C - Lecture au clavier avec `scanf`

La fonction `scanf` permet de lire des données au clavier, mais elle doit être utilisée avec soin : elle comporte plusieurs pièges (espaces, restants, dépassements…). Ce chapitre t'explique comment l'utiliser correctement *et* comment vider le buffer lorsque c'est nécessaire.

5TTR 6TTR Découverte

Objectifs

À la fin de ce cours, tu sauras :

- utiliser `scanf` pour lire les principaux types ;
- comprendre le rôle de `&` ;
- choisir les bons formats (`%d`, `%f`, `%s`, …) ;
- éviter les pièges liés au buffer ;
- **vider le buffer proprement** ;
- basculer vers `fgets` quand c'est plus fiable.

Lire les types simples

Lire un entier

```c
int age;
scanf("%d", &age);
```

Lire un flottant

```c
float moyenne;
scanf("%f", &moyenne);
```

Lire un caractère

```c
char lettre;
scanf(" %c", &lettre);
```

💡 L'espace avant `%c` permet d'ignorer le `\n` restant dans le buffer.

Lire une chaîne de caractères (`char[]`)

Sans espaces

```c
char nom[30];
scanf("%s", nom);
```

⚠️ `%s` s'arrête au premier espace.

Avec espaces ⟶ utiliser `fgets`

```c
fgets(nom, sizeof(nom), stdin);
```

Les pièges de `scanf`

Oublier le `&`

❌ `scanf("%d", age);` ✔ `scanf("%d", &age);`

Lire trop de caractères avec `%s`

❌ `scanf("%s", nom);` ✔ `scanf("%29s", nom);`

`%c` lit les retours à la ligne

✔ `scanf(" %c", &lettre);`

Mélanger formats et types → résultats incohérents

Ne jamais faire :

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

`scanf` ne vide jamais le buffer

Si tu tapes :

25⏎

→ `25` est lu → `⏎` reste en mémoire → la prochaine lecture peut capter ce `\n`

C'est une source majeure de bugs.

Comment vider le buffer correctement

Il existe plusieurs techniques selon la situation. Voici les 3 **méthodes fiables** à enseigner.

✔ Méthode 1 : Consommer tout ce qui reste jusqu'au prochain `\n`

```
int c;
while ((c = getchar()) != '\n' && c != EOF) {
    // on vide
}
```

Cette méthode :

- lit caractère par caractère,
- s'arrête quand on atteint le `\n`,
- vide proprement le buffer.

👉 C'est la méthode **classique et recommandée** en C.

✔ Méthode 2 : Appeler `fflush(stdin)` (⚠ non standard)

```c
fflush(stdin);
```

⚠ Important :

- Cela **fonctionne sous Windows / MinGW**,
- mais **n'est pas standard** → peut ne pas fonctionner ailleurs.

👉 À éviter pour un cours strictement ISO C. Mais utile à connaître si tu compiles sous Windows.

✔ Méthode 3 : Lire tout ce qui arrive dans une chaîne temporaire

Utile après une lecture incorrecte :

```c
char tampon[100];
fgets(tampon, sizeof(tampon), stdin);
```

Cela vide proprement **toute la ligne restante**.

Quand vider le buffer ?

1. **Après un** `scanf("%d")` **ou** `scanf("%f")` si tu comptes lire un `char` juste après.
2. **Après une mauvaise saisie** (ex : l'utilisateur tape du texte au lieu d'un nombre).
3. **Avant une lecture avec** `fgets`, si un `scanf` a été utilisé juste avant.
4. **Chaque fois que tu observes un comportement bizarre** où une fonction semble "sauter" une entrée.

Formats les plus utiles

Type	Format
`int`	`%d`
`float`	`%f`
`double`	`%lf`
`char`	`" %c"`
chaîne simple	`%s`
chaîne limitée	`%20s`

Utiliser `scanf` avec plusieurs valeurs

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

Vérifier si la lecture a réussi

```c
if (scanf("%d", &x) != 1) {
    printf("Erreur de saisie\n");
}
```

Alternatives plus sûres

- `fgets()`
- `sscanf()`
- parsing manuel avec `strtol()` ou `strtof()`

À retenir

- `scanf` lit dans tes variables → `&` obligatoire sauf pour les tableaux.
- `%s` est dangereux : le limiter (`"%29s"`) ou utiliser `fgets` .
- `%c` doit presque toujours avoir un espace devant.
- **Scanner laisse toujours des caractères dans le buffer** → il faut parfois le vider.
- Méthode la plus sûre : `while (getchar() != '\n');`