Fiche n°1 : Données et variables scalaires

Un type scalaire décrit une donnée non décomposable : entier signé ou non signé, nombre flottant, Booléen, caractère...

Les types **`char`** et **`unsigned char`** peuvent être utilisés pour représenter de petits entiers ou des caractères imprimables.

Tous les types à valeur entières (char, int, long etc.) peuvent être utilisé pour représenter des Booléens : 0 pour **`true`**, toute autre valeur pour **`false`**.

type	Architecture 32 bits taille	bornes	Architecture 64 bits taille	bornes
`char`	1	-128 .. 127	1	-128 .. 127
`unsigned char`	1	0 .. 255	1	0 .. 255
`short`	2	-32768 .. 32767	2	-32768 .. 32767
`unsigned short`	2	0 .. 65535	2	0 .. 65535
`int`	4	-2^{31} .. $2^{31}-1$	4	-2^{31} .. $2^{31}-1$
`unsigned int`	4	0 .. $2^{32}-1$	4	0 .. $2^{32}-1$
`long`	4	-2^{31} .. $2^{31}-1$	8	-2^{63} .. $2^{63}-1$
`unsigned long`	4	0 .. $2^{32}-1$	8	0 .. $2^{64}-1$
`float`	4	$\pm 10^{-44}$ .. 10^{38}	4	$\pm 10^{-44}$ .. 10^{38}
`double`	8	$\pm 10^{-323}$ .. 10^{308}	8	$\pm 10^{-323}$ .. 10^{308}

Une **variable** se déclare en donnant d'abord un type, puis un nom ou **identificateur**.

```
int x;
int y = 12;
char z = 'A';
unsigned char w = 151;
```

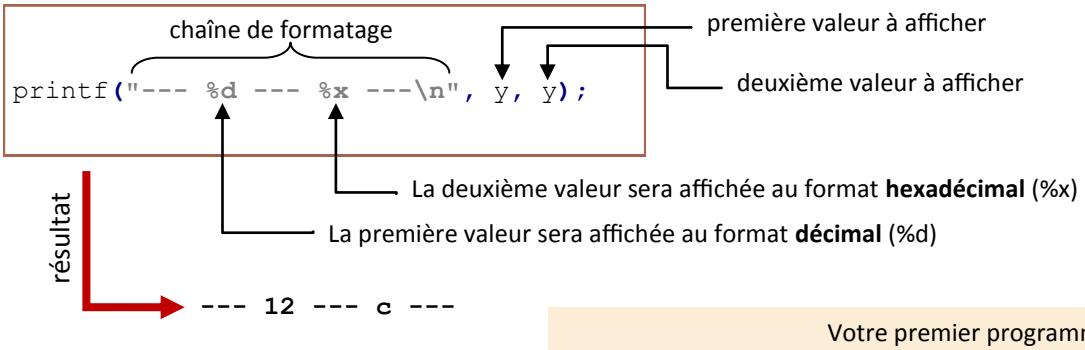
x est une variable de type **int**

y est une variable de type **int** initialisée avec la valeur 12

z est initialisée avec le code ASCII du caractère 'A'

Une variable de type caractère peut recevoir une valeur entière dans la limite de la capacité du type concerné.

Afficher une valeur à l'écran en utilisant **`printf`** : exemple avec un entier.



Autres formats d'affichage (liste non exhaustive)	
`%04x`	hexadécimal avec 0 à gauche
`%c`	caractère
`%f`	float
`%lf`	double

Votre premier programme

```c
#include <stdio.h>

int main()
{
    char z = 'A';

    printf("Le code ASCII de %c est %x\n", z, z);

    return 0;
}
```

Nombre d'octets occupés en mémoire par une variable `x` :

`sizeof(x)`

```c
int y = 12;

printf("%d\n", sizeof(x));
```

Fiche n°2 : Opérations sur les scalaires

On désigne par **types entiers** les types **`int`**, **`char`**, **`short`**, **`long`**, en version signées et non signées.

On désigne par **types flottants** les types **`float`** et **`double`**.

Règle de promotion des entiers :

Tout opérande de type **`char`**, **`unsigned char`**, **`short`**, **`unsigned short`** est converti en **`int`** de même valeur.

Des exceptions existent pour certaines architectures. (1)

opérateurs	opérandes entiers	opérandes flottants
`+ - *`	somme, différence, produit	
`/`	division entière	division
`%`	reste de la division entière	non défini

Si les opérandes sont de types différents :

- Entier et flottant : l'entier est converti en flottant.
- Entiers de tailles différentes sauf `long` et `unsigned long` : application de la règle de promotion des entiers. Le résultat est de type `unsigned int` si l'un des deux opérandes résultant est de type `unsigned int`, sinon il est de type `int`.
- L'un des opérande est un entier long : le résultat sera de type `long` ou `unsigned long`. (1)

Assignation d'une variable scalaire

```
int x = 10;

double y = 2.0;

x = x + y;

y = x + y;

x++;

int z = ++x;

z *= 2;

printf("%d %lf %d\n",x,y,z);
```

`x` est converti en `double` (10.0), puis le résultat (12.0) est reconverti en `int` (12).

`x` est converti en `double` (12.0), puis le résultat (14.0) est placé dans `y`.

raccourci pour `x = x+1.`

raccourci pour `z = z * 2.`

Attention !

`z = x++` aurait donné un résultat différent. A utiliser avec précaution. En cas de doute, décomposer en deux instructions.

raccourci pour `x = x+1; z = x.`

Quelques fonctions mathématiques (liste non exhaustive)	
`double sqrt(double x)`	racine carrée
`double pow(double x, double y)`	puissance
`double log(double x)`	logarithme népérien
`double sin(double x)`	sinus

```
#include <math.h>

int main()
{
    double z = 16;

    printf("%lf\n", sqrt(z));

    return 0;
}
```

Opérateurs Booléens (Les opérandes sont des entiers)	
`&&`	et
`||`	ou
`!`	négation

```
printf("%d %d %d\n", 1&&0, 10||12, !57);
```

résultat → **0 1 0**

En C++ et dans les versions récentes de C, il existe un type **`bool`** avec deux valeurs prédéfinies **`true`** (1) et **`false`** (0).

Fiche n°3 : Pointeurs et fonctions sur scalaires

On appelle **pointeur** une adresse mémoire.

Le **type d'un pointeur** détermine le type de la donnée pointée, c'est-à-dire de la valeur située en mémoire à l'adresse concernée.

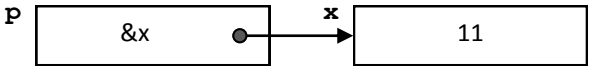
On peut déclarer des **variables de types pointeurs** destinées à contenir des pointeurs.

```
int x = 10;
```
La variable `p` est de type pointeur sur `int`.

```
int* p = &x;
```
Ceci est l'adresse de la variable `x`.

```
*p = *p + 1;
```
Ceci désigne la valeur pointée par `p`, c'est-à-dire le contenu de la variable `x`.

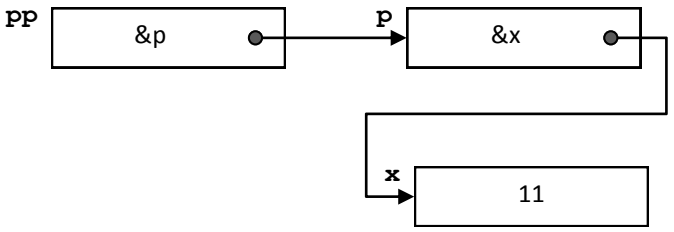
```
printf("%d\n", x);
```
La valeur 11 est affichée



Si on connait l'adresse d'une variable, on peut modifier sa valeur.

Un exemple de pointeur sur un pointeur

```
int x = 10;
int* p = &x;
int** pp = &p;
**pp = **pp + 1;
printf("%d\n", x);
```



Fonctions

Une fonction peut accepter ou non un ou plusieurs **paramètres** de types scalaires ou pointeurs. Elle peut **retourner** un pointeur, une valeur scalaire, ou rien. Dans ce dernier cas, sa valeur de retour est déclarée **`void`**.

Une fonction peut avoir ses propres **variables** dites **locales**. Les paramètres d'une fonction sont des variables locales initialisée avec des **copies** des valeurs passée lors de l'appel de cette fonction.

```
int sumsq(int a, int b)
{
    a = a+a;
    b = b*b;
    return a + b;
}
```

```
int main()
{
    int a=11,b=22;

    int c=sumsq(a,b);

    printf("%d %d %d\n",a,b,c)
    return 0;
}
```

`sumsq` ne peut pas modifier les valeurs des variables `a` et `b` car elle ne dispose que de copies de ces valeurs.

```
11 22 506
```

Cette fonction échange les valeurs de deux variables dont elle reçoit les valeurs en paramètre.

```
void exg(int* a, int* b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

En passant en paramètres les **adresses** des variables `a` et `b`, on permet à la fonction `exg` de modifier leurs valeurs.

```
int main()
{
    int a=11, b=22;
    exg(&a,&b);
    printf("%d %d\n",a,b);
    return 0;
}
```

```
22 11
```

Fiche n°4 : Saisie de valeurs et instructions de contrôle

Saisie d'une valeur au clavier

```
int x,y;

scanf("%d %d",&x,&y);

printf("%04x %04x\n",x,y);
```

La fonction **scanf** permet la saisie d'une valeur au clavier. Elle accepte en paramètre une chaine de formatage (comme la fonction d'affichage **printf**) et les **adresses** des variables dans lesquelles les valeurs saisies doivent être placées.

Attend que l'utilisateur saisisse deux valeur de type **int** et les place dans les variables **x** et **y**.

Alternatives

```
if(cond : expression à valeur Booléenne )
{
   bloc exécuté si cond est vraie
}
else
{
   bloc exécuté si cond est fausse
}
```

```
if(cond : expression à valeur Booléenne )
{
      bloc exécuté si cond est vraie
}
```

Pour mémoire, une valeur Booléenne est en fait un entier. Les accolades peuvent être omises pour les blocs constitués d'une seule instruction.

Instructions itératives

```
while( cond : expression à valeur Booléenne )
{
   bloc exécuté tant que cond est vraie
}
```

```
do
{
   bloc exécuté une première fois, puis
   tant que cond est vraie
}
while( cond : expression à valeur Booléenne );
```

Les instructions alternatives et itératives peuvent être imbriquées et / ou composées en séquences à volonté.

```
for( init , cond , maj )
{
   bloc exécuté tant que cond est vraie
}
```

raccourci pour

```
init
while( cond )
{
   bloc exécuté tant que cond est vraie
   maj
}
```

Fiche de préparation n°1

Donnez les valeurs affichées par les programmes suivants. Tentez de trouver les réponses **sans** exécuter le programme !

```
void simul()
{
    printf("%d %x\n", 100, 100);

    printf("%d\n", 'D'-'A');

    double a=1.0;
    int b=1;
    printf("%d\n", sizeof(a+b));

    int i;
    for(i=0; i<10; i++)
    {
        if(i%3)
            printf("X");
        else
            printf("-");
    }
}
```

```
void f1(int x)
{
    x++;
}

void f2(int* x)
{
    (*x)++;
}

void f12()
{
    int x=11;
    f1(x); printf("%d\n",x);
    f2(&x); printf("%d\n",x);
}
```

Complétez la fonction suivante de manière à ce que son exécution affiche le texte indiqué.

```
void tv()
{
    int i,j;
    for(i=0; i<=1; i++)

        for(j=0; j<=1; j++)

            printf(                                        );
}
```

Partie à compléter

Résultat

```
0 et 0 = 0
0 et 1 = 0
1 et 0 = 0
1 et 1 = 1
```