C - Assigner une valeur à des strings

Contrairement à Python ou JavaScript, le langage C ne possède pas un type “string” intégré. Les textes sont gérés sous forme de tableaux de caractères, ce qui change entièrement la manière de les créer, de les modifier et de les copier. Cet article t’aide à comprendre cette logique particulière.

5TTR 6TTR 4GMS Découverte

En C, un tableau de char (`char nom[30];`) n’est pas une variable assignable

Un tableau **ne peut pas être affecté** avec une simple instruction d’assignation.

Donc ceci est **illégal** :

```
1 char nom[30];
2 nom = "Alice";    // ❌ interdit
```

Pourquoi ? Parce qu’un tableau **n’est pas une variable "classique"**, c’est un bloc mémoire fixe. On ne peut pas "remplacer" un tableau : on peut seulement **écrire à l’intérieur** case par case.

Les seuls moments où un tableau peut recevoir une chaîne littérale sont :

👉 **lors de la déclaration + initialisation**

```
1 char nom[30] = "Alice";   // ✔ OK (initialisation, pas assignation)
```

👉 **ou en utilisant une fonction de copie**

```
1 strcpy(nom, "Alice");     // ✔ OK
```

Donc : 💡 *Toutes les affectations de chaînes dans un tableau déjà existant nécessitent une fonction de copie.*

Que se passe-t-il avec `strcpy()` ?

`strcpy(destination, source);`

Elle copie **caractère par caractère** :

- de `source`,
- vers `destination`,
- jusqu'au `\0` final.

Donc :

```
1 | strcpy(nom, "Alice");
```

Écrit réellement ceci en mémoire :

```
nom[0] = 'A'
nom[1] = 'l'
nom[2] = 'i'
nom[3] = 'c'
nom[4] = 'e'
nom[5] = '\0'
```

Exception importante : les *pointeurs* de type `char*`

Si tu déclares ceci :

```
1 | char *nom = "Alice";
```

Ici, tu n'as **pas** un tableau. Tu as un **pointeur** qui pointe vers une chaîne littérale *stockée en mémoire en lecture seule*.

Donc pour un `char*`, ceci est permis :

```
1 | nom = "Bob";   // ✔ valide (on change l'adresse stockée dans le pointeur)
```

Mais **tu ne peux pas modifier le contenu** de la chaîne littérale !

```
1 | nom[0] = 'C';   // ⚠ comportement indéfini → crash probable
```

Donc :

Type	Peut-on faire `nom = "Texte"` ?	Peut-on modifier le contenu ?
`char nom[30]` (tableau)	❌ Non	✔ Oui
`char *nom` (pointeur)	✔ Oui	❌ Non (dangereux)

Conclusion

Si tu utilises un tableau de char pour stocker une chaîne :

🎯 **Obligatoire** : utiliser une fonction de copie (`strcpy`, `strncpy`, `fgets`, etc.)

Parce qu'un tableau ne peut pas être affecté avec `=`.

Si tu utilises un pointeur `char*` vers un littéral :

- Tu peux faire `nom = "Alice";`
- Mais tu ne peux **pas modifier** les caractères du littéral.

Alors, quand utiliser `strcpy` ?

💡 **Toujours** quand tu veux mettre du texte dans un tableau de char :

```
1  char nom[30];
2  strcpy(nom, "Alice");      // ✔ bon
```

Ou quand tu veux copier une variable string dans une autre.

Bonus — Recommandation moderne

Pour éviter les dépassements de mémoire :

```
1  strncpy(nom, "Alice", sizeof(nom));
2  nom[sizeof(nom) - 1] = '\0';
```

Ou mieux :

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