

### Display format

type	code	print
<code>char c='a'</code>	<code>%c</code>	a
	<code>%d</code>	97
<code>char s[]="Str"</code>	<code>%s</code>	Str
<code>int n=5</code>	<code>%d</code>	5
<code>hexa</code>	<code>%x</code>	fb
<code>octal</code>	<code>%o</code>	...
<code>float n=2.3521</code>	<code>%f</code>	2.352100
<code>%[a].</code>	<code>%6.2f</code>	___2.35
<code>[b]f/e(xpo)</code>		
a: espcé reservé	<code>%8.2f</code>	___2.35
b: decimales	<code>%.3f</code>	2.3521
<code>float n;</code> <code>n=pow(9,9);</code>	<code>%f</code>	283242953.000
	<code>%e</code>	2.824295e+011
<code>(int) *ptr;</code> <code>ptr=&amp;n;</code>	<code>"%p", ptr</code>	adr n (hex)
<code>int j=007</code>	<code>%02d</code>	07
<code>force zero avt</code>	<code>%03d</code>	007

### While Loop

`while (i >= 0)`

A while loop continues executing the while block as long as the condition in the while holds.

#### break;

The break statement immediately exits the innermost loop in which it is found.

#### continue;

The continue statement skips to the bottom of the innermost loop in which it is found and tests whether to repeat the loop again.

### Opérateur ternaire ?

sorte de if ... then ... else ...

```
int i,k,j = 3;
```

```
i = ((j == 2) ? 1 : 3) i=1;
```

```
k = ((i > j) ? i : j) k = 2 (i=1 et j=2);
```

```
i = j == 2 ? 1 : 3 ; ) (optionnels
```

### cool stuff

`sizeof(type)` poids du type en bytes

`sizeof(tableau)` poids du tableau en arg

Programmation modulaire:

`#include "nom.h"`

fichier dans le rep du projet

`#include <fichier.h>`

dans le rep "include" de l'IDE

### Allocation dynamique tab 2,3...d

`int m[3][5];` déclaration statique

`int **matrice;` déclaration dynamique

`p[i] = *(p+i)`

Les écritures suivantes sont **TOUTES** pareil:

`p[1][2] + 3` `(*(p+1)) [2] + 3`

`*(p[1] + 2) + 3` `*(*(p + 1) + 2) + 3`

### Tableaux

Ensemble de variables du **même type**

stockées les unes après les autres en mémoire.

Taille fixée avant la compilation.

Chaque case d'un tableau de type **int** contient une variable de type **int**.

Cases numérotées via **indices**. Start @ 0.

<code>int tab[3];</code>	Adresse	Valeur
<code>tab[0] = 10;</code>	1600	10
<code>tab[1] = 2;</code>	1601	2
<code>tab[2] = 6;</code>	1602	6

### Tableaux (cont)

Un tab commence à l'indice 0. Ce tab a donc les indices 0,1 et 2. Pour un tab de **n** cases, indice max = **n-1**.

La variable **tab** est un pointeur sur la première case du tableau.

`printf("%d", tab);` → 1600

Adresse de la première case (deci)

`printf("%d\n", tab[0]);` → 10

Valeur contenue dans la première case.

`printf("%d\n", *tab);` → 10

Comme **tab** est un pointeur, le cast avec \* devant le nom renvoie la valeur contenue dans la case qu'il pointe.

`printf("%d", tab[1])` et `printf("%d", *(tab + 1))`

Equivalents: Return val de la seconde case.

#### Initialisation:

`int tab[2] = {10,2,6};`

Est équivalent a la première initialisation.

`int tab[5] = {10,2};`

Initialise les cases non-attribuées a 0.

(valeurs insérées: **10,2,0,0,0**).

`int tab[100] = {0};` `int tab[100] = {1};`

Les 100 cases à 0. 1,0,0,...

### Tableaux & fonctions

```
void affiche(int *tableau, int
lenTab){
    int i;
    for (i=0; i<lenTab; i++){
        printf("%d\n", tableau[i]);
    }
}

int main(){
    int tab[4] = {1,6,3,12};
    affiche(tab, 4);...
```

La fonction prend en paramètre un pointeur sur `int(tableau)` ainsi que la taille du tableau. `void affiche(int *tableau, int lenTab)` produit le même résultat la seconde notation à l'avantage de rendre le code plus lisible en montrant que cet argument est un tableau et non pas un simple pointeur. **pas nécessaire de préciser la taille du tableau entre []**.

### Strings

char str[4];	Adresse	Valeur
str[0]='S';	18000	0x53 ('S')
str[1]='O';	18001	0x4F ('O')
str[2]='L';	18002	0x4C ('L')
end char	18003	0x00 ('\0')

`char str[]="SOL"` produit le même résultat.

**Scanf** de str: (%s et pas de &)

```
int i; char prenom[100];
scanf("%s", prenom);
for(i=0; i<strlen(prenom); i++)
    printf("%c", prenom[i]);
```

#####A COMPLETER AVEC FGSETS#####

### String exemple

### Structures

Type de variable personnalisé composé de sous-variables (et tableaux).

#### initialisation structure:

```
struct Personne {
```

```
    char nom[100], prenom[100];
```

```
    int age;};
```

#### initialisation nouveau type:

```
struct Personne vladimir;
```

#### alias de structure:

```
typedef struct Personne Employe;
```

#### initialisation nouveau type via alias:

```
Employe vladimir;
```

#### init instance + variables d'instance:

```
Employe
```

```
jo = {"", "", 0}, (virgule) (initialisation ss valeur)
```

```
claudie = {"Fokan", "Claudie", 16}, (virgule)
```

```
george = {"Berger", "George", 32}, (virgule)
```

```
a = {"Plo", "Ad", 4}; (point virgule)
```

```
("%s\n%s\n%d\n", a.nom, a.prenom, a.age);
```

### Structures (cont)

#### Tableau d'instances:

```
Employe cadres[1];
```

```
cadres[0] = claudie;
```

```
cadres[1] = george;
```

```
for(int i=0; i<2; i++)
```

```
    pf("cadre %d: %s\n", i+1, cadres[i].prenom);
```

### Structures & Pointeurs

```
1 typedef struct Coordonnees Coord; //alias
2 struct Coordonnees{//definition du type Coordonnees
3     int x; //variable x de la structure
4     int y; //variable y de la structure
5 };
6 void initCoord(Coord *point){
7     point->x = 2; //equivalent a (*point).x = 2
8     point->y = 3; //equivalent a (*point).y = 3
9 }
10 int main(){
11     Coord p; //instanciation variable p de type Coord
12     printf("Temps=0\n");
13     printf("Adr p : %p\n", &p); //0061ff28 **
14     printf("Adr p x: %p\n", &p.x); //0061ff28 **
15     printf("Adr p y: %p\n", &p.y); //0061ff2c **
16     printf("Valeur p : %d\n", p); //2699264 **
17     printf("Valeur p X: %d\n", p.x); //2699264 **
18     printf("Valeur p Y: %d\n", p.y); //4194432 **
19     initCoord(&p); //adresse de p dans la fonction
20     printf("Temps=1\n");
21     printf("Adr p : %p\n", &p); //0061ff28 **
22     printf("Adr p x: %p\n", &p.x); //0061ff28 **
23     printf("Adr p y: %p\n", &p.y); //0061ff2c **
24     printf("Valeur p : %d\n", p); //2
25     printf("Valeur p X: %d\n", p.x); //2
26     printf("Valeur p Y: %d\n", p.y); //3
27     ...// instance p de la structure Coord est un
28 } //un pointeur qui pointe sur la première variable
29 // de l'instance p(x).
30 // On travaille sur:
31 monPoint.x = 10; // Variable, on utilise le "point"
32 pointeur->x = 10; // Pointeur, on utilise la flèche
```

### enum

```
1 typedef enum Volume Volume;
2 enum Volume{
3     FAIBLE, MOYEN, FORT
4 }; //val: 0 1 2
5 typedef enum Poids Poids;
6 enum Poids{ //Deux enum ne peuvent pas
7     L=10, M=20, H=30 //avoir les même var.
8 };
9 int main(){
10     Volume musique = MOYEN;
11     printf("%d\n", musique); //1
12     Poids tam = H;
13     if(musique == MOYEN){
14         ...
15     }
16     printf("%d\n", tam); //30
17 }
```



### scanf() : Syntax

```
scanf("command string", &var1,&var2,...);
```

```
int day, month, year;
scanf("%d/%d/%d", &day,&month,&year);
```

```
=
```

```
scanf(" %d", &day);
scanf(" %d", &month);
scanf(" %d", &year);
```

### ADD A SPACE IN FRON OF THE %

When prompted to enter values:

[enter][tab] and [space] can be used to separat4e the entered values.

To finish use [enter]

```
char a, b;
scanf("%c %c", &a, &b);
```

[a][enter][b][enter] OK

[a][space][b][enter] OK

[a][b][enter] OK

When requested all char programmed need to be filled in in the prompt to go further.

### Pointeurs

Quand on déclare une variable, une case mémoire est attribuée à cette variable qui sans être initialisée vaut une valeur aléatoire.

```
int v1; C000 [4CF1] = v1
```

```
v1 = 10; C000 [000A] = v1
```

Après initialisation la case mémoire est remplacée par la valeur d'initialisation.

Quand on déclare un \*pointeur 2 cases mémoire sont attribuées. Une pour la variable **pointeur** et une autre pour l'entité \*pointeur. Comme pour une variable classique déclarée sans être initialisée **pointeur** vaut une valeur alléatoire.

```
int *pointeur; BFFC [0020] = pointeur
```

```
0020 [0000] = *pointeur
```

### Pointeurs (cont)

Un \*pointeur a besoin de deux cases mémoires avant initialisation. Une pour stocker la variable qui le référence et une autre qui stock le pointeur en lui même. L'adresse ou se trouve le pointeur en lui même est la valeur aléatoirement attribuée à la variable qui le référence.

```
C000 [000A] = v1
```

```
pointeur = &v1; C000 [000A] = *pointeur
```

```
BFFC [0020] = pointeur
```

Une fois la variable qui référence le pointeur (**pointeur**) initialisée sur une variable(**va1**, le pointeur en lui même (**\*pointeur**) "fusionne" avec cette variable. Toute modification de la valeur de **\*pointeur** modifie la valeur de **v1** et vice versa.

```
printf("adr:%p,val:%p",&*pointeur,*pointeur);
=>adr:C000,val:000A
```

```
printf("adr:%p,val:%p",&v1,v1);
=>adr:C000,val:000A
```

```
*pointeur = 20; => v1 = 20
```

```
v1 = 1020; => *pointeur = 1020
```

```
pointeur pointe sur v1 => *pointeur = v1
```

Une variable stock un nombre

Un **pointeur** stock l'adresse d'une variable

Un **\*pointeur** est l'outil qui permet ça

### Pointeurs exemple

```
int a, b, c, *ptr;
```

```
ptr = &a; //ptr pointe sur a
```

```
*ptr = 4; //a = 4
```

```
b = a + 5; //b = 9
```

```
ptr = &b; //ptr pointe sur b
```

```
c = *ptr; //c = 9
```

```
a = 4 ; b = 9 ; c = 9
```

```
char a, b, c, *ptr;
```

```
a = b = 3; //a = 3 b = 3
```

### Pointeurs exemple (cont)

```
ptr = &a; //ptr pointe sur a
```

```
c = *ptr ++ 2; //a = 5 c = 5
```

```
ptr = &c; //ptr pointe sur c
```

```
++ (*ptr); //c = 6
```

```
a = 5 ; b = 3 ; c = 6
```

### Pointeurs & fonctions

```
void decoupe(int heures, int
minutes) {
    h = m/60;
    m = m%60;}
int main() {
    int h = 0, m = 90;
    decoupe(&h, &m);
    printf("%d h et %d m\n",h, m);}
```

Les pointeur permettent de faire rendre à une fonctions plusieurs "returns" en modifiant directement depuis l'intérieur d'une fonction la valeur d'une variable dans un espace nom différent de celui de la fonction. Le "proper" return est généralement utilisé pour rendre des codes en cas d'erreur.

### double pointeurs

Pointe sur l'adresse d'un pointeur

