


# Feuille 1

## Petits exercices en C pour commencer

### 1 Valeur absolue

Le code pour cet exercice est donné ci-dessous :

```
/*
 * Calcul de la valeur absolue.
 *
 */
#include <stdio.h>

int main() {
    int n, abs, items;

    /* Saisie */
    printf("Entrer un entier: ");
    items = scanf("%d", &n);

    if (items == 1) {
        /* Calcul (compliqué...) de la valeur absolue */
        if (n < 0)
            abs = -n;
        else
            abs = n;


        /* Affichage */
        printf("|%d| = %d\n", n, abs);
    } else {
        printf("Erreur de lecture (pas un entier ou fin de fichier rencontrée)\n");
    }
    return 0;
}
```

#### Note:

Ici, on regarde le résultat de la fonction `scanf` pour voir si on a réussi à lire un nombre (en fait cette fonction renvoie le nombre de lectures qui ont réussi; ici comme on a qu'un seul '`%`', `scanf` peut renvoyer 0 ou 1 seulement).

### 2 Somme d'entiers

Pour cet exercice, on va utiliser un boucle `do` (plutôt que `while`) qui est mieux adaptée. En effet, ici on est sûr de devoir rentrer au moins une fois dans la boucle.

Le code pour cet exercice est donc :

```
/*
 * entiers.c      -- nombre d'entiers lus
 *
 */

#include <stdio.h>

int main(void) {
    int n;                /* Entier saisi */
    int cpt = 0;          /* Compteur */
    int items;            /* nombre d'éléments lus par scanf */

    do {
        printf("Entrer un entier: ");
        items = scanf("%d", &n);

        if (n > 0 && items == 1)
            /* on a bien saisi un nombre et il est positif */
            cpt += 1;
    } while (n > 0 && items == 1);

    printf("Nombre d'entiers saisis: %d\n", cpt);


    return 0;
}
```

#### Remarque:

Il est assez rare que l'on puisse utiliser la boucle **do-while** puisque, quand on doit effectuer une séquence d'actions, il est normal de traiter aussi le cas où cette liste est vide (ce que permet la boucle **while**). *A contrario*, la boucle **do-while** est bien adaptée lorsqu'on interagit avec un utilisateur, puisqu'il faut lui poser une question et recommencer si sa réponse est incorrecte ou si on a pas lu toutes les valeurs qu'il devait entrer.

## 3 Calcul du maximum d'une suite d'entiers

---

Pas de difficulté ici .

```
/*
 * max.c      -- max d'une suite d'entiers
 *
 */

#include <stdio.h>

int main(void) {
    int n, items;           /* Entier saisi et résultat de scanf */
    int cpt = 0;            /* Compteur */
    int sum = 0;            /* Somme */
    int max = -1;           /* Maximum */

    do {
        printf("Entrer un entier: ");
        items = scanf("%d", &n);

        if (items == 1 && n > 0) {
            /* Le nombre saisi est positif */
            cpt += 1;
            sum += n;
            if (n > max)
                max = n;
        }
    } while (items == 1 && n > 0);


    if (cpt > 0) {
        /* On affiche rien si aucun nombre positif n'a été saisi */
        printf("Le maximum des %d nombres saisis est %d. La somme est %d.\n",
               cpt, max, sum);
    }
    return 0;
}
```

## 4 La commande cat-num

### 4.1 Une première version

Cette première version n'est pas tout à fait correcte:

- affichage de la ligne 1 même si le fichier est vide
- affichage d'une ligne supplémentaire en fin de fichier.

La première version du programme :

```
/*
 * cat-num : affiche ligne par ligne stdin en les numérotant
 *
 */

// CETTE VERSION EST INCORRECTE:
// Si le fichier d'entrée est vide, le programme affiche tout de même
// Le chiffre 1 (alors qu'il ne devrait rien afficher).

#include <stdio.h>

int main() {
    int c;           /* caractere courant */
    int no_ligne = 1; /* No de ligne courant */

    printf("%3d ", no_ligne++);
    while ((c=getchar())!=EOF) {
        putchar(c);
        if (c == '\n') printf("%3d ", no_ligne++);
    }
    return 0;
}
```


**Note:**

On utilise ici le format `"%3d"` dans `printf`. Celui-ci permet d'afficher le numéro de ligne sur 3 caractères cadrés à droite. La fonction `printf` propose de nombreuses options accessibles après le caractère `'%'`. Pour avoir la liste de toutes ces options, sous Unix, on peut faire `bash $ man 3 printf`

## 4.2 Version correcte

Dans cette version, on va éviter les deux problèmes cités précédemment (numéro de ligne trop à la fin et affichage du chiffre 1 même si le fichier est vide).

L'idée ici est d'avoir une variable qui contient toujours le caractère précédent et d'afficher le numéro de ligne seulement quand celui-ci est le caractère `'\n'`. Pour que le numéro de la première ligne soit affiché, il suffit d'initialiser le caractère précédent à un saut de ligne. Dans cette version, les numéros de lignes sont donc affichés toujours dans la boucle. Si le fichier d'entrée est vide, on n'entre pas dans la boucle (et donc, rien n'est affiché).

La version correcte de ce programme est donc la suivante .

```

/*
 * cat-num : affiche ligne par ligne stdin en les numérotant
 *
 */

// Cette version est CORRECTE: si le fichier d'entrée est vide, rien
// n'est affiché. Principe: on affiche le numéro de ligne quand le
// caractère précédent est un newline

#include <stdio.h>

int main() {
    int c, prec_c;    /* Caractère courant, précédent */
    int no_ligne = 1; /* No de ligne courant */

    prec_c = '\n';    /* permet d'afficher la ligne 1 (si elle existe) */

    while ((c=getchar()) != EOF) {
        if (prec_c == '\n')
            printf("%3d ", no_ligne++);
        putchar(c);
        /* placer le caractère courant dans prec_c */
        prec_c = c;
    }

    return 0;
}

```

## 5 Conversion

La solution donnée ici utilise, dans un même programme, les affichages avec un **cast** (une conversion de type) et, ensuite, avec l'appel à la fonction **rint**.

### cast

Un **cast** permet de changer le type d'une expression. Par exemple

```
(int) 3.2    // résultat l'entier 3
```

permet de considérer la valeur flottante 3.2 comme un entier (ici le résultat sera 3).

### fonction rint

la fonction **rint** permet d'arrondir (et non pas tronquer comme le **cast**) un nombre flottant. Le résultat est lui-même un nombre flottant:

```
rint(3.2)    // résultat le flottant 3.0
```

Noter que pour utiliser **rint**, il faut faire l'édition de lien avec la bibliothèque mathématique (option **-lm**). La compilation du fichier peut donc se faire avec:

```
$ gcc -Wall -std=c99 -o 5-conv 5-conv.c -lm
```

Le fichier **5-conv.c**  contient les deux conversions:

```

/*                                     -*- coding: utf-8 -*-
 *
 *  Table de conversion Celsius/Fahrenheit (par cast puis par appel a rint)
 */

// Pour utiliser rint, il faut faire l'édition de lien avec la
// bibliothèque mathématique (option -lm).
// La compilation se fait donc de la façon suivante:
//      $ gcc -Wall -std=c99 -o conv conv.c -lm

#include <stdio.h>
#include <math.h>

int main() {
    printf("Avec le cast\n\n");
    printf("+-----+-----+\n");
    for (float c = 0.0; c <= 20.0; c += 0.5)
        printf("| %4.1fC | %2dF |\n", c, (int) (9*c)/5+32 );
    printf("+-----+-----+\n");

    printf("Avec le rint\n\n");
    printf("+-----+-----+\n");
    for (float c=0.0; c <= 20.0; c += 0.5)
        printf("| %4.1fC | %2.0fF |\n", c, rint((9*c)/5+32) );
    printf("+-----+-----+\n");

    return 0;
}

```

**Note:**

- Le premier format utilisé ( `%4.1f` ) permet d'afficher le résultat sur 4 caractères dont 1 après la virgule soit: 2 chiffres, un point décimal et 1 chiffre.
- Le deuxième format utilisé est `%2.0f` car on affiche un **flottant** (et pas un entier) sur 2 chiffres avec 0 chiffre après la virgule.

## 6 Comptage et wc

---

Les corrigés des deux derniers exercices (*comptage* et *wc*) seront donnés avec ceux de la feuille suivante.