Ft_printf

Les flag de type (%s %d %x...)

Sommaire :

Les flags dans ft_printf que nous devons gérer sont les suivants :

```
char *s = "|coucou|";
int a = 10;
char c = 'c';
```

- %s, afficher une chaîne de caractères :

```
printf("%s\n", s);
|coucou|$
```

Ne fonctionne qu'avec des chaînes, il n'est pas possible d'afficher un caractère via cette méthode. Le comportement est indéfini.

- %d, %i afficher un entier :

```
printf("%d %i\n", a, a);
10 10$
```

Ne fonctionne pas du tout avec un autre type, du genre float etc. Il faut utiliser à la place %f.

- %c, afficher un caractère :

```
printf("%c\n", c);
c$
```

A l'inverse de %s, nous ne pouvons évidemment pas récupérer un caractère à partir d'une chaîne, il faut spécifier le caractère dans la chaîne.

- %p, afficher l'adresse d'un élément (cela utilise un pointeur sur void *) sous forme héxadecimale

```
printf("%p\n", &a);
0x7ffdc857e15c$
```

- %u, afficher un nombre en unsigned int :

```
printf("%u\n", -1024);
4294966272$
```

- %x ou %X, afficher un nombre dans sa version hexadecimale, soit avec les lettres capitales, soit avec les lettres minuscules :

```
printf("%x\n", 268);
printf("%X\n", 268);
10c$
10C$
```

- %%, échapper le caractère ‘%’

```
printf("%d%%\n", a);
10%$
```

*Les flags . * - 0 (précision padding)*

Sommaire :


```
char *s = "|coucou|";
int a = 10;
char c = 'c';
```

- **Les chiffres** sont en quelque sorte des flags. En effet, lorsqu'on écrit “%5s” ou encore “%5d”, cela va ajouter vers la gauche avec un certains nombre d'espaces la chaîne de char ou le nombre.
- Le formule est, comme d'habitude, taille spécifiée - ft_strlen(arg), pour une taille spécifiée strictement supérieure à ft_strlen(arg) (ou de sa conversion aka “10” en int est plus grand que “a” en hex).
- Ainsi :

```
printf("%10s", s);
  |coucou|
printf("%10d", a);
        10
```

- Le flag “-”. C'est la même chose que pour les chiffres, à la différence près que cela va ajouter à droite selon la même formule qu'exposée plus haut.

```
printf("%-10s", s);
|coucou|
printf("%-10d", a);
10
```

- Le flag “0” est utilisé exclusivement avec des nombres. Par exemple avec un chiffre ou le flag “*” (vor plus bas pour ce flag) :

```
printf("%010x", a);
000000000a
printf("%0*X",10, a);
000000000A
```

- **Le flag “.”** : sert à faire une précision. Le flag a un comportement différent selon que l'on affiche une chaîne de char ou un nombre. Disponible avec les flags %s %d %i %x %X

```
printf("%.3s %.3i\n", s, a);
|co 010$
```

- On voit ici que lorsqu'on met une chaîne cela trime la chaîne avec le nombre spécifié. Si le nombre spécifié est plus grand que la longueur de l'argument, ce dernier est imprimé entièrement.
- Lorsqu'on rentre un nombre (entier ou hexadécimal), printf affiche le nombre précédé du chiffre spécifié - le nombre de chiffres du nombre.
  Ici 3 - 2 = 1. Donc un 0 est imprimé. Si deux était spécifié, on aurait le nombre (2 - 2 = 0).
- Si 0 est spécifié pour le chiffre 0, alors rien ne sera affiché.

```
printf(".0d\n", 0);
$
```

- **Le flag * :** c'est en fait un flag de taille. En effet, à la place d'un chiffre particulier, on peut mettre cette sorte de wildcard afin de la remplacer via un argument dans printf.
- Il fonctionne différemment selon la façon dont on l'utilise et avec quel type on l'utilise.
    - Ainsi avec le flag “.” :

```
printf("%.*s %.*i", 3, s, 3 ,a);
              Est strictement équivalent à :
printf("%.3s %.3i", s, a);
|co 010
```

- Tout seul avec %s, il y a un padding vers la droite avec pour formule : taille spécifiée - ft_strlen(va_arg) rempli d'espaces dès lors que le nombre en argument est plus grand que la taille de l'argument à padder
- Si le nombre est plus petit ou égal à la taille de l'argument à padder, alors ce dernier n'est pas paddé :

```
printf("%*s", 25, s);
                  |coucou|
```

- Si le flag est un nombre (%d, %id, %x, %X) alors à la place d'espace, nous avons des 0, selon la même formule évoqué plus haut.

```
printf("%*x", 25, a);
0000000000000000000000000a
```