clé **"static"** à l'interieur d'une fonction, la variable n'est plus supprimée à la fin de la fonction. La prochaine fois que l'on appellera la fonction, la variable aura conservé sa valeur.

```
int incremente(){
 static int nb=0;
 return nb++;
}
```

Si il est mis avant une fonction, cette fonction ne sera accessible seulement dans le fichier.

```
static int foo(){
  //instructions
}
/* Ou on peut mettre à jour le prototype */
static int foo();
```

Les pointeurs

pour voir l'adresse d'une variable:

```
int nb=10;
printf("l'adresse e la variable nb est %p", &nb); // %p retourne une valeur
hexadecimale de l'adresse de la variable. Pour avoir en décimal, on remet
%d.
```

dans ce cas:

- "nb" désigne la valeur de la variable

- "&nb" désigne l'adresse de la variable Un pointeur contient une adresse. Pour creer un pointeur, on rajoute le symbole * devant le nom de la variable:

```
int *pointeur=NULL; // Ne pas oublier d'initialiser un pointeur avec le mot clé NULL
```

Pour faire un pointeur vers une autre variable:

```
int nb=10;
int *pointeurNb = &nb;
printf("%d", pointeurNb); // retourne la valeur du pointeur (en déciaml) qui correspond à l'adresse de la variable vers laquel le pointeur pointe.
printf("%d", &pointeurNb); // retourne l'adresse du pointeur (en décimal)
printf("%d", *pointeurNb); // retourne la valeur de la variable vers laquel le pointeur pointe (dans ce cas c'est 10)
```

ce qu'il se passe dans la mémoire:

variable	Adresse	Valeur
nb	1602223372	10
pointeurNb	1491322395	1602223372

valeur pointeurNb = adresse nb. L'adresse du pointeur est une adresse allouée par la machine comme pour toutes autres variables normales. Exemple de programme avec des pointeurs:

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

void decoupeMinutes(int *heures, int *minutes);
int main(){
    int heures, minutes;
    printf("entrez le nombre d'heures:");
    scanf("%d",&heures);
    printf("entrez le nombre de minutes:");
    scanf("%d",&minutes);
    decoupeMinutes(&heures, &minutes); // On passe en argument les adresses des variables heures et minutes
    printf("%d heures %d minutes: ", heures, minutes);
    return 0;
}

void decoupeMinutes(int *heures, int *minutes){ // Les deux arguments de la fonctions sont des pointeurs
    *heures += *minutes/60; // On modifie directement la valeur de la variable vers lequel le pointeur pointe
```

```
    *minutes = *minutes%60;
}
```

Les tableaux

Les tableaux prennent un espace contigu en mémoire:les cases sont les unes à la suite des autres. Toutes les cases su tableau sont du même type. Pour creer un tableau:

```
int tableau[10]; // créer un tableau de 10 entiers
```

Pour récurperer les valeurs d'un tableau:

```
int tableau[3]; // créer un tableau de 3 entiers
tableau[0]=12; // Affecte la valeur 12 dans la 1er case du tableau
tableau[1]=34; // Affecte la valeur 34 dans la 2eme case du tableau
tableau[2]=42; // Affecte la valeur 42 dans la 3eme case du tableau
printf("%d", tableau); // Affiche l'adresse de la première case du tableau
printf("%d", tableau[0]); // Affiche la valeur de la premiere case du
tableau
/* OU */
printf("%d", *tableau); // pareil que tableau[0]
printf("%d", *(tableau+1)); // pareil que tableau[1]
```

ATTENTION, selon le compilateur, on ne peut pas faire de tableau en mettant entre crochet une variable. Il faut ecrire directement le nombre de case du tableau
Pour initialiser un tableau:

```
int tableau[4];
for (int i=0;i<4;i++){
  tableau[i]=0 // initialise le tableau à 0
}
/* OU */
int tableau[4]={11,2,4,0};
/* OU */
int tableau[4]={13}; //initialise la première case à 13 et toutes les
autres cases seront mises à zero
/* OU */
int tableau[]={13,16,12}; //initialise un tableau avec les cases mises
entre crochet
```

Les chaînes de caractères

On utilise le type char pour stocker une lettre:

```
char lettre ='a';
```

Pour afficher le caractères avec printf:

```
char lettre = 'A';
printf("%c\n", lettre);
```

Une chaine de caractère n'est en fait qu'un tableau de char:

```
char mot[]={'f','o','o','\0'};
```

IMPORTANT: Les chaines de caractères doivent toujours se finir par le caractère "\0". Il indique la fin de la chaîne de caractères (on peut donc passer la chaîne de caractères à une fonction sans avoir a indiquer la taille de la chaîne).
Pour afficher une chaîne de caractères:

```
char mot[]="foo"; // le caractère \0 est automatiquement rajouté. Ne marche
que pour l'initialisation
printf("%s\n", mot);
```

Pour récuperer une chaîne via scanf:

```
char mot[100];
scanf("%s", &mot);
```

Pour la manipulation de chaînes, la librairie string.h est disponible:

- strlen pour calculer la longueur d'une chaîne.
- strcpy pour copier une chaîne dans une autre.
- strcat pour concaténer 2 chaînes.
- strcmp pour comparer 2 chaînes.
- strchr pour rechercher un caractère.
- strpbrk pour rechercher le premier caractère de la liste.
- strstr pour rechercher une chaîne dans une autre.
- sprintf pour écrire dans une chaîne.

Directive de préprocesseur

include

La directive **include** permet de rajouter un fichier dans un autre.

define

la directive define permet de definir une constante de préprocesseur (associer une valeur à un mot):

```
#define A 10
#define B 10
#define TAILLE_TAB A*B // la constante de préprocesseur TAILLE_TAB = A*B
donc 100
int main(){
  int tab[TAILLE_TAB]; // On peut utiliser la constante de préprocesseur
pour définir la taille d'un tableau, car ce n'est pas une variable.
}
```

les constantes prédefinies par le processeur

- __LINE__ : numero de la ligne actuelle
- __FILE__ : nom du fichier actuel
- __DATE__ : date de compilation
- __TIME__ : heure de compilation

Les macros

le mot clé define permet aussi de demander au préprocesseur de remplacer un mot par un code source tout entier.

```
#define FOO() printf("foo");
int main(){
  FOO() // remplace FOO() par la macro definie dans les directives de
préprocesseur
  return 0;
}

/* OU */

#define FOO(nb, str) printf("foo %d", nb); \
              printf("foo1 %s", str); \
              printf("foo2"); // macro avec plusieurs lignes et des
paramètres
int main(){
  FOO(10, "foobar")
  return 0;
}
```

Les conditions

```
#if condition
    /* Code source à compiler si la condition est vraie */
```

```
#elif condition2
    /* Sinon si la condition 2 est vraie compiler ce code source */
#endif
```

ifdef et ifndef

- ifdef: si la constante est définie
- ifndef: si la constante n'est pas definie Utile pour eviter les problèmes d'inclusion infinie:

structures et enumerations

Structures

Une structure est un assemblage de variables de differnts types. Ils sont généralement définis dans les fichiers .h. Pour definir une structure:

```
struct Coor{
  int x;
  int y;
}; // Ne pas oublier le point virgule aprés l'accolade fermante
```

Pour utiliser la structure:

```
int main(){
  struct Coor point; //creer une variable de type Coor (la structure
définit précedemment). On doit mettre le mot clé struct pour préciser à
l'ordinateur que c'est un type personnalisé
  return 0;
}
```

Pour ne pas avoir à utiliser le mot "struct" à chaque fois que l'on veut utiliser notre type, on peut utiliser la directive typedef permettant de preciser à l'ordinateur le nom su type personnalisé:

```
/* main.h */
typedef Coordonnees Coor; //définition du type Coor correspondant au struct
Coordonnees
struct Coordonees{
  int x;
  int y;
};

/* main.c */
#include "main.h"
int main(){
  Coor point; // On peut maintenant utiliser directement Coor sans mettre
struct avant
```

```
  return 0;
}
```

Pour utiliser un pointeur sur une fonction:

```
void int_coor(Coor *point){
  (*point).x=0; // il ne faut pas oublier les parenthèses
  (*point).y=0;
}
```

enumeration

Une enumeration ne contient pas de sous variable, mais seulement des valeurs possibles pour une variable

```
enum Couleur{
    BLEU=10,
    BLANC=20,
    ROUGE=30,
};

int main(){
    enum Couleur couleur=ROUGE;
    printf("%d", couleur); // Affiche 30
    return 0;
}
```

Les fichiers

Pour manipuler un fichier, il faut d'abord inclure la stdio.h contenant des fonctions de base permettant d'ecrire et de lire un fichier.
La structure FILE permet de definir un fichier.

On utilise differentes fonctions ensuite differentes fonctions pour lire ou écrire un fichier:

- ecriture fichier:
  - **fputc**: ecrit un caractère dans un fichier.
  - **fputs**: écrit une chaîne dans un cichier.
  - **fprintf**: écrit une chaîne formatée dans un fichier

```
/* Prototypes */
int fputc(int caractere, FILE* pointeurSurFichier);
char* fputs(const char* chaine, FILE* pointeurSurFichier);
fprintf(fichier, "Le Monsieur qui utilise le programme, il a %d ans", age);
```

- lecture fichier:

    - **fgetc**: lit un caractère dans un fichier
    - **fgets**: lit une chaîne dans un fichier
    - **fscanf**: lit une chaîne formatée

```
/* Prototypes */
int fgetc(FILE* pointeurDeFichier);
char* fgets(char* chaine, int nbreDeCaracteresALire, FILE*
pointeurSurFichier);
```

- se deplacer dans un fichier:
    - **ftell**: indique la position du curseur dans le fichier
    - **fseek**: positionne le curseur à un endroit précis
    - **rewind**: remet le curseur au début du fichier

```
/* Prototypes */
long ftell(FILE* pointeurSurFichier);
int fseek(FILE* pointeurSurFichier, long deplacement, int origine);
void rewind(FILE* pointeurSurFichier);
```

exemple d'utilisations:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_BUFFER 10

void write_file(const char *nomFichier, const char *content){
    FILE *fichier =NULL;
    fichier=fopen(nomFichier, "w+");
    if(fichier != NULL){
        fputs(content ,fichier);
        printf("%s\n"), content;
        if (fclose(fichier)==EOF){
            printf("erreur lors de la fermeture fichier test.txt\n");
        }
    }
    else{
        printf("echec ouverture fichier %s\n", nomFichier);
    }
}

void read_file(char *text, const char *nomFichier){
    FILE *fichier=NULL;
    int i=0;
    fichier=fopen(nomFichier, "r+");
    if(fichier != NULL){
        for(i; i<MAX_BUFFER;i++)
        {
```

```
            text[i]= getc(fichier);
            if (text[i]==EOF){
                 text[i]=0;
                 break;
             }
        }
        if (fclose(fichier)==EOF){
            printf("erreur lors de la fermeture fichier test.txt\n");
        }
    }
    else{
        printf("echec ouverture fichier %s\n", nomFichier);
    }
}

int main(int argc, char *argv[])
{
    char text[MAX_BUFFER]={0};
    read_file(text, "test.txt");
    write_file("test1.txt","Bienvenue en Russie");
    printf("contenu fichier: %s\n", text);
    return 0;
}
```

les modes d'ouverture:

r	lecture seule. Le fichier doit avoir été créé au préalable.
w	ecriture seule, si le fichier n'existe pas, il est créé.
a	mode ajout
a+	ajout en lecture /ecriture à la fin. ecrit et lit du texte à partir de la fin du fichier
w+	lecture et ecriture. Suppression du contenu au préalable
r+	lecture et ecriture. Le fichier doit avoir été créé au préalable.

renommer et supprimer un fichier

- rename:

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

- remove:

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

Allocation dynamique

sizeof

La fonction sizeof permet de determiner la taille en memoire d'une variable

```
int main(int argc, char *argv[]){
    printf("%d\n",sizeof(char)); // retourne 1. Une variable du type char
occupe 1 octet en memoire
}
```

type	taille (en Octets)
char	1
int	4
long	8
float	4
double	8

Allocation manuelle de la mémoire (malloc et free)

La bibliothèque stdlib.h contient 2 fonctions permettant d'allouer manuellement de la memoire:

- **malloc**("Memory Allocation"): demande à l'os la permission d'utiliser de la mémoire

```
void* malloc(size_t nombreOctetsNecessaires);
```

- **free**: indique au système que l'on n'a plus besoin de la mémoire que l'on a demandée.

```
void free(void* pointeur);
```

Pour allouer manuellement de la mémoire, 3 étape:

1. Appeler malloc pour demander de la mémoire
2. Vérifier la valeur retournée par malloc pour savoir si le système a bien réussi à allouer la mémoire
3. Liberer la mémoire avec free. **ATENTION**, Si on ne fait pas cette étape, on risque des fuites de mémoire.

Mots-clés

Mot clés	Déscription
break	quitte une boucle

Mot clés	Déscription
continue	saute l'iteration actuelle de la boucle et passe à l'itération suivante
static	3 cas possibles: • en dehors des fonctions: creer une variable globale accessible seulement dans le fichier • avant la declaration d'une fonction: permet de rendre la fonction accessible seulement dans le fichier • dans une fonction: la variable déclarée en static dans une fonction retiendra sa dernière valeur lorsque la fonction sera rappelée

Regles d'ecritures:

- nom de variable: ecrire les noms de variables en minuscule. Si c'est une variable avec plusieurs mots, ecrire en camelCase (en mettant la première lettre de la variable en minuscule)
- nom de struct: ecrire en CamelCase (en mettant la première lettre en majuscule)

Notes importantes

- Il y a une difference entre ' et ". on utilise " pour un string et ' pour un caractère

```
char a='f'
/* != */
char *a="foobar"
```