ft_split

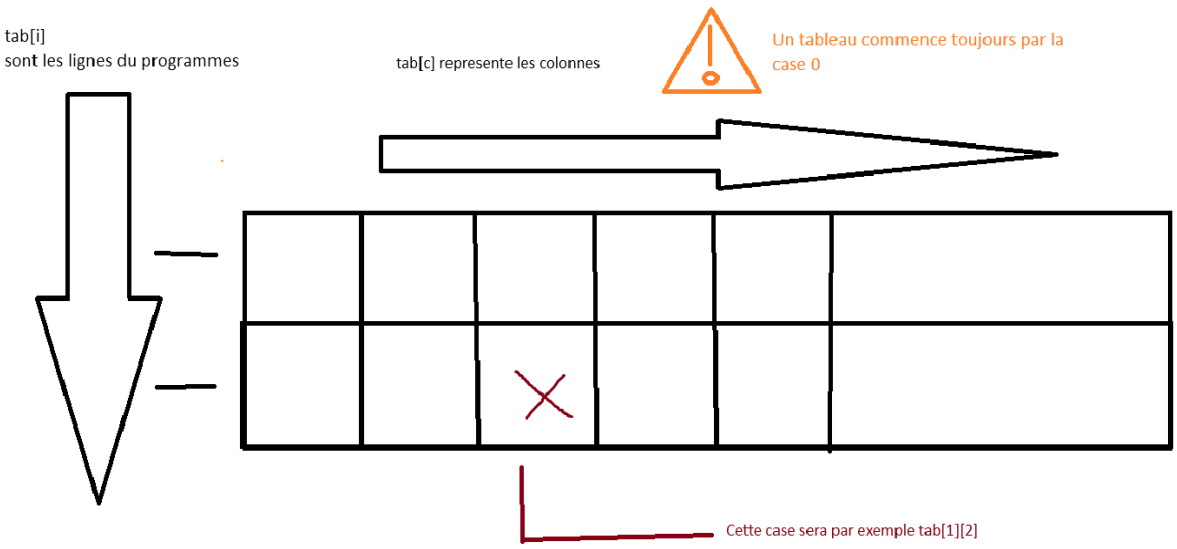
ft_split, ou la fonction dont il ne faut pas prononcer le nom (Oui c'est l'équivalent de Voldemort.) C'est une fonction qui fait peur mais qui en réalité n'est pas si complexe, je vais tenter ici, de la décortiquer et de l'expliquer de la manière la plus simple possible.

• Pour utiliser cette fonction, vous avez besoin de connaitre ces choses :

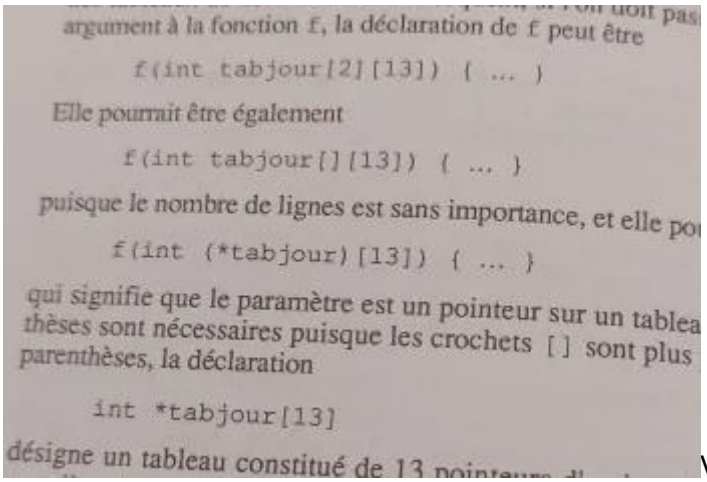
- Utilisation de ft_strlen (On va calculer des tailles de chaine de caracteres !
- Malloc (Ou l'allocation de mémoire dynamique)
- Savoir comment parcourir un tableau a 2 dimensions.

<u>Rappel concernant les tableaux à 2 dimensions</u>

Les tableaux à deux dimensions déclarées comme ceci = tab[i][c] ou encore **tab, sont des pointeurs de pointeurs, leur lecture peut paraitre difficile au début, mais rassurez vous ce n'est pas insurmontable. tab[i] va ici compter les 'lignes ' tandis que tab[c] sera là pour les colonnes.



J'ai eu du mal avec une chose lors de mon apprentissage du C, « Mais comment je fais pour accéder à la deuxième colonne ?
Il y a plusieurs manières de faire cela. Prenez cet extrait :


argument à la fonction f, la déclaration de f peut être

```
f(int tabjour[2][13]) { ... }
```

Elle pourrait être également

```
f(int tabjour[][13]) { ... }
```

puisque le nombre de lignes est sans importance, et elle po

```
f(int (*tabjour)[13]) { ... }
```

qui signifie que le paramètre est un pointeur sur un tablea
thèses sont nécessaires puisque les crochets [] sont plus
parenthèses, la déclaration

```
int *tabjour[13]
```

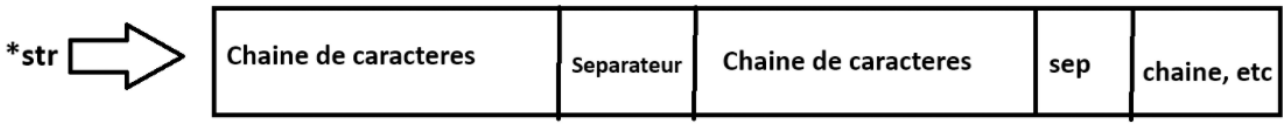
désigne un tableau constitué de 13 pointeurs d

Vous avez autant de manière de parcourir un tableau que de coder ft_putnbr, n'est-ce pas merveilleux ?

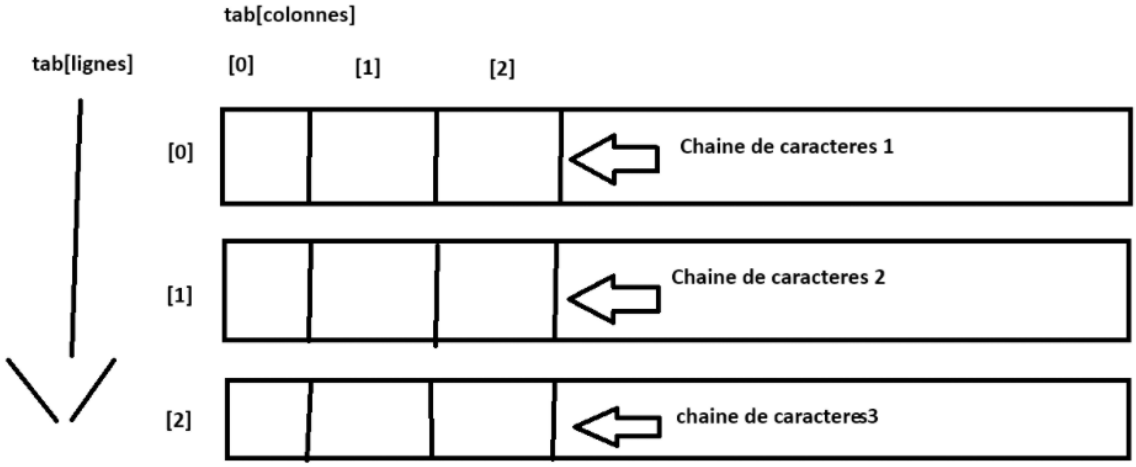
Revenons-en au fait, un tableau a deux dimensions n'est pas très complexe quand on peut se l'imaginer, garder à l'esprit qu'un tableau en langage C n'a pas grand-chose à voir avec un réel tableau.

A quoi sert ft_split ?

Ft_split, sert, à l'aide d'un séparateur, à séparer une chaine de caractères, je l'admets dis comme ça c'est flou alors voici un schéma qui sera plus parlant :



Ce qu'on veut faire de notre chaine maintenant, c'est ranger chaque chaine de caractères dans un emplacement du tableau dès que notre programme tombera sur un ou plusieurs « séparateurs », on souhaite un résultat comme celui-ci :



Maintenant que vous avez le concept, continuons. Par logique, pour pouvoir faire ceci il va nous falloir un tableau, par lequel on va allouer une taille dynamique grâce a malloc en se basant sur la longueur des chaines de caractères et du '\0' de fin. Il nous faudra également pour ça délimiter l'endroit où se trouve chaque separateur.

Reproduire ft_split

1ere étape :

Pour commencer, nous allons créer une fonction 'ft_countline' :

```c
int ft_countline(char *str, char charset)
{
    int i;
    i = 0;
    int compt = 0;

    while(str[i])
    {
        if(str[i] == charset)
        {
            compt++;
        }
        i++;
    }

    return compt + 1;

}
```

Notre première étape va être de connaitre de combien de lignes nous allons avoir besoin (voir schéma du tableau). Pour cela nous donnons deux données a notre fonction.

1. Notre chaine de caractères (*str).
2. Notre caractère qui va servir de séparateur (charset).

On initialise un ‘i’ pour parcourir notre chaine, ainsi qu’un compteur pour compter notre nombre de séparateur.
On crée une boucle qui parcourt ‘i’ entièrement, et qui rajoute 1 au compteur si elle croise un séparateur.
Pour que cela soit plus clair, notre nombre de « séparateur » va nous indiquer le nombre de lignes que nous allons devoir rajouter et compter, mais le séparateur n’étant pas en fin de chaine, on rajoute +1 au compteur.

➔ On appelle ft_countline de cette manière dans notre fonction ft_split =

```
char **tabc;
tabc = malloc(ft_countline(str, charset) * sizeof(char *)); //on multipl
if(tabc == NULL)
{
    return 0;
}
```

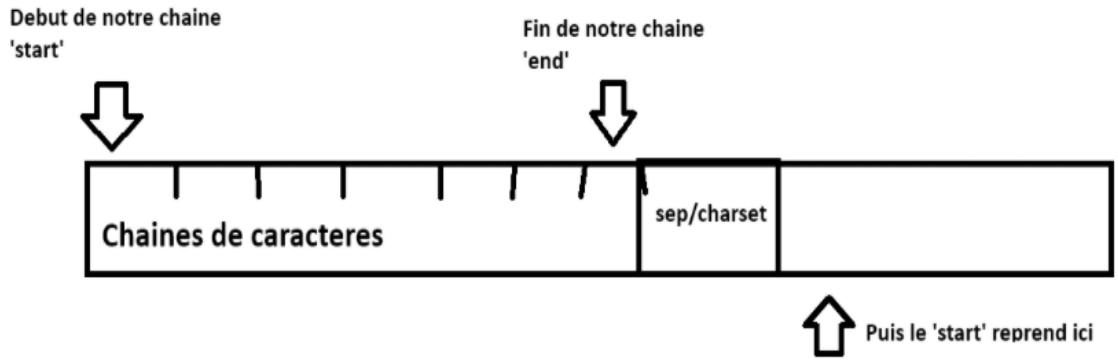
Ici, on alloue de la place pour ‘*tabc’ avec malloc, on appelle notre fonction ft_countline en lui donnant notre chaine de caractères ainsi que le séparateur, notre fonction va nous renvoyer notre résultat puis le multiplier par la taille de *char.

On effectue évidemment la vérification au cas où l’allocation de mémoire échoue.

2eme étape :

On va créer une fonction ‘ft_worldplace’, cette fonction va nous servir à connaitre le nombre de places nécessaires dont nous allons avoir besoin pour ranger nos chaines de caractères séparées (sans oublier le ‘\0’ à la fin !)

Pour cela, nous allons créer une variable qui va contenir le début de notre chaines ‘start’ et une variable qui va contenir la fin ‘end’.



Notre ‘start’ commence à zéro, notre end est l’équivalent de i – 1 ; car, quand on arrive sur notre séparateur/charset, la fin de notre chaine de caractères est une case avant ! Mais comment procéder ?

```
while(str[i])
{
    if(str[i] == charset)
    {
        end = i - 1;
        tabc[y] = ft_wordplace(str, start, end);
        y++;
        start = i + 1;
    }
    i++;
}
```

Nous créons une boucle dans notre ft_split. Tant que str (Notre chaine source) n’a pas été parcouru entièrement, on continue

Notre ‘start’ est donc à 0, tant que l’on n’a pas atteint la fin de notre chaine de caractères on continue, si on atterrit sur un séparateur on définit que la fin de chaine ‘end’ est à la place de i – 1 ; on appelle notre fonction ft_wordplace :

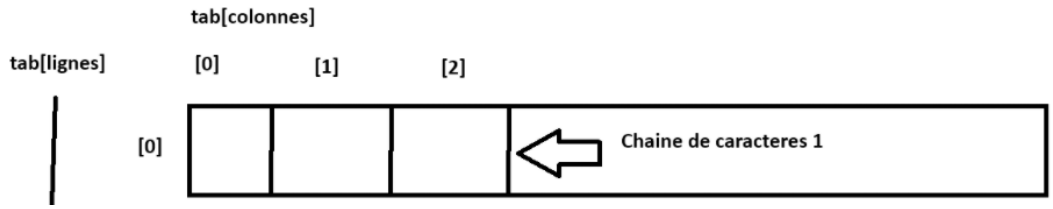
```
char    *ft_wordplace(char *str, int start, int end)
{
    char *chcar;
    int i;
    i = 0;

    chcar = malloc((end - start + 1) * sizeof(char *)); //idem que le malloc dans ft_split
    while(start <= end)
    {
        chcar[i] = str[start];
        start++;
        i++;
    }
    chcar[i] = '\0';
    return chcar;
}
```

Elle va prendre en paramètre une chaine de caractères ainsi que la valeur de ‘start’ et la valeur de ‘end’.

Dans notre fonction, on crée un char * ‘chcar’ qui va contenir la taille de notre chaine de caractères.

1. On utilise donc malloc dessus, malloc prendra en compte la taille de la chaine qu’on va calculer en faisant ‘end’ – ‘start’ + 1 (on n’oublie pas le ‘\0’ !) fois la taille d’un *char.
2. On crée une boucle pour remplir notre première chaine de caractères (Celle-ci la en dessous :>)

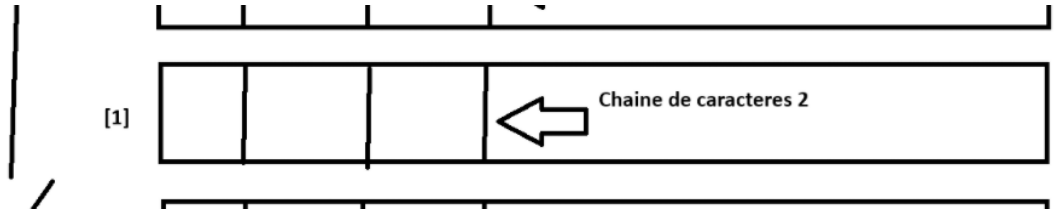


Cette boucle continuera tant que start sera inférieur ou égal à ‘end’. On copie évidemment les données de ‘str’ (à partir de start) dans notre chaine actuelle de chcar. On n’oublie pas d’incrémenter ‘i’ et ‘start’ et SURTOUT, on n’oublie pas le ‘\0’ en fin de chaine. On return notre phrase.

Suite à ça on retourne dans notre boucle de ft_split,

```
while(str[i])
{
    if(str[i] == charset)
    {
        end = i - 1;
        tabc[y] = ft_wordplace(str, start, end);
        y++;
        start = i + 1;
    }
    i++;
}
```

On incrémente [y] pour aller à notre 2eme chaine de caractères (ici)



Et on définit notre start a i + 1 ; Comme pour notre end, start va maintenant se trouver après le « séparateur » qui est la case sur laquelle on est !
On ferme notre condition.

3eme étape :

Pour finir, on n’oublie pas d’ajouter notre dernière ligne !

```
tabc[y] = ft_wordplace(str, start, i - 1); //on ajoute la derniere ligne
y++;
tabc[y] = NULL;


return tabc;
```

Notre boucle s'est arrêtée sur notre dernier 'charset/separateur' sauf que comme dit précédemment il n'y a pas de séparateur a la fin, on ajoute donc notre dernière boucle manuellement.

On appelle de nouveau notre fonction wordplace qui va délimiter le nombre de places dont on va avoir besoin, on envoie comme données notre chaine de caractère, l'endroit où se trouve notre start qui a été défini en i + 1 ; a la fin de notre boucle, et la valeur de i – 1, qui correspond au 'end', le – 1 est là pour éviter de compter le caractère NULL '\0'.

➔ On return notre ** tableau.

Voici le code source entier de la fonction ft_split :

```
char    **ft_split(char *str, char charset)
{

    int i;
    int end;
    int start;
    end = 0;
    start = 0;
    int y;
    y = 0;
    i=0;



    char **tabc;
    tabc = malloc(ft_countline(str, charset) * sizeof(char *)); //on multi
    if(tabc == NULL)
    {
        return 0;
    }
    while(str[i])
    {
        if(str[i] == charset)
        {
            end = i - 1;
            tabc[y] = ft_wordplace(str, start, end);
            y++;
            start = i + 1;
        }
        i++;
    }
    tabc[y] = ft_wordplace(str, start, i - 1); //on ajoute la derniere lig
    y++;
    tabc[y] = NULL;


    return tabc;
}
```

Si vous voulez tester votre programme, utilisez ce main :

```c
int main()
{
    char *str = "salut les amis";
    char **tabc;
    tabc = ft_split(str, ' ');
    int i = 0;
    while(tabc[i])
    {
        printf("%s\n", tabc[i]);
        i++;
    }
    //on free le tableau de tableau :
    i = 0;
    while(tabc[i])
    {
        free(tabc[i]);//on free chaque ligne
        i++;
    }
    free(tabc);//on free le tableau en lui meme
    return 0;
}
```

Et voilà, maintenant ft_split ne vous fera plus peur et vous n'en ferai plus de cauchemars la nuit !