

# VI.

## Les entrées/sorties

Mode de communication :

**écran, clavier** (avec écho à l'écran), souris, sortie audio, ...

Les fonctions d'entrée/sortie font des conversions

- écrire à l' écran : *codage binaire -> suite de caractères « lisibles »*
- lire au clavier : *suite de touches -> codage binaire*

➡ Il y a un code de conversion pour chaque type des données :  
`int, float, ...`

les fonctions d'entrée/sortie sont déclarées dans :

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

## 1. Ecrire :

Objectif : afficher un message en insérant éventuellement des « valeurs »

Syntaxe : **printf** ( *format* [ ,*expression*] )

un *format* :

- suite de caractères entre " "
- avec des emplacements réservés par *%lettreCode*
- où *lettreCode* précise le type de la valeur (**int**, **float**, ...)

```
float prix=30.0 ;
```

```
printf("prix HT:%f€      TTC:%f€\n", prix, prix*1.2);
```

## Codes les plus courants :

- **%c** pour les `char` vus comme des caractères
- **%d** pour les `int`
- **%u** pour les `int` non signé (`unsigned`)
- **%f** pour les `float` et `double` (en virgule flottante)
- **%e** pour les `float` et `double` (en mode exposant)

## Modificateurs

- **%hd** pour les `short`
- **%ld** pour les `long`
- **%Lf** pour les `long double`
- **%5.2f** 5 caractères dont 2 après la virgule

*Voir le résumé*

## Remarques :

- les caractères à afficher sont stockés dans un **buffer** d'écriture (zone mémoire spécifique)
  - ce buffer est « recopié-vidé » à l'écran lorsque l'on écrit ' \n '
- ➔ décalage possible entre :
- l'exécution de `printf`
  - et l'affichage effectif

## Deux conseils :

- ➔ Terminez dans la mesure du possible un format avec \n
- ➔ Affichez des résultats intermédiaires (mettre des « mouchards »)

## 2. Lire

Objectif : initialiser des variables

Syntaxe : **scanf** ( *format* , *adrVariable* [ , *adrVariable* ] )

Syntaxe simplifiée de *adrVariable* : **&***variable*

Exemple :     int i, j;     float x;  
              scanf ("%d", **&i**) ;  
              scanf ("%d %f", **&j**, **&x**) ;

## Codes les plus courants :

- **%c** pour les `char` vus comme des caractères
- **%d** pour les `int`
- **%u** pour les `int` non signé (unsigned)
- **%f** pour les `float` et `double` (en virgule flottante)
- **%e** pour les `float` et `double` (en mode exposant)

## Modificateurs :

- **%hd** et **%hu** pour les `short` et les unsigned `short`
- **%ld** et **%lu** pour les `long` et les unsigned `long`
- **%lf** pour les `double`
- **%Lf** pour les `long double`

*Voir le résumé*

## Remarque sur la lecture au clavier :

- la frappe au clavier est provisoirement stockée dans un "buffer"

1	2	3		3	.	1	4
---	---	---	--	---	---	---	---

*frappe clavier*

'1'	'2'	'3'	' '	'3'	'.'	'1'	'4'				
-----	-----	-----	-----	-----	-----	-----	-----	--	--	--	--

*buffer*

- la touche <del> permet de corriger la ligne courante (buffer)
- la touche <entrée> valide la ligne ...

- et stocke '\n'

'1'	'2'	'3'	' '	'3'	'.'	'1'	'4'	'\n'			
-----	-----	-----	-----	-----	-----	-----	-----	------	--	--	--

- elle déclenche l'interprétation d'après les formats de lecture

```
scanf ("%d", &i); // lecture d'un entier
```

- les caractères non interprétés restent dans le buffer

' '	'3'	'.'	'1'	'4'	'\n'						
-----	-----	-----	-----	-----	------	--	--	--	--	--	--



## Lecture de plusieurs nombres :

```
int i; float x;
```

```
scanf ("%d", &i);
```

```
scanf ("%f", &x);
```

ou encore

```
scanf ("%d %f", &i, &x);
```

un seul espace → saut de tous les séparateurs à la lecture

Séparateurs : ' ' '\t' '\n'

**Lecture de caractères :**      `char a;`

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

ou encore

`a = getchar() ;`

**Remarque :** les séparateurs sont aussi des caractères...

La touche `<entrée>` a un double rôle :

- elle déclenche la lecture (l'analyse) de la ligne de saisie
- et elle stocke aussi le caractère `'\n'` dans le buffer de lecture

- Lire le prochain caractère (y compris un séparateur ! ) :

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

- Lire le prochain caractère non-séparateur :

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

## Lecture d'un nombre suivi d'un caractère :

```
char car;  
int t;  
printf("donnez un temps en précisant s, m ou h\n");  
scanf("%d%c", &t, &car); /* saut des séparateurs */
```

ou

```
scanf("%d%c", &t, &car); /* pas de saut de séparateur*/
```

**Ces deux écritures ne sont pas équivalentes !!!**

Il faut parfois “nettoyer” le buffer de lecture :

- après un échec de lecture en prévision des prochaines lectures
- avant de lire des caractères

➔ Réinitialiser le buffer de lecture : `rewind(stdin);`

## Résumé des contraintes définies par le format de lecture :

**Un seul blanc** entre le format de plusieurs variables ("`%d %f %c`")

➔ ignore les suites de séparateurs entre les valeurs saisies

**Un seul blanc** avant le format d'un caractère ("`%c`")

➔ ignore les séparateurs qui précèdent un caractère

**Caractères** entre le format de deux variables ("`%d coucou %f`")

➔ frappe imposée de ces caractères entre les deux valeurs saisies

**Remarque :** une erreur de lecture provoque l'abandon de la fct `scanf`

- les variables qui suivent ne sont pas lues
- les caractères non interprétés restent à lire (au prochain `scanf...`)
- on peut réinitialiser le buffer de lecture avec `rewind(stdin)`

# VII.

## Quelques compléments

# 1. Autres opérateurs

D'autres opérateurs existent mais ne sont pas indispensables

- condenser l'écriture d'expressions
- à l'origine pour optimiser le code produit par le compilateur
- souvent obsolètes avec les **optimiseurs** actuels.

Voir documentation pour :

- `+=`    `-=`    `*=`    `/=`
- *expression ? expression : expression*
- opérateurs sur les bits

Les célèbres `++` et `--` (post et pré-incrémentation)

```
int n=5 ;  
while (n-- > 0)  
    printf("%d ", n);  
affiche -> 4 3 2 1 0
```

```
int n=5 ;  
while (--n > 0)  
    printf("%d ", n);  
affiche -> 4 3 2 1
```

## 2. Conversion

Il y a plusieurs façons de convertir une valeur dans un autre type :

**Affectation** d' une variable avec une valeur d' un autre type

➔ conversion forcée

```
float x ;  
int xTronque, xArrondi;  
x=2.6 ;  
xTronque = x ;  
xArrondi = x+0.5 ;
```

## 2. Conversion

Il y a plusieurs façons de convertir une valeur dans un autre type :

**Opération** sur des variables de types différents :

➔ conversion implicite vers le type de hiérarchie la plus élevée :

**long double, double, float, long, int, short, char**

```
float orbiteTerre , Pi=3.141593 ;
```

```
long rayon=149597870; /* dist. terre-soleil */
```

```
orbiteTerre = 2*Pi*rayon;
```



## 2. Conversion

Il y a plusieurs façons de convertir une valeur dans un autre type :

**Conversion explicite :**            *(type) expression*

```
float x, partieDecimale;  
x = 3.141593 ;  
partieDecimale = x - (int) x;
```

Rappel :

```
int i=1, j=2 ;  
float x, y ;  
x = i/j ;                    /* division entiere */  
y = (float) i/j ;        /* division reelle */
```

### 3. Directives pour le préprocesseur

Le **préprocesseur** :

- modifie le texte du programme avant la compilation
- au moyen de **directives** sur une ligne
- annoncées par le caractère **#** en début de ligne.

Exemple : *les fichiers « inclus »*

**#include** <stdio.h>    *inclut les fcts d'entrée/sortie*

**#include** <math.h>    *inclut les fcts mathématiques (sqrt, sin, ...)*

**#include** <stdlib.h>

Les **macros** sans argument :

- principalement employées pour définir des constantes
- définies sur une même ligne

**#define** *identificateur* *suite-de-caractères-contenus-sur-la-ligne*

- toutes les occurrences de l'*identificateur* dans la suite du programme seront remplacées par la *suite-de-caractères-contenus-sur-la-ligne*

```
#define LARGE 640
```

```
#define HAUT 480
```

```
int nbPixels;
```

```
nbPixels = HAUT*LARGE ;
```

Conseil : parenthéser la *suite-de-caractères*

```
#define PI    22./7
float perimetre, diametre ;
scanf("%f", &perimetre);
diametre = perimetre/PI ; danger !!!
```

Conseil : parenthéser la *suite-de-caractères*

```
#define PI ( 22./7 )  
float perimetre, diametre ;  
scanf("%f",&perimetre);  
diametre = perimetre/PI ; danger !!!  
                / ( 22./7 ) ;
```

## 4. **break** et **continue** (*pour les accrocs du C*)

**break** interrompt brutalement la boucle qui le contient.

➔ Le programme se poursuit après la boucle.

**continue** provoque l'abandon de l'itération courante.

➔ La boucle se poursuit sur l'itération suivante.

```
for (i=0 ; i<100 ; i++)  
{ ...  
    if (cas particulier) continue; /*i sera incrémenté*/  
    ...  
}
```

## 5. switch par l'exemple :

### *gestion de fiches étudiant*

```
int numero;
printf("1 : saisir une fiche etudiant\n") ;
printf("2 : afficher une fiche etudiant\n") ;
printf("3 : modifier une fiche etudiant\n") ;
scanf("%d", &numero) ;
switch (numero)
{ case 1 :                                /* valeur explicite      */
    actionSaisirFiche                    /* suite d'instructions */
    break ;                             /* aller à fin switch  */
    case 2 :
        actionAfficherFiche
        break ;
    case 3 :
        actionModifierFiche
        break ;
    default : /* facultatif */
        Printf("choix non valide\n") ;
}
/* fin switch */
```