

Entrées/sorties

Affichage écran

Chaine de caractères: stockage, bonnes pratiques

Ecriture/Lecture texte

Ecriture/Lecture fichiers (ASCII)

Entrees/sorties

Affichage/lecture écran/fichier

Ecran:

Information utilisateur

Debug d'un programme

Communication avec l'utilisateur

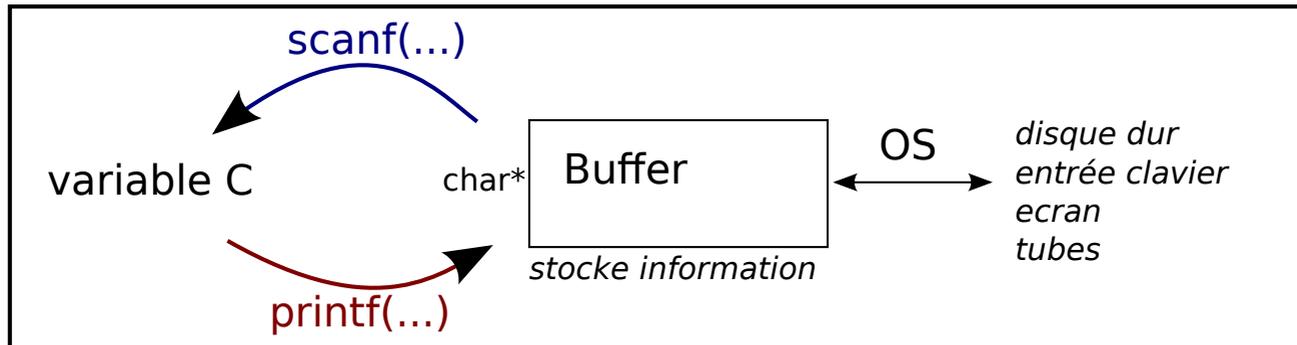
Fichier:

Suivi execution (fichier log)

Sauvegarde information (disque=mémoire non volatile)

Communication entre programme (disque=mémoire partagée)

En C: écriture fichier // écriture écran



Entrees/sorties

Rappels caractère *char*

Un caractère (char) = nombre entre 0-256

1 octet
= 1 emplacement mémoire
= 2 caractères hexa

ex.

```
char c='M';
```

4D

```
int value=(int)c;
```

↖ 77

Dec	Hx	Oct	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOF (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	X	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	:	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Caractère: entre ' '

Entrées/sorties

Affichage écran

→ **Chaine de caractères: stockage, bonnes pratiques**

Ecriture/Lecture texte

Ecriture/Lecture fichiers (ASCII)

Entrees/sorties

Rappels chaine de caractère

*const char** , *char []*

Chaine de caractère = suite de caractères

char nom[10]; → réserve 10 emplacements mémoire

nom[0]='m';

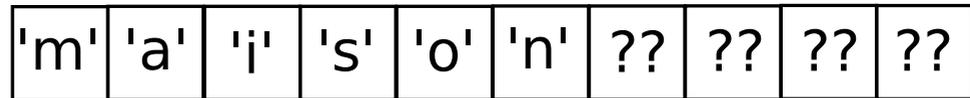
nom[1]='a';

nom[2]='i';

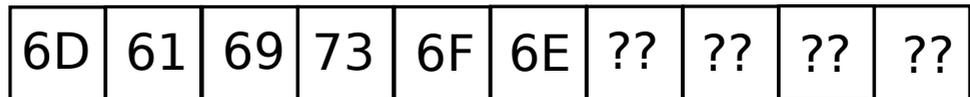
nom[3]='s';

nom[4]='o';

nom[5]='n';



↓ en mémoire

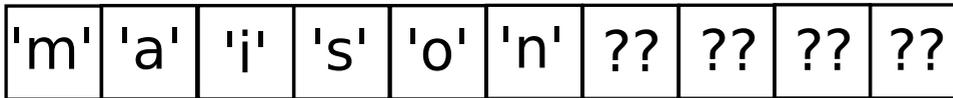


Entrees/sorties

Rappels chaine de caractère

`const char*` , `char []`

Chaine de caractère = suite de caractères



↓ en mémoire



`nom` correspond à la première case de la chaine

`nom` = pointeur sur la 1ere case

```
#include <stdio.h>

int main()
{
    char nom[10];
    nom[0]='m';
    nom[1]='a';
    nom[2]='i';
    nom[3]='s';
    nom[4]='o';
    nom[5]='n';

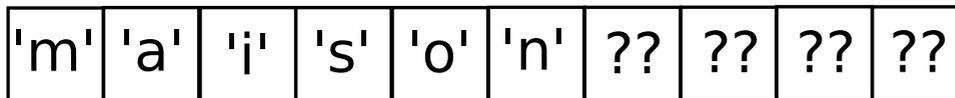
    printf("%s\n",nom);
    return 0;
}
```

Entrees/sorties

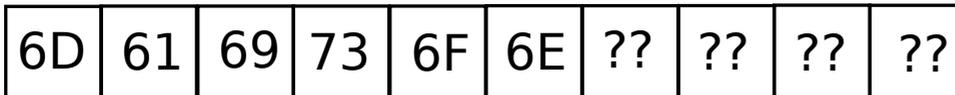
Rappels chaine de caractère

*const char** , *char []*

Chaine de caractère = suite de caractères



↓
en mémoire



Comment savoir où s'arrêter ?

```
#include <stdio.h>

int main()
{
    char nom[10];
    nom[0]='m';
    nom[1]='a';
    nom[2]='i';
    nom[3]='s';
    nom[4]='o';
    nom[5]='n';

    printf("%s\n",nom);
    return 0;
}
```

Entrees/sorties

Rappels chaine de caractere

`const char*` , `char []`

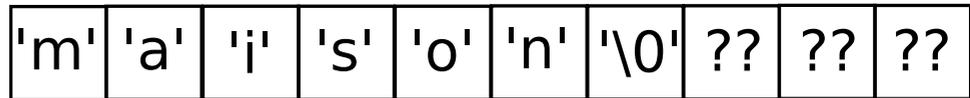


Fin de chaine = principe de **sentinelle de fin**

```
#include <stdio.h>

int main()
{
    char nom[10];
    nom[0]='m';
    nom[1]='a';
    nom[2]='i';
    nom[3]='s';
    nom[4]='o';
    nom[5]='n';
    nom[6]='\0';

    printf("%s\n", nom);
    return 0;
}
```



↓ en memoire



Affiche **maison**

Comportement deterministe

Entrees/sorties

Rappels chaine de caractere

`const char* , char []`

Fin de chaine = `'\0'`

Toujours



Toujours

=> Toujours prévoir une case pour placer `'\0'`

Toujours

Fonction pour trouver la taille d'un mot:

```
int calcul_taille(char mot[])
{
    char *pointeur_debut=mot;
    char *pointeur_fin=pointeur_debut;

    while(*pointeur_fin!='\0')
        ++pointeur_fin;

    return (int)(pointeur_fin-pointeur_debut);
}
```

Entrees/sorties

Rappels chaine de caractere

*const char** , *char []*

Fonction pour trouver la taille d'un mot:

```
int calcul_taille(char mot[])
{
    char *pointeur_debut=mot;
    char *pointeur_fin=pointeur_debut;

    while(*pointeur_fin!='\0')
        ++pointeur_fin;

    return (int)(pointeur_fin-pointeur_debut);
}
```

Attention!! Si '\0' est oublié !!!

=> Comportement indetermine X

Entrees/sorties

Rappels chaine de caractère

`const char*` , `char []`

Les fonctions de manipulation de chaine: `#include <string.h>`

`str<...>`

`strcpy`
`strcmp`
`strcat`
...

jusqu'à '\0'

`strncpy`
`strncmp`
`strncat`
...

jusqu'à '\0'
maximum `n` caractères

→ copie
→ compare
→ concatenation

Bonnes pratiques: Chaîne de caractères

Définir les chaînes "inline" en tant que `char[]` `char[]="ma_chaine"`

Définir les pointeurs (`char*`) en tant que constantes
`const char*`

Attention:

```
char mot[]="abricot";  
mot[2]='l';
```

OK



```
char *mot="abricot";  
mot[2]='l';
```

KO



pointeur vers
une chaîne constante

Bonnes pratiques: Chaîne de caractères

Toujours utiliser les fonctions à tailles limitées:

`strn<...>`

~~`strcpy(mot_1,mot_2);
strcmp(mot_1,mot_2);
strcat(mot_1,mot_2);`~~

`strncpy(mot_1,mot_2,n_max);
strncmp(mot_1,mot_2,n_max);
strncat(mot_1,mot_2,n_max);`

+ sécurisé

+ debug plus aisé

Note:

Le C n'est pas le langage approprié pour le traitement complexe de chaîne de caractères

Bonnes pratiques: Chaîne de caractères

Ne pas utiliser d'opérateurs sur des chaîne de caractères !!!



```
#include <stdio.h> X

int main()
{
    char mot_1[]="abricot";
    char *mot_2="peche";

    mot_2=mot_1;
    printf ("%s\n", mot_2);

    char mot_3[]="abricot";
    if(mot_1==mot_3)
        printf("meme mot");

    char* mot_4="hello ";
    char* mot_5="world";

    mot_4 += *mot_5;

    return 0;
}
```

Attention!!!

gcc -Wall -Wextra

ne donne aucun Warning !

Pourtant ce programme ne fait
probablement pas ce que vous souhaitez!

Bonnes pratiques: Chaîne de caractères

Ne pas utiliser d'opérateurs sur des chaîne de caractères !!!

```
#include <stdio.h> X

int main()
{
    char mot_1[]="abricot";
    char *mot_2="peche";

    mot_2=mot_1;
    printf ("%s\n",mot_2);

    char mot_3[]="abricot";
    if (mot_1==mot_3)
        printf("meme mot");

    char* mot_4="hello ";
    char* mot_5="world";

    mot_4 += *mot_5;

    return 0;
}
```

OK

mauvaise habitude

affectation de pointeurs !!
ce n'est pas une copie de chaîne !

mauvaise habitude

test d'égalité d'adresse de pointeur!
ne compare pas la chaîne (ici test = faux)

mauvaise habitude

mauvaise habitude

ajoute (int)(mot_5[0])=119 à l'adresse de mot_4
Ne réalise absolument la concaténation d'une chaîne!!

Bonnes pratiques: Chaîne de caractères



Ne pas utiliser d'opérateurs sur des chaînes de caractères !!!

```
#include <stdio.h>

int main()
{
    char mot_1[]="abricot";
    char *mot_2="peche";

    mot_2=mot_1;
    printf("%s\n", mot_2);

    char mot_3[]="abricot";
    if(mot_1==mot_3)
        printf("meme mot");

    char* mot_4="hello ";
    char* mot_5="world";

    mot_4 += *mot_5;

    return 0;
}
```

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

#define N 10

int main()
{
    char mot_1[N]="abricot";
    char mot_2[N]="peche";

    strncpy(mot_1, mot_2, N);

    char mot_3[N]="abricot";
    if(strncmp(mot_1, mot_3, N)==0)
        printf("meme mot");

    char mot_4[N]="hello";
    char mot_5[N]=" world";
    strncat(mot_4, mot_5, N);

    return 0;
}
```

OK

Entrées/sorties

Affichage écran

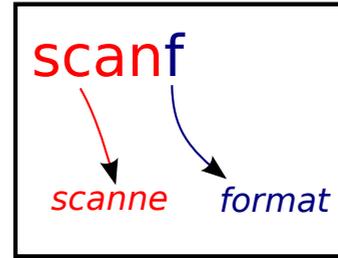
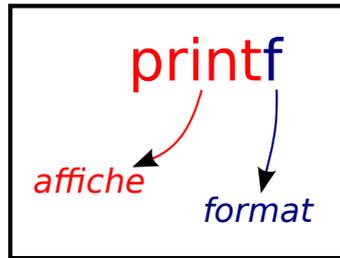
Chaine de caractères: stockage, bonnes pratiques

→ **Ecriture/Lecture texte**

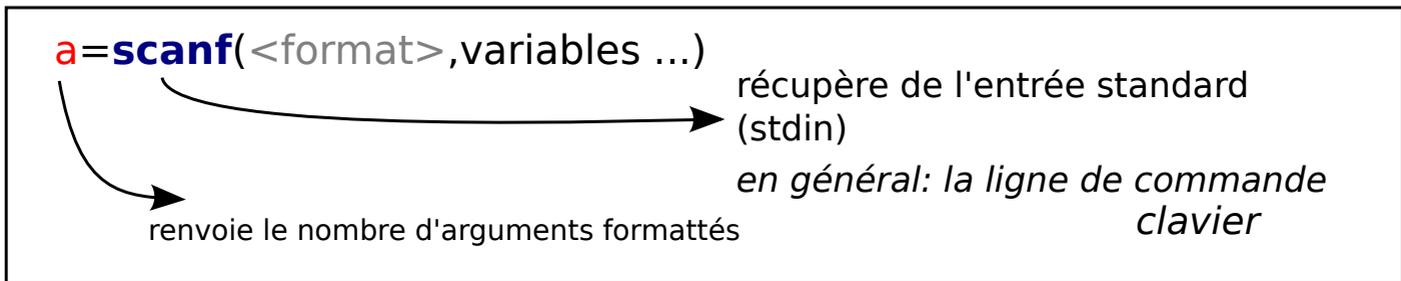
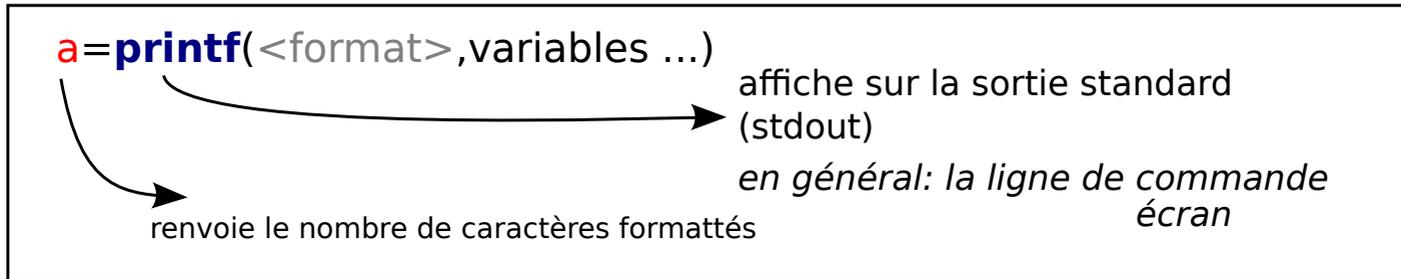
Ecriture/Lecture fichiers (ASCII)

Printf / Scanf

Fonctions de conversions



Principe:



Printf / Scanf

ex

```
int a=12;
printf("%d",a); //affiche: 12

int b=0xA AFF4E3D;
printf("%x",b); //affiche aaff4e3d

char c='e';
printf("%c",c); //affiche: e

char mot[]="bonjour a tous";
printf("%s",mot); //affiche: bonjour a tous

float x=1.25;
printf("%f",x); //affiche 1.250000
printf("%.3f",x); //affiche 1.250
```

Printf / Scanf

ex

printf (et scanf) accepte un nombre d'argument variable
=> Fonction variadiques

```
printf("\n");  
printf("Je suis %s a %c%c%c \n", "etudiant", 'C', 'P', 'E');  
printf("j'ai %d ans\n", 15);  
  
printf("1.3+1.7=%1.2f\n", 1.3+1.7);
```

```
int a=printf("\n");  
int b=printf("Je suis %s\n", "heureux");  
int c=printf("%d-%d=%d\n", 5, 7, 5-7);  
int d=printf("%d+4=%d\n", 3);  
  
printf("%d %d %d %d\n", a, b, c, d);
```

```
Je suis heureux  
5-7=-2  
3+4=1522178016  
1 16 7 15
```

oublie argument
=> comportement indéterminé
(=> Warnings!)

Printf / Scanf

ex

```
int main()
{
    int a=0;
    scanf ("%d", &a);

    float x=0.0;
    scanf ("%f", &x);

    char c='a';
    scanf (" %c", &c);

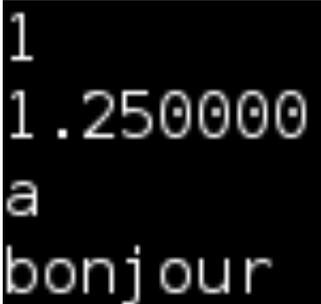
    char buffer[50];
    scanf ("%50s", buffer);

    printf ("%d\n%f\n%c\n%s\n", a, x, c, buffer);

    return 0;
}
```

```
$ ./mon_executable
> 1
> 1.25      entrées
> a         utilisateur
> bonjour
```

affiche:

A screenshot of a terminal window showing the output of the program. The output consists of four lines: '1', '1.250000', 'a', and 'bonjour'. The text is white on a black background.

```
1
1.250000
a
bonjour
```

Printf / Scanf

ex

```
int main()
{
    int a=0;
    scanf ("%d", &a);

    float x=0.0;
    scanf ("%f", &x);

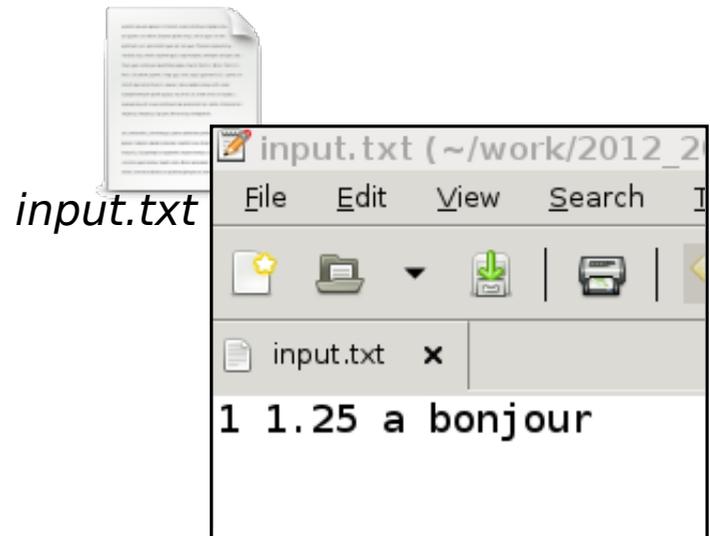
    char c='a';
    scanf (" %c", &c);

    char buffer[50];
    scanf ("%50s", buffer);

    printf ("%d\n%f\n%c\n%s\n", a, x, c, buffer);

    return 0;
}
```

Astuce:
entrée utilisateur:



```
$ ./mon_executable < input.txt
```

```
1
1.250000
a
bonjour
```

l'entrée standard devient le fichier!
=> scanf vient "lire" dans le fichier

Printf / Scanf

ex

```
int main()
{
    int x=0,y=0,z=0;
    scanf("vecteur: %d %d %d\n", &x, &y, &z);

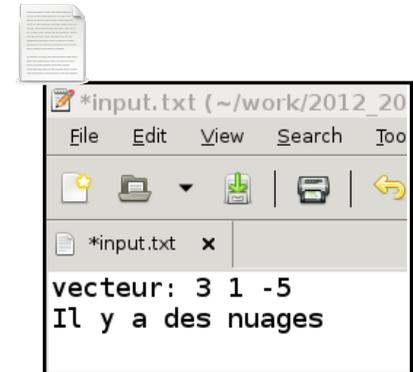
    char buffer[256];
    scanf("Il y a des %256s\n",buffer);

    printf("(%d,%d,%d) , %s\n",x,y,z,buffer);

    return 0;
}
```

```
$ ./mon_executable < input.txt
> (3,1,-5) , nuages
```

input.txt



Printf / Scanf

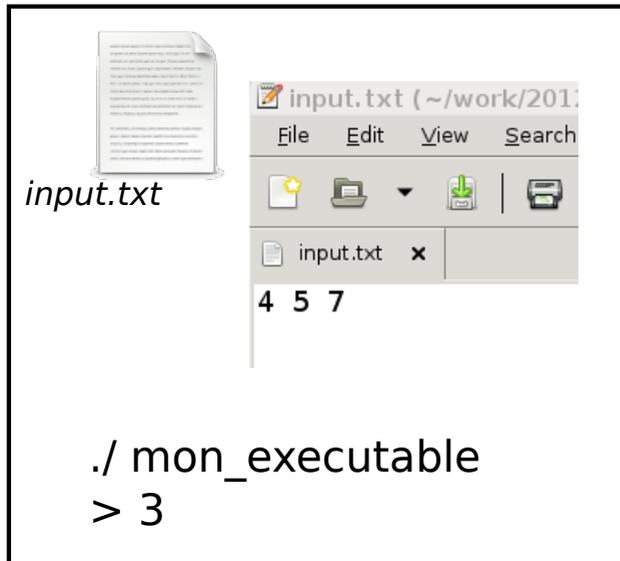
ex

```
int main()
{
    int x=0,y=0,z=0;

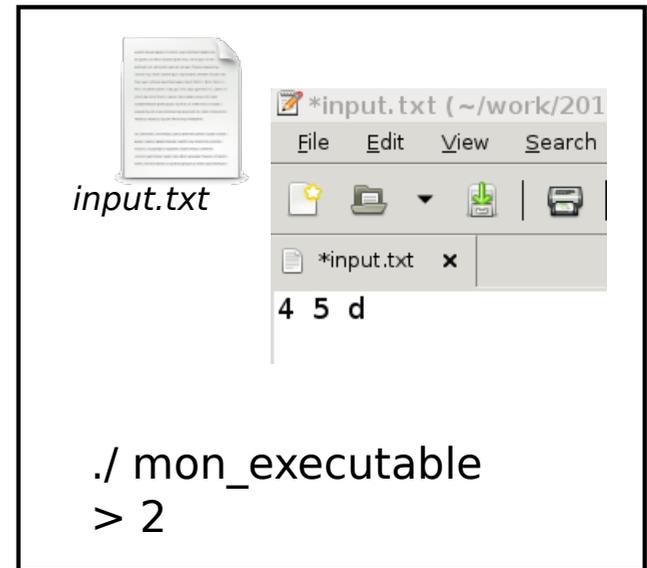
    int valeur=scanf ("%d %d %d",&x,&y,&z);

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

    return 0;
}
```



The screenshot shows a terminal window with a menu bar (File, Edit, View, Search) and a toolbar with icons for file operations. The terminal title is "input.txt (~/work/201...". The terminal content shows the command `./ mon_executable` followed by the input `> 3`. The output of the program is `4 5 7`.



The screenshot shows a terminal window with a menu bar (File, Edit, View, Search) and a toolbar with icons for file operations. The terminal title is "*input.txt (~/work/201...". The terminal content shows the command `./ mon_executable` followed by the input `> 2`. The output of the program is `4 5 d`.

Printf / Scanf

Formatage sur d'autres entrées/sorties

sprintf / **sscanf**

string: chaîne de caractères

Principe:

`a=sprintf(buffer,<format>,variables ...)`

affiche dans le buffer

renvoie le nombre de caractères formatés

`a=sscanf(buffer,<format>,variables ...)`

récupère du buffer

renvoie le nombre d'arguments formatés

Printf / Scanf

```
int main()
{
    char buffer[512];
    float x=cos(0.5);
    sprintf(buffer, "%s, (%d,%2.3f)", "bonjour", 1+4, x);

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

    return 0;
}
```

```
$ ./mon_executable
> bonjour, (5,0.878)
```

Printf / Scanf

Analyse d'une chaine de caractères

```
int main()
{
    char buffer[512]="Il fait beau ce matin. Il est 13 heures et fait 25.1 degres \n";

    char temps[25];
    int heure=0;
    float temperature=0.0;
    sscanf(buffer,"Il fait %s ce matin. Il est %d heures et fait %f degres \n",&temps,&heure,&temperature);

    printf("%s %d %1.2f\n", temps,heure,temperature);

    return 0;
}
```

```
$ ./mon_executable
> beau 13 25.10
```

Entrées/sorties

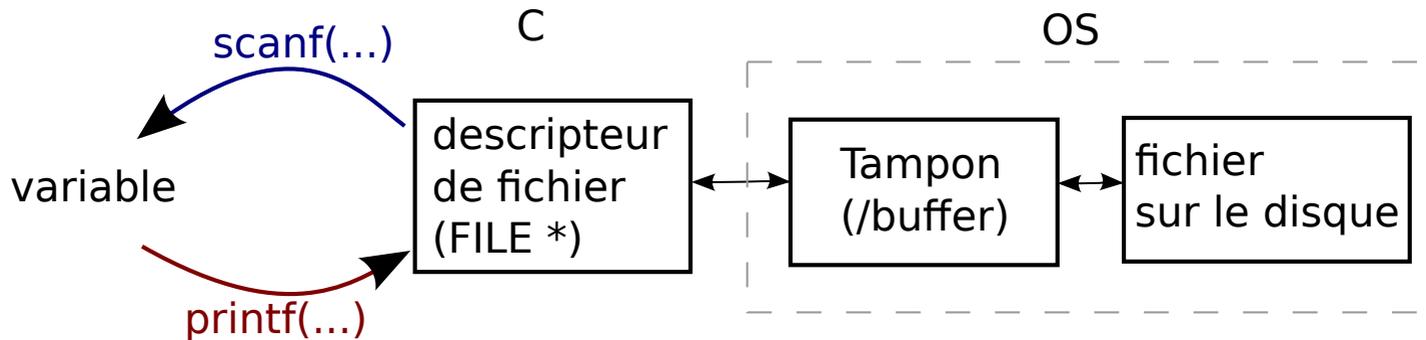
Affichage écran

Chaine de caractères: stockage, bonnes pratiques

Ecriture/Lecture texte

→ **Ecriture/Lecture fichiers (ASCII)**

Ecriture/lecture fichier



Ouvrir un fichier (pour l'écriture)

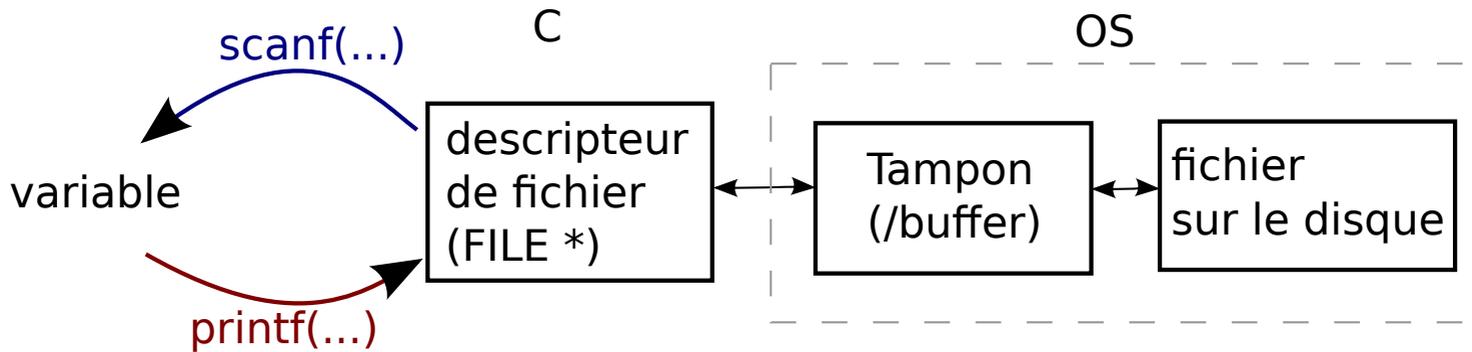
```
FILE* mon_descripteur=NULL; //pointeur vers descripteur de fichier  
mon_fichier=fopen("mon_fichier.txt", "w"); //ouverture en écriture (w)  
//créé le fichier si il n'existe pas  
if(mon_fichier==NULL) //gestion d'erreur  
{printf("Erreur ouverture fichier [mon_fichier.txt]\n");abort();}
```

Ouvrir un fichier (pour la lecture)

```
FILE* mon_descripteur=NULL; //pointeur vers descripteur de fichier  
mon_fichier=fopen("mon_fichier.txt", "r"); //ouverture en lecture (r)  
//le fichier doit déjà exister  
if(mon_fichier==NULL) //gestion d'erreur  
{printf("Erreur ouverture fichier [mon_fichier.txt]\n");abort();}
```



Ecriture/lecture fichier

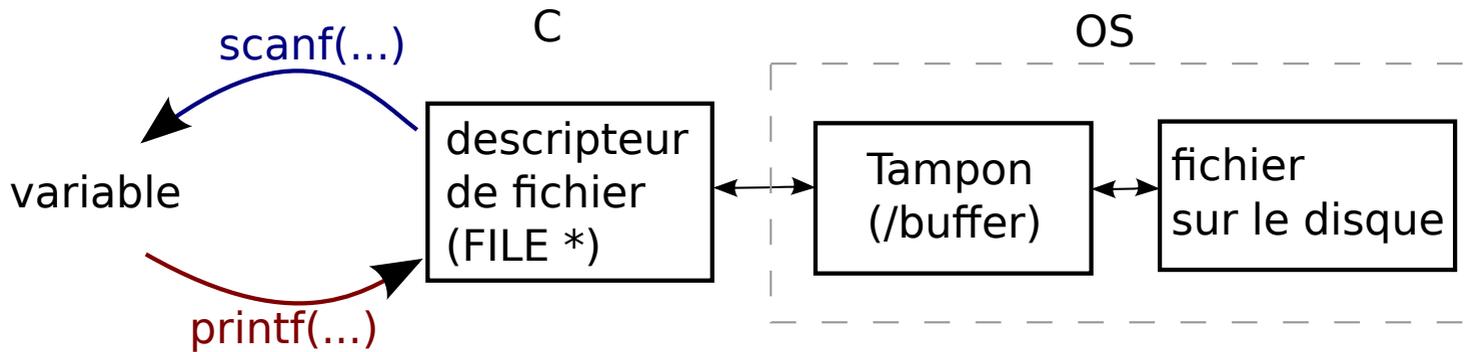


Fermer un fichier

```
int valeur=fopen(mon_descripteur); //ouverture fichier
if(valeur!=0) //gestion d'erreur
{printf("Erreur ouverture fichier \n");abort();}
```



Ecriture/lecture fichier

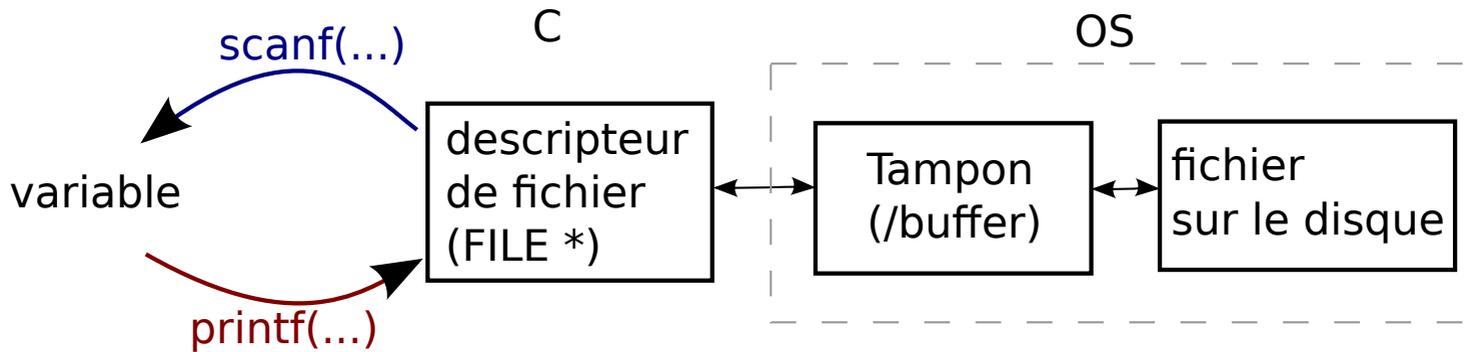


Ecrire dans un fichier *fprintf(...)*

```
int valeur=fprintf(mon_descripteur, "%s", "Mon texte a moi\n");  
if (valeur!=1) //gestion d'erreur  
{printf("Erreur ecriture fichier\n"); abort();}
```



Ecriture/lecture fichier



Lire dans un fichier *fscanf(...)*

```
char buffer[25]; //prepare un buffer de 25 cases
//lit au plus 25 caracteres
int valeur=fscanf(mon_descripteur, "%25s\n", buffer);
if(valeur!=1)//gestion d'erreur
{printf("Erreur lecture fichier\n");abort();}
```



Ecriture/lecture fichier

Lecture/ecriture par `fprintf(...)` et `fscanf(...)`

Similaire à `printf(...)`, `scanf(...)`

↑
écrit sur la sortie standard
(`stdout => écran`)

↑
lit sur l'entrée standard
(`stdin => clavier`)

Exemples minimalistes fonctionnels :
(Attention: sans gestion d'erreur)

```
#include <stdio.h>

int main()
{
    FILE *fid=fopen("mon_fichier.txt", "w");
    fprintf(fid, "prix carottes : 15 euros \n");
    fclose(fid);

    return 0;
}
```

lecture

```
#include <stdio.h>

int main()
{
    char legume[128];
    int prix=0;

    FILE *fid=fopen("mon_fichier.txt", "r");
    fscanf(fid, "prix %128s : %d euros \n", legume, &prix);
    fclose(fid);

    printf("les %s coutent %d euros\n", legume, prix);

    return 0;
}
```

écriture

Ecriture/lecture fichier

Exemples minimalistes fonctionnels : Ecriture
(Attention: sans gestion d'erreur)

```
#ifndef INCLUDE_GUARD_PERSONNE_H
#define INCLUDE_GUARD_PERSONNE_H

struct personne
{
    char nom[15];
    char fonction[15];
    int salaire;
    int anciennete;
};

#endif
```



personne.h

```
#include <stdio.h>
#include "personne.h"

int main()
{
    //base de donnees
    struct personne employe[4]={{ "Charlie", "Directeur", 55000, 3},
                                  {"Robert", "DRH", 45000, 15},
                                  {"Brigitte", "R&D", 38000, 4},
                                  {"Raymond", "Technicien", 23000, 25}};

    //ouverture fichier
    FILE *fichier=fopen("ma_base.txt", "w");

    //ecriture fichier
    int k=0;
    for(k=0;k<4;++k)
    {
        fprintf(fichier, "%s %s %d %d \n", employe[k].nom,
                employe[k].fonction,
                employe[k].salaire,
                employe[k].anciennete);
    }

    //fermeture fichier
    fclose(fichier);

    return 0;
}
```



ecrivain.h

Ecriture/lecture fichier

Exemples minimalistes fonctionnels : Ecriture
(Attention: sans gestion d'erreur)

```
#ifndef INCLUDE_GUARD_PERSONNE_H
#define INCLUDE_GUARD_PERSONNE_H

struct personne
{
    char nom[15];
    char fonction[15];
    int salaire;
    int anciennete;
};

#endif
```



personne.h

```
#include <stdio.h>
#include "personne.h"

int main()
{
    //base de donnees
    struct personne employe[4]={{ "Charlie", "Directeur", 55000, 3},
                                  {"Robert", "DRH", 45000, 15},
                                  {"Brigitte", "R&D", 38000, 4},
                                  {"Raymond", "Technicien", 23000, 25}};

    //ouverture fichier
    FILE *fichier=fopen("ma_base.txt", "w");

    //ecriture fichier
    int k=0;
    for(k=0;k<4;++k)
    {
        fprintf(fichier, "%s %s %d %d \n", employe[k].nom,
                employe[k].fonction,
                employe[k].salaire,
                employe[k].anciennete);
    }

    //fermeture fichier
    fclose(fichier);

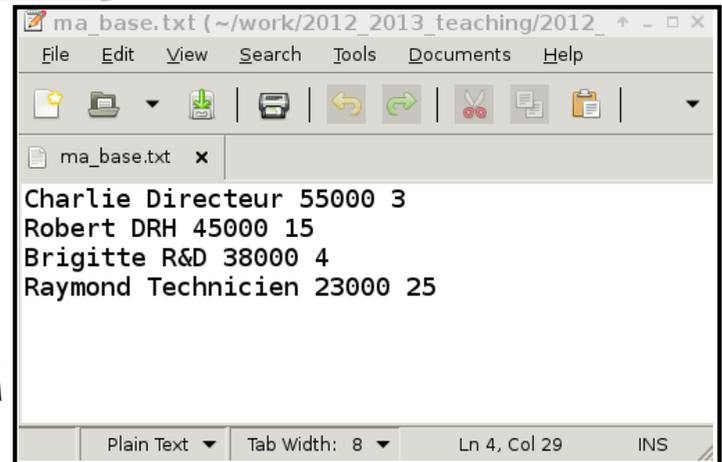
    return 0;
}
```



ecrivain.h



ma_base.txt



```
ma_base.txt (~/work/2012_2013_teaching/2012_ ...)
```

```
Charlie Directeur 55000 3
Robert DRH 45000 15
Brigitte R&D 38000 4
Raymond Technicien 23000 25
```

Plain Text Tab Width: 8 Ln 4, Col 29 INS

Ecriture/lecture fichier

Exemples minimalistes fonctionnels : Lecture
(Attention: sans gestion d'erreur)

```
#ifndef INCLUDE_GUARD_PERSONNE_H
#define INCLUDE_GUARD_PERSONNE_H

struct personne
{
    char nom[15];
    char fonction[15];
    int salaire;
    int anciennete;
};

#endif
```



personne.h

```
#include <stdio.h>
#include "personne.h"

int main()
{
    //base de donnees
    struct personne employe[4];

    //ouverture fichier
    FILE *fichier=fopen("ma_base.txt", "r");

    //ecriture fichier
    int k=0;
    for(k=0;k<4;++k)
    {
        fscanf(fichier, "%15s %15s %d %d \n", employe[k].nom,
            employe[k].fonction,
            &employe[k].salaire,
            &employe[k].anciennete);
    }

    //fermeture fichier
    fclose(fichier);

    return 0;
}
```

Ecriture/lecture fichier

Exemple ecriture fichier avec gestion d'erreur



```
int main()
{
    //base de donnees
    struct personne employe[4]={{ "Charlie", "Directeur", 55000, 3},
                                  {"Robert", "DRH", 45000, 15},
                                  {"Brigitte", "R&D", 38000, 4},
                                  {"Raymond", "Technicien", 23000, 25}};

    //ouverture fichier
    char nom_fichier[]="ma_base.txt";

    FILE *fichier=NULL;
    fichier=fopen(nom_fichier,"w");
    if(fichier==NULL)
    {printf("Erreur ouverture fichier %s\n",nom_fichier);abort();}

    //ecriture fichier
    int k=0;
    for(k=0;k<4;++k)
    {
        int verif=fprintf(fichier,"%s %s %d %d \n",employe[k].nom,
                          employe[k].fonction,
                          employe[k].salaire,
                          employe[k].anciennete);

        if(verif<=0)
        {printf("Erreur ecriture fichier %s\n",nom_fichier);abort();}
    }

    //fermeture fichier
    int ret=fclose(fichier);
    if(ret!=0)
    {printf("Erreur ouverture fichier %s\n",nom_fichier);abort();}
    fichier=NULL;

    return 0;
}
```

Ecriture/lecture fichier

Exemple lecture fichier avec gestion d'erreur



```
int main()
{
    //base de donnees
    struct personne employe[4];

    //ouverture fichier
    char nom_fichier[]="ma_base.txt";
    FILE *fichier=NULL;
    fichier=fopen(nom_fichier,"r");
    if(fichier==NULL)
    {printf("Erreur ouverture fichier %s\n",nom_fichier);abort();}

    //ecriture fichier
    int k=0;
    for(k=0;k<4;++k)
    {
        int verif=fscanf(fichier,"%15s %15s %d %d \n",employe[k].nom,
            employe[k].fonction,
            &employe[k].salaire,
            &employe[k].anciennete);
        if(verif!=4)
        {printf("Erreur lecture fichier %s\n",nom_fichier);abort();}
    }

    //fermeture fichier
    int ret=fclose(fichier);
    if(ret!=0)
    {printf("Erreur fermeture fichier %s\n",nom_fichier);abort();}
    fichier=NULL;

    return 0;
}
```

Bonnes pratiques Ecriture/Lecture

Toujours vérifier que **fopen** s'est bien déroulé



Toujours

Toujours

Toujours

- * erreur nom
- * chemin invalide
(chemin locale dépend de l'endroit de l'execution!)
- * fichier non existant
- * fichier bloqué par une autre application

Bonnes pratiques Ecriture/Lecture

Toujours vérifier que `fopen` s'est bien déroulé

Toujours

Toujours

Toujours

- * erreur nom
- * chemin invalide
(chemin locale dépend de l'endroit de l'execution!)
- * fichier non existant
- * fichier bloqué par une autre application

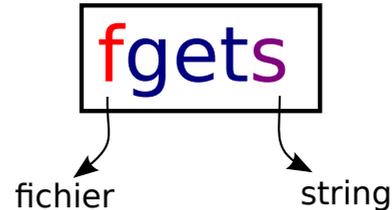
```
FILE *fid=NULL;  
fid=fopen(...);  
if(fid==NULL)  
{  
    ...  
}
```



Bonnes pratiques Ecriture/Lecture

scanf("%s",...) => s'arrête au premier espace rencontré

↳ Pour lire une ligne complète:



`fgets(char buffer[], int taille_max, FILE *fichier)`

lecture fichier

```
#define TAILLE_MAX 128

int main()
{
    FILE *fichier=NULL;
    fichier=fopen("mon_fichier", "r");

    //lit a partir d'un fichier
    char buffer_fichier[TAILLE_MAX];
    fgets(buffer_fichier, TAILLE_MAX, fichier);

    fclose(fichier);
}
```

lecture clavier

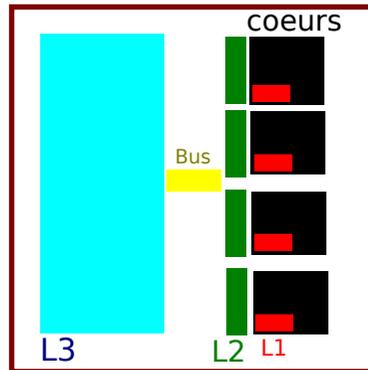
```
#define TAILLE_MAX 128

int main()
{
    //lit entree standard (clavier par default)
    char buffer_fichier[TAILLE_MAX];
    fgets(buffer_fichier, TAILLE_MAX, stdin);
}
```

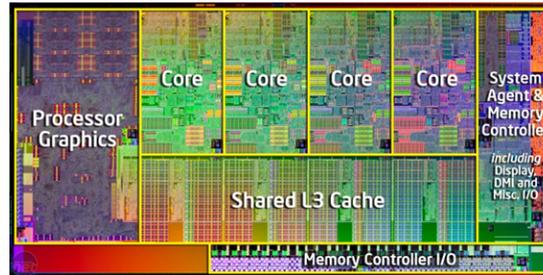
rem. Ne jamais utiliser ~~gets(...)~~
=> Non sécurisé!

Fichiers vitesse

	Transfert (Mo/s)	Tps Accès (ns)	Capacité (Mo)	Prix \$/Go
L1	60 000	0.5	0.032	on chip
L2	25 000	7	0.256	
L3	10 000	20	12	20 000 000
RAM	5 000	60	6000	20
Disque dur	150	10 000 000	1000000	0.1



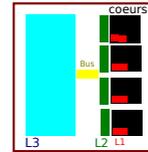
vue schématique
Processeur récent



<http://www.ni.com>

Fichiers vitesse

	Transfert (Mo/s)	Tps Accès (ns)	Capacité (Mo)	Prix \$/Go
L1	60 000	0.5	0.032	on chip
L2	25 000	7	0.256	
L3	10 000	20	12	20 000 000
RAM	5 000	60	6000	20
Disque dur	150	10 000 000	1000000	0.1



vue schématique
Processeur récent

=> Ne pas abuser de lecture/écriture nombreuses opérations

→ Passer par un buffer, puis lire/écrire par blocs dans le fichier

=> Ne pas lire caractères par caractères dans un fichier

→ chargez l'ensemble dans un buffer
puis lire dans le buffer

=> Ouvrir/fermer un même fichier trop souvent

→ Laisser le fichier ouvert tant que la fin de l'écriture n'a pas eu lieu

=> Ne pas utiliser de fichier pour des opérations critiques en temps