Eléments d'Informatique

Cours10 – Chaîne de caractères, bibliothèque <string.h>

Catherine Recanati

UNIVERSITÉ PARIS 13 NORD

Plan général

- **Représentation des nombres. Notion de variable.**
- **Programme. Expressions.**
- **Architecture des ordinateurs: langage machine, langage assembleur, AMIL.**
- **Systèmes d'exploitation : fichiers, processus, compilation.**
- **Instructions de contrôle: boucles et branchements.**
- **Programme. Définition de fonction. Appel fonctionnel.**
- **Tableaux de variables et fonctions d'arguments de type tableau.**
- **Sens d'un programme, pile d'exécution, compilation.**
- **Pointeurs et tableaux.**
- **Chaines de caractères, bibliothèque <string.h>.**
- Allocation dynamique, liste chaînées.
- Révisions.

- Cours 10 –
Chaîne de caractères
bibliothèque <string.h>

- Chaines de caractères
- pointeur ou tableau
- strlen, strcpy, strcat
- lecture de chaînes
- conversions de chaînes en nombres

Plan

Il n'y pas de type « chaîne de caractères » explicite en C. Mais le langage définit les chaînes de caractères constantes, notées par une suite de caractères entre guillemets (*double quote*).

Les chaînes de caractères seront représentées par des tableaux de caractères, et la fin de la chaîne sera indiquée par le caractère spécial '`\0`' .

Plan

Chaînes constantes

Une chaîne constante est notée entre guillemets. Ex: "COUCOU".
Le caractère qui suit le dernier U en mémoire sera le caractère de fin de chaîne.

Les chaînes constantes sont invariantes en mémoire.

On peut représenter un texte par une chaîne, grâce aux caractères de tabulation, retour à la ligne, etc.

Plan

Chaînes constantes

Par exemple, le texte
« COUCOU
Comment vas-tu ? »
sera représenté par la chaîne constante
" COUCOU\nComment vas-tu ? ".

Le caractère \ est un méta-caractère (ou caractère d'échappement). Cela explique que '\n' ait une interprétation spéciale (ici un retour à la ligne).

Plan

Chaînes constantes

Les caractères attendus entre les guillemets sont des lettres de `a-z`, des chiffres de `0-9`, des caractères de ponctuation, mais aussi des caractères tels que `\n, \t, \a, \", \\`.

Si on veut faire figurer un guillemet dans la chaîne, on devra l'introduire avec `\"`. De même un caractère \ sera indiqué par `\\`.

Plan

Chaînes de caractères

On peut déclarer une chaîne de caractères `str` de deux manières :

1. comme tableau de `char`

```
char str[] =  "COUCOU";
```

ou

2. comme pointeur sur un `char`

```
char*  str = "COUCOU";
```

Dans les deux cas, la chaîne de caractères se termine par le caractère nul (à cause de son initialisation par une chaîne constante).

Plan

Chaînes de caractères

Le caractère nul en bout de chaîne permet d'afficher dans le format *string* les chaînes ou les tableau de caractères, avec `printf("%s", str)`.

Les caractères en mémoire sont affichés un à un jusqu'au premier caractère nul.

Si aucun caractère nul ne vient stopper cette impression, il y aura une erreur à l'exécution (faute d'accès mémoire et message *core dumped*).

Tableau de char

```
char  tab[]= "COUCOU" ;
```

tab	'C'	'O'	'U'	'C'	'O'	'U'	'\0'

tab[0]== 'C'
tab[1]== 'O'
...
tab[6]== '\0'

Remarque: La chaîne à 6 caractères, mais le tableau est de taille 7.

Tableau de char

ATTENTION à ne pas initialiser une chaîne de caractères de cette façon:

char tab[] = {'C','O','U','C','O','U'} ;

tab

'C'	'O'	'U'	'C'	'O'	'U'

car on aurait une erreur d'exécution en cas d'accès à tab[6], et on ne pourrait pas afficher avec `printf` le tableau tab dans le format %s (*string*).

Tableau de char

Par contre, on peut bien sûr écrire
char tab[] = {'C','O','U','C','O','U','\0'} ;
(mais c'est plus compliqué à écrire que char tab[] = "COUCOU";)

tab	'C'	'O'	'U'	'C'	'O'	'U'	'\0'

En effet, une chaîne de caractère est un tableau de caractères **dont le dernier caractère est le caractère nul.**

Tableau de `char`

On peut choisir d'imposer une taille supérieure à 7 (pour pouvoir ajouter plus tard d'autres caractères) :

```
char  tab[20]="COUCOU";
```

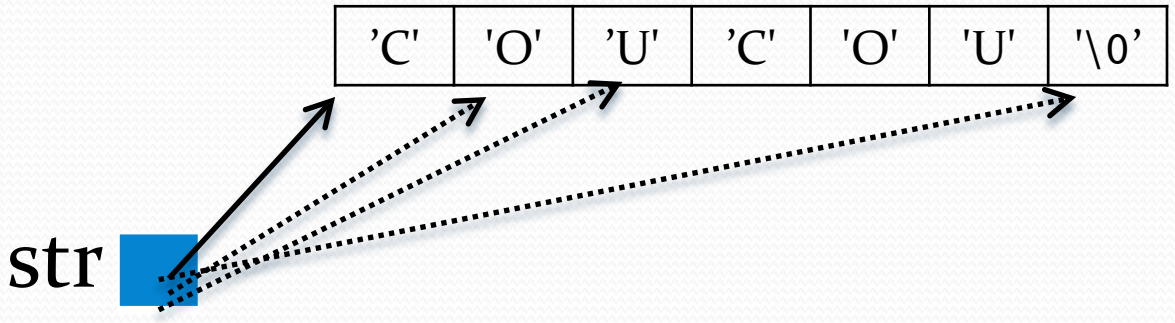
`tab`

'C'	'O'	'U'	'C'	'O'	'U'	'\0'	…	…	…		…

Mais attention à bien gérer la fin du tableau. Il sera prudent d'attribuer un caractère nul à `tab[19]`.

Pointeur sur un char

```
char * str = "COUCOU" ;
```



*str == 'C'

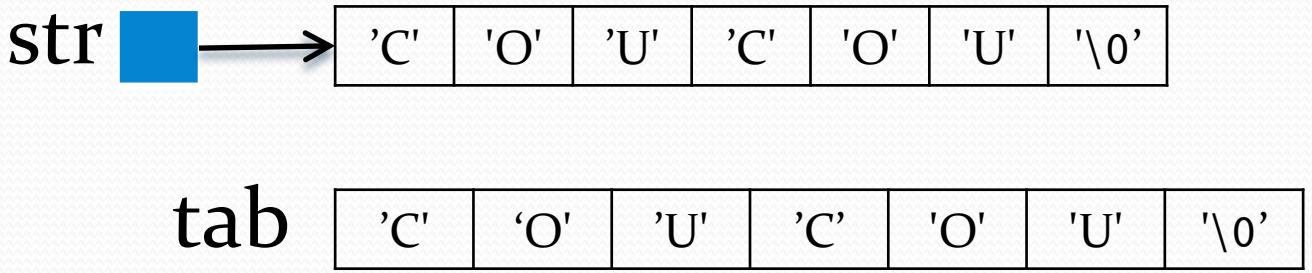
*(str+1) == 'O'

...

*(str +6) == '\0'

Pointeur sur un char

```
char * str = "COUCOU" ;
char tab[]="COUCOU" ;
```

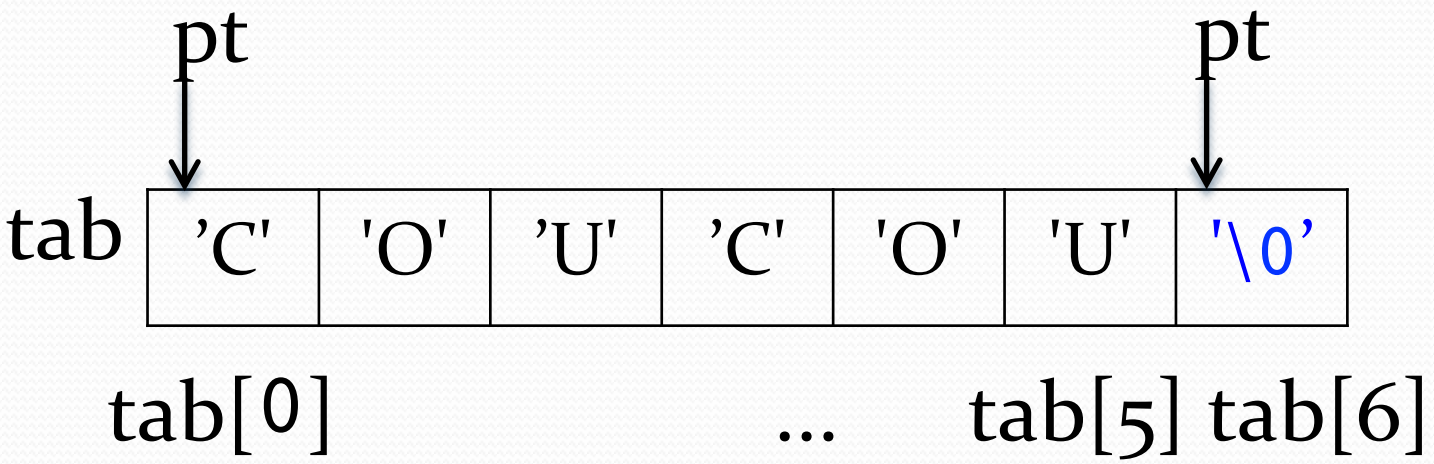


On peut modifier str, faire str++ ou bien str = tab ou str = tab + 1, mais tab est un pointeur constant et ne peut pas être modifié.

Pointeur sur un char

```
char tab[20] = "COUCOU" ;
char *pt = tab ; // pour parcourir tab
for (    ; *pt != '\0' ;   pt++ )
```

Test de fin de chaîne : (*pt == '\0')



Pointeur sur un char

1. Test de fin de chaîne : (*pt == '\0')
   On peut aussi écrire (*pt == 0) car le compilateur fera la conversion de l'entier zéro en caractère nul. On peut aussi écrire (*pt == NULL).
2. Pour déclarer un pointeur ou une chaîne de caractères, on pourra aussi écrire char *pt = NULL;

   La constante NULL est définie par zéro dans <stdlib.h>.

Plan

Pointeur sur un `char`

Les pointeurs sont un outil formidable pour parcourir des chaînes de caractères et pour écrire des fonctions manipulant des chaînes de caractères.

Nous allons maintenant voir, à titre d'exemples, quelques unes des fonctions de manipulation de chaînes de la bibliothèque <string.h>.

Plan

`int strlen(const char* str);`

La fonction `strlen` renvoie le nombre de caractères d'une chaîne de caractères passée en argument.

La chaîne étant déclarée constante, elle ne sera pas modifiée par la fonction. On pourra passer en argument une chaîne de caractères déclarée comme pointeur sur un caractère, aussi bien qu'un tableau de caractères (`char str[]`).

`int strlen(const char* str);`

Algorithme

-int nb = 0;

-Parcourir la chaîne str dans une boucle à l'aide d'un pointeur p:

tant que *p != '\0'

- à chaque tour de boucle ajouter 1 à nb

-Retourner nb en sortie de boucle

```
int strlen (const char *  str )      {
   char * p = str ; // p pointe au début
   int nb=0 ;  // pour compter les car.

   while (*p != 0) {
         // p pointe sur un char non nul
         nb++; // augmenter le compteur
         p++ ; // avancer p
   } // ici, p pointe sur le caract. nul
   return nb;
}
```

On peut supprimer p, car *p n'est jamais modifié :

```
int strlen (const char *s)
{       int nb=0 ;   // le nb de caractères
        while (*s++)
        // si c'est vrai, c'est que *s != 0
        // donc *s est un vrai caractère
                nb++;   // compter un de +
        return nb;
}
```

char * strcpy (char *dest, const char *src) ;

La fonction `strcpy` recopie une chaîne source src dans une chaîne destination `dest`. La chaîne source est déclarée constante et ne sera pas modifiée par cette fonction.

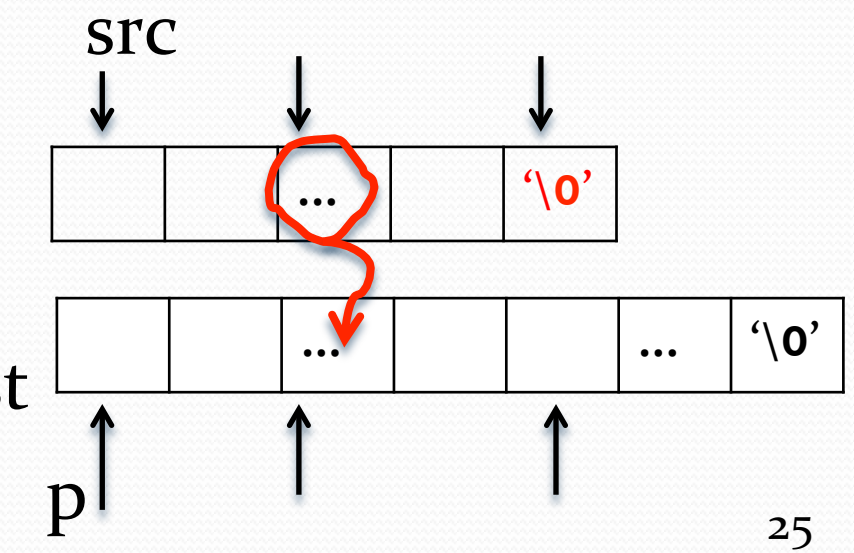
Cette fonction renvoie un pointeur sur la chaîne `dest` qui sera modifiée (et doit être assez grande pour que la copie soit possible).

char * strcpy (char *dest, const char *src) ;

Algorithme

- Parcourir, la chaîne src et la chaîne dest avec un pointeur p dans une boucle
  - recopier le contenu pointé dans dest: faire *p = *src.
- En sortie de boucle, *src == '\0' mais on ne l'a pas recopié: faire *p = *src
- Retourner dest

```
char * strcpy (char *dest,
                    const char *src) {
   char *p = dest;
   while (*src != 0) {     // on va recopier
      *p++ = *src++;       // et avancer
    }
   *p = *src ;
   return dest;
}
```



Plan

char * strcat (char *dest, const char *src) ;

Cette fonction concatène la chaine *dest* et la chaine *src*, et retourne un pointeur sur la chaine initiale *dest* - laquelle aura été modifiée et contiendra le résultat de cette mise bout à bout.

N.B. Il faut que la chaîne *dest* soit suffisamment grande (en mémoire allouée) pour que la concaténation soit possible. Mais comme pour la fonction précédente, aucun test n'est effectué pour vérifier que sa taille est assez grande.

char * strcat (char *dest, const char *src) ;

Algorithme

On va parcourir la chaîne *src* et simultanément la chaîne *dest* avec un pointeur p dans une boucle, en recopiant les caractères de *src* dans *dest* avec *p=*src.

Mais p sera initialisé pour pointer sur la fin de la chaîne *dest* (normalement il doit y avoir de l'espace alloué après).

```
char *strcat (char *dest, const char *src)
{
      char *p = dest;
      while (*p++)  // tant que *p n'est pas nul
            ;  // ne rien faire (mais on avance p dans le test)
      // ici, p pointe sur la fin de dest (caract. nul: *p == 0)
      while (*src != 0) {
            *p++ = *src++;     // recopier et avancer
       }
      *p = *src;     // on recopie le caract. nul
      return dest;
}
```

Plan

Lecture de chaînes

Un inconvénient de la fonction `scanf()` que nous avons introduite dans le TD2 est qu'elle est complexe à utiliser.

La principale difficulté vient de ce que les entrées au clavier sont rangées dans une mémoire tampon (un *buffer*) et que ce *buffer* garde les caractères non encore lus par le programme. En particulier si vous entrez un nombre suivi d'un return, le caractère return reste dans le buffer.

Lecture de chaînes

Pour remédier à ce problème, il faut vider le buffer en appelant la fonction `clean()` après un appel à scanf. La définition de cette fonction est donnée dans le cours en ligne https://openclassrooms.com/courses/utiliser-les-bonnes-fonctions-d-entree

Une deuxième difficulté pour lire des chaînes de caractères avec `scanf` est que son format de lecture %s ne peut pas lire une chaîne de caractères avec des blancs.

Lecture de chaînes

Une alternative à l'utilisation de `scanf` est `fgets`.

```
#include <stdio.h>
#include <stdlib.h>
int main ()  {
    char chaine[80] = "";

    printf("Entrez une chaine:\n");
    fgets(chaine, sizeof(chaine),
                  stdin);
    printf("Chaine: %s.\n", chaine);

    return EXIT_SUCCESS;
}
```

Conversions de chaînes

Il existe plusieurs fonctions pour convertir des chaînes de caractères (type char*) en nombre.

Dans la bibliothèque <stdlib.h>, on a

```
int atoi(char * str)
```
ascii **to** **i**nt

```
long atol(char * str)
```
ascii **to** **l**ong

```
float atof(char * str)
```
ascii **to** **f**loat

Conversions de chaînes

Dans la bibliothèque <string.h> on trouvera également

```
long strtol(const char * str);
```

string **to** **l**ong

```
double strtod(const char * str);
```

string **to** **d**ouble

```
unsigned long strtoul(const
                      char * str);
```

string **to** **u**nsigned **l**ong

Plan

- Chaînes de caractères
- Tableau de `char`
- Pointeur `sur un char`
- strlen, strcpy strcat
- Lecture de chaînes: fgets
- Conversion de chaînes de car.

Merci pour votre attention !

Des questions ?