CI 7

7- LANGAGE C

CHAINE DE CARACTERES ET ENTRÉES-SORTIES

Loïc Cuvillon
l.cuvillon@unistra.fr

Sommaire chapitre 7

Chaîne de caractères

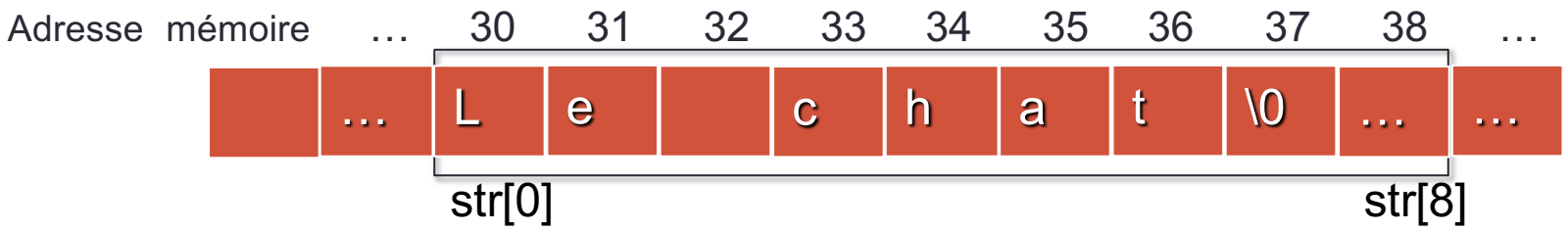
Chaîne de caractères ou « string » en anglais:

un tableau de `char` + le caractère nul `'\0'` qui indique le dernier caractère

```
char str[9]="Le chat";
```

ou

```
char str[9]={'L','e',' ','c','h','a','t','\0'};
```

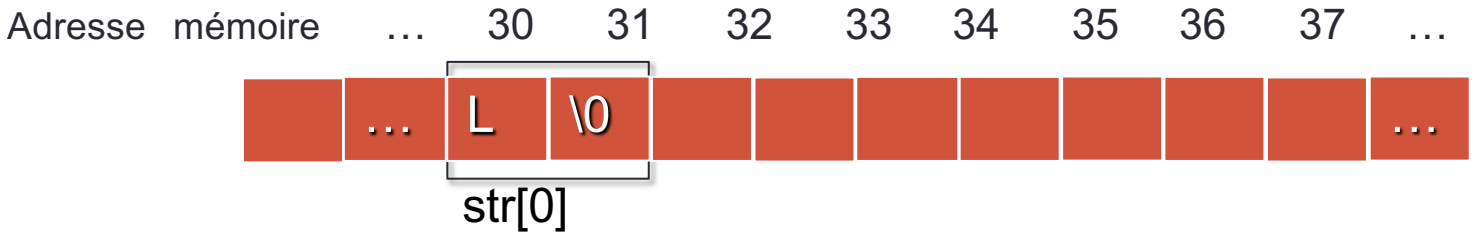


Note: un tableau de 8 éléments aurait suffi pour cette chaîne.

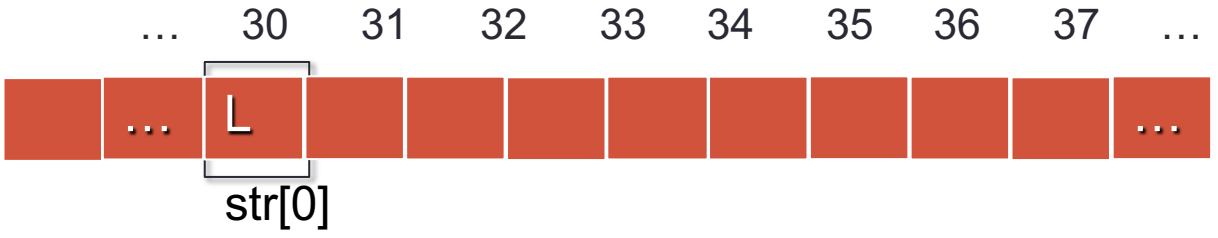
Chaîne de caractères

- Les ” ” délimitent une constante de type chaîne de caractères

- Ne pas confondre la chaîne: `char str[]=”L”;`



avec le tableau de caractère: `char str[]={’L’};`

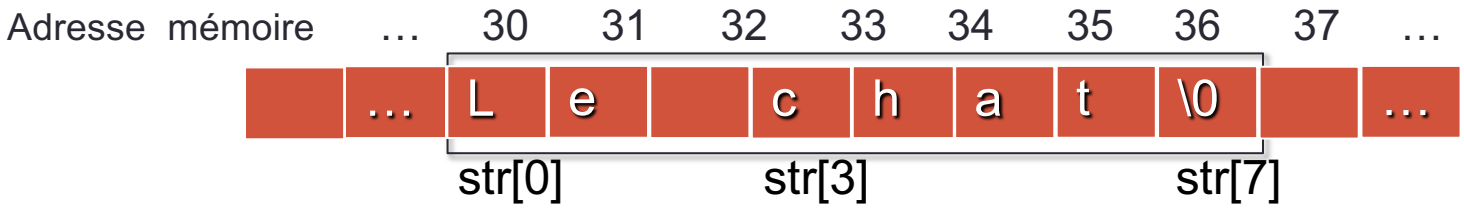


Chaîne de caractères

- Au programmeur de déclarer une taille suffisante incluant le `\0` !

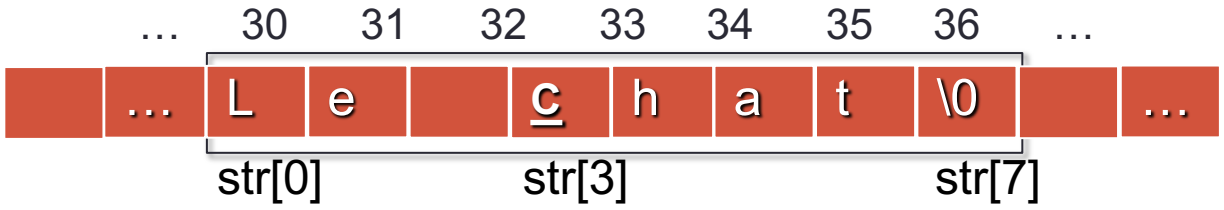
Si taille non définie, taille est calculée au plus juste à la compilation.

```
char str[]="Le chat";    → taille 8
```



- Accès aux élément de la chaîne (idem. tableau) :

```
str[3]='C';
```



Affichage d'une chaîne de caractères

```
printf("%s", char* )
```

- `%s`, format qui indique l'affichage d'une chaîne (`string`)
- **affichage caractère après caractère jusqu'à rencontrer le '\0'**
  à partir de l'adresse ou pointeur donnée par `char*`
  du tableau en paramètre

Notes:

-un rare cas où on passe à `printf` l'adresse et non la valeur de la variable

-alternative au `%s`: faire une boucle avec un `printf("%c",str[i])` et un test d'arrêt sur le '\0' .

Saisie d'une chaîne de caractères

```
scanf("%s", char* str )
```

1. **lit la mémoire tampon du clavier tant qu'un espacement (espace vide, tabulation, `\n` (newline),…)**
   n'est pas rencontré → ne lit qu'un seul mot.

2. la lecture est copiée dans la chaine passée par adresse/pointeur
   `str` (ou `&str[0]`)

3. `scanf` ajoute le '`\0`' de fin de chaîne

Note: `scanf` n'admet que des adresses de variables car il doit modifier la variable qui lui est passé ☺

Exemple

```
char str1[6];

printf("Votre nom?:\n");

scanf("%s",str1);

printf("Hello %s !\n",str1);
```

A l'exécution :


```
Votre nom?:

loic dupont ↵
```

(note: `dupont` reste dans la mémoire du clavier jusqu'au prochain `scanf`)

```
Hello loic !
```

Remplir une chaîne de caractères

- Initialisation lors de la déclaration uniquement:

```
char str[]=”This is a string.”;
char str[80]={’a’,’ ’,’s’,’t’,’r’,’i’,’n’,’g’,’\0’};
```

- Ensuite invalide,

```
str = ”This is new content”;
```

← erreur compilation ☹

rappel : `str` est le nom d’un tableau, soit l’adresse constante du premier élément ←→ `&str[0] = ”This is new content”;`

→ il faut copier élément par élément ou utiliser une fonction qui s’en charge (comme pour tout tableau en C)

Remplir une chaîne de caractères

avec `#include<stdio.h>`:

- `scanf("%s",str);` à partir d'une saisie au clavier.
- `sprintf(char* dest ,"format",var1, var2,… );`

idem. `printf` mais écrit dans la chaîne `dest` au lieu du terminal.

```
sprintf(str,"This is new content");

char name[]="Jo";
sprintf(str,"Hello %s %i",name, 1);
```

Remplir une chaîne de caractères

avec `#include<string.h>`:

- `strcpy(char *dest, char *src);`

[string copy] , copie la source `src` dans la destination `dest`.
Il faut que `dest` soit déclaré assez grand pour la copie.

```
strcpy(str, "This is new");
strcpy(str2, str );
```

Note: Pas besoin de préciser le nombre de caractères de `src` à copier, le `\0` indique la fin !
(C'est l'idée astucieuse des chaînes et du caractère spécial nul \0.)

Remplir une chaîne de caractères

- Ecrire sa propre fonction `string_copy()` :

```
void string_copy(char *dest, char *src)
{
      int i=0;
            /*ici version compacte: copie dans la condition*/
            /* copie src[i] dans dest[i] puis test sur dest[i]*/
      while ( (dest[i]=src[i]) != '\0')
            {
                  i++;
            }
}
```

Manipulation des chaînes de caractères

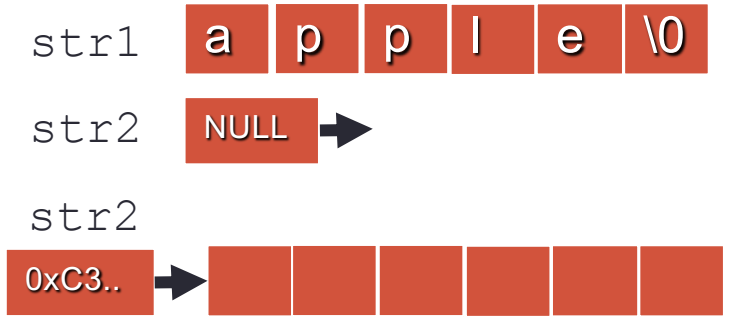
avec `#include <string.h>`

`strlen(char* s1)`	donne la longueur de la chaîne `s1` (le '`\0`' est omis dans le décompte)
`strcat(char *s1, char *s2)`	concatène, ajoute `s2` à la fin de `s1`
`strchr(char *s1, char ch)`	retourne (l'adresse) un pointeur sur la première occurrence de `ch` dans `s1`. Sinon retourne `NULL`.
`strcmp(char* s1, char* s2)`	retourne la valeur 0 si `s1` et `s2` ont un contenu identique. Rappel: le test ~~`(s1 == s2)`~~ est un test entre les pointeurs (adresses) constant sur le 1er élément du tableau et non leur contenu.
…	…

```
#include <string.h>

char str1[]="apple";
char *str2=NULL;

str2= (char*)
   malloc( (strlen(str1)+1) *sizeof(char));


scanf("%s",str2);


if (strcmp(str1,str2)==0)
    { printf("str1 et str2 identiques");}
else
    { printf("str1 et str2 différents");}
```

Note: `if ( !strcmp(str1,str2))` fonctionne aussi

strcmp exemple :



saisie clavier : apple ↵

affiche:

str1 et str2 identiques

```
#include <stdio.h>
#include <string.h>
  …
char str2[10], msg[20]="il etait";
char *position=NULL;
int date=1628;


sprintf(str2," en %d",date);



strcat(msg,str2);

position=strchr(msg,'1');
if (position != NULL)
    { printf("%s\n",position); }
  …
```

`strcat`, `strchr` exemple :

`msg` il etait\0

← pointeur pour mémoriser une adresse

← conversion d'un entier (codé binaire) en 4 caractères ASCII : 1,6,2 et 8

`str2` en 1628\0

`msg` il etait en 1628\0

← `position` vaut `&msg[12]`

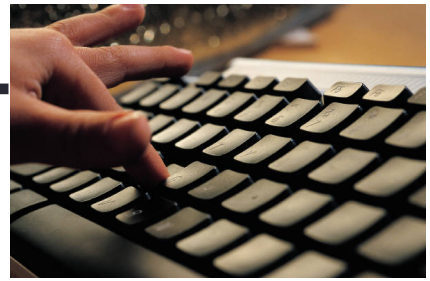
← affiche `1628`

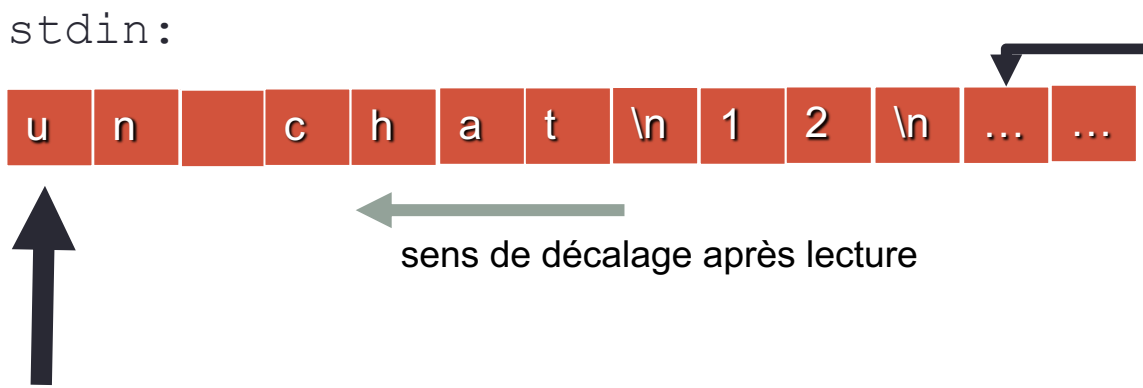
Sommaire chapitre 7

Entrées formatées: `stdin`

- `stdin` (standard input) est la mémoire tampon du clavier.

Pression touche clavier [entrée]:
1. ajoute un `\n` au texte saisi
2. ajoute le texte à la suite dans `stdin`





`scanf()`,`getchar()`, `fgets()`…

- lit dans `stdin` et retire ce qu'elle a lu.
- la fonction attend une saisie clavier seulement si `stdin` est vide, sinon lit immédiatement dedans

Entrées formatées: `scanf`

```
int scanf(char format[], arg1,…)
```

- lit les caractères de l'entrée standard (`stdin`) et les interprète en fonction du format spécifié

- où `arg1, arg2,…` sont des adresses de variables
  - `&var`, si de type simple (`char`, `int`, `double`…)
  - `str` (≅ `&str[0]`), si un tableau de caractères

- retourne le nombre de variable ayant été assignées ou `EOF`
  - `EOF(End Of File)` indique une erreur ou fin de fichier.

scanf

`int scanf(char format[], arg1,…)`

- supprime automatiquement tous les espacements au début de `stdin` lors de la lecture (sauf si lecture d'un caractère : `%c`)
  - Espacement: un ou plusieurs caractères ' ', '\n', '\t', …
- arrête la lecture dès que le contenu de `stdin` ne correspond plus au format

Problème avec `scanf` ?

- vérifier le code de retour de `scanf` pour savoir si une valeur a été assignée à toutes les variables.

```
int main()
{
    int value=0, nb_lu=0;

    nb_lu=scanf("%d", &value);

    printf("%d var(s) lue(s) :%d", nb_lu, value);
    return 0;
}
```

A l'exécution: la saisie `23` ↵ donne l'affichage `1 var(s) lue(s):23`

la saisie incorrecte `er` ↵ donne l'affichage `0 var(s) lue(s):0`

scanf

Ne pas mettre d'espacement à la fin du format !

```
scanf("%i   ", );
scanf("%i\n", );
```

sinon `scanf` attend la marque de fin d'un espacement, soit le début d'une seconde saisie pour retourner.

`scanf` et la lecture d'un caractère

- Exception : Lors de la lecture d'un caractère, un retour à la ligne ou un espace en début de `stdin` n'est pas ignoré par `scanf`.

- D'où le problème pour saisir un caractère :

```
int number;
char c, e;

printf("Entrer un entier:");

scanf("%d",&number);


printf("Entrer un char:");
scanf("%c",&c);


printf("Lu: *%d* *c*",number, c);
```

Saisie au clavier :

12 ⏎

`stdin`

`stdin après scanf("%d",)`

← Le `\n` restant dans `stdin` est lu sans attente de saisie!

← Affiche sur 2 lignes :

```
Lu : *12* *
*
```

scanf et la lecture d'un caractère

- Solution : retirer de stdin le /n

```
int number;
char c, e;


printf("Entrer un entier:");

scanf("%d",&number);


scanf("%c",&c);

printf("Entrer un char:");
scanf("%c",&c);

printf("Lu: *%d* *c*",number, c);
```

Saisie au clavier :

12 ⏎

stdin

stdin après scanf("%d",)

stdin après 1er scanf("%c",)

stdin vide : attente saisie

a ⏎

affichage :

Lu : *12* *a*

`scanf` et la lecture d'un caractère

- Solution : vider tout le contenu de `stdin` avant une lecture

```
int number;
char c, e;

printf("Entrer un entier:");

scanf("%d",&number);


e='\0'; /*initialisation !*/
while (e != '\n')
  { scanf("%c", &e); }

printf("Entrer un char:");
scanf("%c",&c);

printf("Lu: *%d* *c*",number, c);
```

Entrée-sortie dans une chaîne

On peut lire à partir d'une chaîne de caractères.

- `int sscanf(char *string, char *format, arg1,… );`
  - même format que `scanf()`
  - l'entrée est lue de la chaîne **`string`**
  - retourne le nombre d'objets lus ou une erreur (valeur négative)

```
int number;
char animal[30];
char str[]="12 canaris";

/* extraire de str le nombre  et l'animal : */
sscanf(str,"%d %s",&number, animal);
```

fgets

Pour remplacer `gets`, et lire une ligne entière de manière sûre :

```
char* fgets( char *s, int num, FILE *stream);
```

avec `num`, la taille du tableau `s`.

1. la fonction lit dans le fichier désigné par `stream`
   au plus `num`-1 caractères ou s'arrête plus tôt si un '`\n`' est rencontré.

2. le résultat de la lecture (dont l'éventuel '`\n`') est copié
   dans la châine de caractère `s`.

Note: On peut lire le clavier en utilisant `stdin` (standard input) comme fichier.

fgets

```
#include <stdio.h>
#define MAX_SIZE 100

int main()
{int date;
 char str[MAX_SIZE];
 char name[20];

 fgets(str,MAX_SIZE, stdin);
 sscanf(str,"%s %d",name,
&date);



 printf("%d-%s\n",date,name);

 return 0; }
```

A l'exécution:

`Skylwalker 1984` ↵ (saisie)

`1984-Skylwalker`

`fgets + sscanf` est la façon recommandée (évite les problèmes de `scanf`) de lire à l'écran quelque soit ce que l'on cherche à lire:

- toute une ligne dont le `\n` est lue → `stdin` est vidé ☺
- évite le dépassement de tableaux