La bibliothèque standard <*stdio*> contient un ensemble de fonctions qui assurent la communication de la machine avec le monde extérieur. Dans ce chapitre, nous allons en discuter les plus importantes :

printf()	écriture formatée de données
scanf()	lecture formatée de données
putchar()	écriture d'un caractère
getchar()	lecture d'un caractère

I) Écriture formatée de données :

1) printf() :

La fonction **printf** est utilisée pour transférer du texte, des valeurs de variables ou des résultats d'expressions vers le fichier de sortie standard *stdout* (par défaut l'écran).

Ecriture formatée en langage algorithmique :

```
écrire <Expression1>,<Expression2>, ...
```

Ecriture formatée en C :

```
printf("<format>",<Expr1>,<Expr2>, ... )
```

"<format>" : *format de représentation*
<Expr1>,... : *variables et expressions dont les valeurs sont à représenter*

La partie "***<format>***" est en fait une chaîne de caractères qui peut contenir :

* du texte,
* des séquences d'échappement,
* des spécificateurs de format,
* Les **spécificateurs de format** indiquent la manière dont les valeurs des expressions <Expr1..N> sont imprimées,
* La partie "<format>" contient exactement un spécificateur de format pour chaque expression <Expr1..N>,
* Les spécificateurs de format commencent toujours par le symbole **%** et se terminent par un ou deux caractères qui indiquent le format d'impression,
* Les spécificateurs de format impliquent une conversion d'un nombre en chaîne de caractères. Ils sont encore appelés symboles de conversion.

Exemple 1 :

La suite d'instructions :

```
int A = 1234;
int B = 567;
printf("%i fois %i est %li\n", A, B, (long)A*B);
```

va afficher sur l'écran :

```
1234 fois 567 est 699678
```

Les arguments de **printf** sont

- la partie format **"%i fois %i est %li"**
- la variable **A**
- la variable **B**
- l'expression **(long)A*B**

Le 1^{er} Spécificateur (**%i**) indique que la valeur de **A**

Le 2e — Sera imprimée comme entier relatif ==>**1234**
Spécificateur (**%i**) indique que la valeur de **B**
Sera imprimée comme entier relatif ==> **567**
Le 3e — Spécificateur (**%li**) indique que la valeur de **(long)A*B** sera imprimée comme entier relatif ==> **699678**
long

Exemple 2 :

La suite d'instructions :

```
char B = 'A';
printf("Le caractère %c a le code %i !\n", B, B);
```

va afficher sur l'écran :

```
Le caractère A a le code 65 !
```

La valeur de **B** est donc affichée sous deux formats différents :

% c comme caractère : **A**
% i comme entier relatif : **65**

Spécificateurs de format pour printf :

SYMBOLE	*TYPE*	*IMPRESSION COMME*
% d ou % i	Int	entier relatif
% u	Int	entier naturel (unsigned)
% o	Int	entier exprimé en octal
% x	Int	entier exprimé en hexadécimal
% c	Int	caractère
% f	Double	rationnel en notation décimale
% e	Double	rationnel en notation scientifique
% s	Char*	chaîne de caractères

a) Arguments du type long :

Les spécificateurs **% d**, **% i**, **% u**, **% o**, **% x** peuvent seulement représenter des valeurs du type **int** ou **unsigned int**. Une valeur trop grande pour être codée dans deux octets est coupée sans avertissement si nous utilisons **% d**.

Pour pouvoir traiter correctement les arguments du type **long**, il faut utiliser les spécificateurs % **ld**, **% li**, **% lu**, **% lo**, **% lx**.

Exemple :

```
long N = 1500000;
printf("%ld, %lx", N,N);      ==> 1500000, 16e360
printf("N= %x" , N);          ==> N= e360, 16
printf("N= %d" , N);          ==> N= -7328, 22
```

b) Arguments rationnels :

Les spécificateurs **%f** et **%e** peuvent être utilisés pour représenter des arguments du type **float** ou **double**. La mantisse des nombres représentés par **%e** contient exactement un chiffre (non nul) devant le point décimal. Cette représentation s'appelle la *notation scientifique* des rationnels.

Pour pouvoir traiter correctement les arguments du type **long double**, il faut utiliser les spécificateurs **%Lf** et **%Le**.

Exemple :

```
float N = 12.1234;
double M = 12.123456789;
long double P = 15.5;
```

```
printf ("%f", N);          ==> 12.123400
printf ("%f", M);          ==> 12.123457
printf ("%e", N);          ==> 1.212340e+01
printf ("%e", M);          ==> 1.212346e+01
printf ("%Le", P);         ==> 1.550000e+01
```

c) Largeur minimale pour les entiers :

Pour les entiers, nous pouvons indiquer la *largeur minimale* de la valeur à afficher. Dans le champ ainsi réservé, les nombres sont justifiés à droite.

Exemples :

(_ <=> position libre)

```
printf ("%4d", 123);       ==>  _123
printf ("%4d", 1234);      ==>  1234
printf ("%4d", 12345);     ==>  12345
printf ("%4u", 0);         ==>  ___0
printf ("%4X", 123);       ==>  __7B
printf ("%4x", 123);       ==>  __7b
```

d) Largeur minimale et précision pour les rationnels :

Pour les rationnels, nous pouvons indiquer la *largeur minimale* de la valeur à afficher et la *précision* du nombre à afficher. La précision par défaut est fixée à six décimales. Les positions décimales sont arrondies à la valeur la plus proche.

Exemples :

```
printf ("%f", 100.123);        ==>  100.123000
printf ("%12f", 100.123);      ==>  __100.123000
printf ("%.2f", 100.123);      ==>  100.12
printf ("%5.0f", 100.123);     ==>  __100
printf ("%10.3f", 100.123);    ==>  ___100.123
printf ("%.4f", 1.23456);      ==>  1.2346
```

II) Lecture formatée de données :

1) scanf() :

La fonction **scanf** est la fonction symétrique à **printf**; elle nous offre pratiquement les mêmes conversions que **printf**, mais en sens inverse.

Lecture formatée en langage algorithmique :

```
lire <NomVariable1>,<NomVariable2>, ...
```

Lecture formatée en C :

```
scanf("<format>",<AdrVar1>,<AdrVar2>, ...)
```

"<format>" : *format de lecture des données*

<AdrVar1>,... : *adresses des variables auxquelles les données seront attribuées*

* La fonction **scanf** reçoit ses données à partir du fichier d'entrée standard *stdin* (par défaut le clavier).
* La chaîne de format détermine comment les données reçues doivent être interprétées.
* Les données reçues correctement sont mémorisées successivement aux ***adresses*** indiquées par **<AdrVar1>,...** .

* ***L'adresse d'une variable*** est indiquée par le nom de la variable précédé du signe **&**.

Exemple :

La suite d'instructions :

```
int JOUR, MOIS, ANNEE;
scanf("%i %i %i", &JOUR, &MOIS, &ANNEE);
```

lit trois entiers relatifs, séparés par des espaces, tabulations ou interlignes. Les valeurs sont attribuées respectivement aux trois variables **JOUR**, **MOIS** et **ANNEE**.

* **scanf** retourne comme résultat le nombre de données correctement reçues (type **int**).

Spécificateurs de format pour scanf :

SYMBOLE	*LECTURE D'UN(E)*	*TYPE*
% d ou % i	entier relatif	**int***
% u	entier naturel (unsigned)	**int***
% o	entier exprimé en octal	**int***
% b	entier exprimé en hexadécimal	**int***
% c	caractère	**char***
% s	chaîne de caractères	**char***
% f ou % e	rationnel en notation décimale ou exponentielle (scientifique)	**float***

Le symbole ***** indique que l'argument n'est pas une variable, mais l'*adresse* d'une variable de ce type.

a) Le type long :

Si nous voulons lire une donnée du type **long**, nous devons utiliser les spécificateurs **%ld**, **%li**, **%lu**, **%lo**, **%lx**. (Sinon, le nombre est simplement coupé à la taille de **int**).

b) Le type double :

Si nous voulons lire une donnée du type **double**, nous devons utiliser les spécificateurs **%le** ou **%lf**.

c) Le type long double :

Si nous voulons lire une donnée du type **long double**, nous devons utiliser les spécificateurs **%Le** ou **%Lf**.

d) Indication de la largeur maximale :

Pour tous les spécificateurs, nous pouvons indiquer la *largeur maximale* du champ à évaluer pour une donnée. Les chiffres qui passent au-delà du champ défini sont attribués à la prochaine variable qui sera lue !

Exemple :

Soient les instructions :

```
int A,B;
scanf("%4d %2d", &A, &B);
```

Si nous entrons le nombre **1234567**, nous obtiendrons les affectations suivantes :

```
A=1234
B=56
```

le chiffre 7 sera gardé pour la prochaine instruction de lecture.

e) Les signes d'espacement :

Lors de l'entrée des données, une suite de signes d'espacement (espaces, tabulateurs, interlignes) est évaluée comme un seul espace. Dans la chaîne de format, les symboles **\t**, **\n**, **\r** ont le même effet qu'un simple espace.

Exemple :

Pour la suite d'instructions

```
int JOUR, MOIS, ANNEE;
scanf("%i %i %i", &JOUR, &MOIS, &ANNEE);
```

les entrées suivantes sont correctes et équivalentes :

```
13 07 2020
      ou
13      007      2020
      ou
13
07
2020
```

f) Formats 'spéciaux' :

Si la chaîne de format contient aussi d'autres caractères que des signes d'espacement, alors ces symboles doivent être introduits exactement dans l'ordre indiqué.

Exemple :

La suite d'instructions :

```
int JOUR, MOIS, ANNEE;
scanf("%i/%i/%i", &JOUR, &MOIS, &ANNEE);
```

Accepte les entrées :	rejette les entrées :
13/07/2020	**13 7 2020**
13/07/02020	**13 /7 /2020**

g) Nombre de valeurs lues :

Lors de l'évaluation des données, **scanf** s'arrête si la chaîne de format a été travaillée jusqu'à la fin ou si une donnée ne correspond pas au format indiqué. **scanf** retourne comme résultat le nombre d'arguments correctement reçus et affectés.

Exemple

La suite d'instructions

```
int JOUR, MOIS, ANNEE, RECU;
RECU = scanf("%i %i %i", &JOUR, &MOIS, &ANNEE);
```

réagit de la façon suivante (- valeur indéfinie) :

Introduit :		RECU	JOUR	MOIS	ANNEE
13 7 2020	==>	**3**	**13**	**7**	**2020**
13/7/2020	==>	**1**	**13**	**-**	**-**
13.7 2020	==>	**1**	**13**	**-**	**-**
13 7 20.20	==>	**3**	**13**	**7**	**20**

III) Écriture d'un caractère :

La commande

```
putchar('a');
```

transfère le caractère *a* vers le fichier standard de sortie *stdout*. Les arguments de la fonction **putchar** sont ou bien des caractères (c'est-à-dire des nombres entiers entre 0 et 255) ou bien le symbole **EOF** *(End Of File)*.

EOF est une constante définie dans <*stdio*> qui marque la fin d'un fichier. La commande **putchar(EOF);** est utilisée dans le cas où *stdout* est dévié vers un fichier.

1) Type de l'argument :

Pour ne pas être confondue avec un caractère, la constante **EOF** doit nécessairement avoir une valeur qui sort du domaine des caractères (en général **EOF** a la valeur -1). Ainsi, les arguments de **putchar** sont par définition du type **int** et toutes les valeurs traitées par **putchar** (même celles du type **char**) sont d'abord converties en **int**.

Exemples :

```
char A = 225;
char B = '\a';
int  C = '\a';
putchar('x');    /* afficher la lettre x       */
putchar('?');    /* afficher le symbole ?      */
putchar('\n');   /* retour à la ligne          */
putchar(65);     /* afficher le symbole avec   */
                 /* le code 65  (ASCII : 'A')  */
putchar(A);      /* afficher la lettre avec    */
                 /* le code 225 (ASCII : 'ß')  */
putchar(B);      /* beep sonore                */
putchar(C);      /* beep sonore                */
putchar(EOF);    /* marquer la fin du fichier */
```

IV) Lecture d'un caractère :

Une fonction plus souvent utilisée que **putchar** est la fonction **getchar**, qui lit le prochain caractère du fichier d'entrée standard *stdin*.

1) Type du résultat :

Les valeurs retournées par **getchar** sont ou bien des caractères (0 - 255) ou bien le symbole **EOF**.

Comme la valeur du symbole **EOF** sort du domaine des caractères, le type résultat de **getchar** est **int**.

En général, **getchar** est utilisé dans une affectation :

```
int C;
C = getchar();
```

getchar lit les données de la zone tampon de *stdin* et fournit les données seulement après confirmation par 'Enter'. La bibliothèque <*conio*> contient une fonction du nom **getch** qui fournit immédiatement le prochain caractère entré au clavier.