TEXTES

1. Chaînes de caractères
2. Copie de chaînes
3. Constantes
4. Opérations
5. Recherches
6. Entrées/sorties
7. `gets/puts`
8. Conversions
9. Liste de chaînes
10. Saisie de liste
11. Arguments de `main`
12. Macros avec chaînes
13. Le type fonction
14. Pointeur sur fonction
15. Tableau de fonctions
16. Fonction en paramètre

Chaînes de caractères

- Une chaîne est un tableau de caractères terminé par le caractère `NUL` (`'\0'`, entier zéro)

b	o	n	j	o	u	r	\0		
0	1	2	3	4	5	6	7	8	9

- Comme un tableau ordinaire

```
char car, chaine[100];
char salut[10] =
   {'b', 'o', 'n', 'j', 'o', 'u', 'r', '\0'};
...
car = salut[3];
...
```

- Initialisation simplifiée

```
char salut[10] = "bonjour";
...
car = salut[3];
...
```

- Le caractère `NUL` est automatiquement mis en fin de chaîne dans les cas suivants :
  — initialisation par constante
  — lecture par `scanf`
  — fonctions standards (attention aux strn...)
  (on peut affecter directement la valeur `'\0'`)

Copie de chaînes

- `strcpy(char * s, const char * t)` de `<string.h>` effectue la copie de `t` vers `s`
- Trois fonctions équivalentes

```
void copie1(char s[], char t[]){
     int i = 0;
     while (s[i] = t[i])
         i++;
}
void copie2(char s[], char t[]){
     int i = 0;
     while (*(s+i) = *(t+i))
         i++;
}
void copie3(char * s, char * t){
     while (*s++ = *t++)
         ;
}
```

- Noter que `t[i]` est évalué comme `*(t+i)`
- Le marqueur de fin de chaîne est important pour le test : "`while (*s++ = *t++)`" est la forme réduite de "`while ((*s++ = *t++) != '\0')`" qui contient une affectation

- L'argument d'appel correspondant à s doit être un tableau de taille suffisante

```
...
void copie (char * s, char * t) {
   while (*s++ = *t++)
      ;
}
...
char * source = "Bonjour";
char tab[10];
char * poi;
...
/* copie "Bonjour" dans tab */
copie(tab, "Bonjour");
/* idem */
copie(tab, source);
...
/* copie "Bonjour" dans un tableau
   de taille 8 point\'e par poi    */
poi = (char *) malloc(strlen(source)+1));
copie(poi, source);
...
```

Constantes

- Une constante de chaîne est une suite de caractères entre guillemets (")
- Concaténation à la compilation :
  `"Bon" "jour"` est équivalent à `"Bonjour"`
- Elle est de type "tableau de caractères", représentée en mémoire avec le marqueur de fin `'\0'`)
- Une constante de chaîne n'est pas modifiable

Exemple

```
char * x = "Bonjour";

main(){
     char chaine[80];
     scanf("%s", chaine);
     if (strcmp(x, chaine) == 0)
          printf("C'est \"gagne\"\n");
     else printf("C'est \"perdu\"\n");
}
```

Chaînes et pointeurs

- `char chaine[50] = "Bonjour";`
  définit et initialise le tableau `chaine`; l'utilisateur doit prévoir une taille suffisante (≥ 8, ici)
- `char chaine[] = "Bonjour";`
  idem, sauf allocation du nombre minimum de caractères nécessaires par le compilateur (**8**, ici)
- `chaine` est un pointeur (comme tout tableau en C) de type `char *`
- Le contenu du tableau "`chaine`" est modifiable, mais le pointeur "`chaine`" est constant
- `char * x = "Bonjour";`
  définit et initialise le pointeur `x` sur la constante
- La valeur du pointeur `x` est modifiable, mais la chaîne sur laquelle il pointe est constante

```
...
char chaine[50] = "Bonjour";
char * x = "Bonjour";
...
/*** possible ***/
if(*chaine == 'B') ...... chaine[3] = 't';
if(*x == 'B') ...... x++;
...
/*** illegal ****/
chaine++; ...... *(x+3) = 't';
...
```

Opérations

```
size_t strlen(const char *chaine);
```

- Longueur
  `strlen("Bonjour")` vaut 7

```
char *strcpy(char *destination, const char *source);
```

- Copie (comme précédemment)

```
char *strcat(char *destination, const char *source);
```

- Concaténation

```
...
char d[50] = "Bonjour";
char * s = " a tous !";
char * p;
...
p = strcat(d, s);
/* le tableau d contient "Bonjour a tous !" */
/* p pointe sur d  (egalite des pointeurs)  */
...
```

```
int strcmp(const char *x, const char *y);
```

- Comparaison dans l'ordre lexicographique

$$\texttt{strcmp(x, y)} \text{ est } \begin{cases} < 0 & \text{si } \texttt{x} < \texttt{y} \\ = 0 & \text{si } \texttt{x} = \texttt{y} \\ > 0 & \text{si } \texttt{x} > \texttt{y} \end{cases}$$

```
char *strncpy(char *destination, const char *source,
            size_t n);
char *strncat(char *destination, const char *source,
            size_t n);
int strncmp(const char *x, const char *y,
            size_t n);
```

- Comme les fonctions précédentes mais travaillent sur un préfixe de longueur `n` de `source`, `x`, et `y`

Recherches

```
char *strchr(const char *chaine, int car);
```

- Recherche d'un caractère : donne l'adresse dans `chaine` de la première occurrence de `car`, `NULL` s'il n'apparaît pas

```
if(strchr("aeiouy",car)) printf("VOYELLE !");
```

```
char *strrchr(const char *chaine, int car);
```

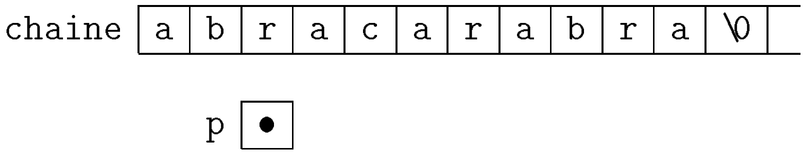
- Recherche d'un caractère : donne l'adresse dans `chaine` de la dernière occurrence de `car`, `NULL` s'il n'apparaît pas

```
printf("Nom de base : %s\n", strrchr(nom_de_fichier,'/')+1);
```

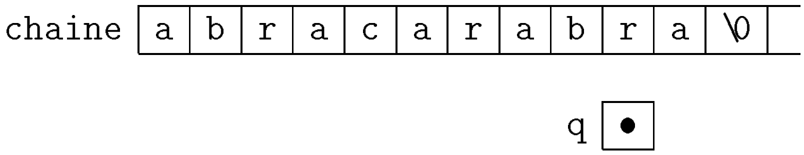
```
char *strstr(const char *chaine, const char *motif);
```

- Recherche d'une chaîne : donne l'adresse dans `chaine` du premier caractère de la première occurrence de `motif`, `NULL` s'il n'apparaît pas

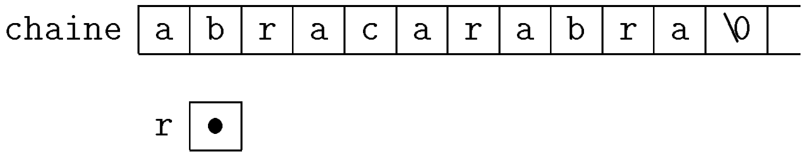
```
char chaine[50] = "abracarabra";
p = strchr(chaine, 'r');
```




```
char chaine[50] = "abracarabra";
q = strrchr(chaine, 'r');
```




```
char chaine[50] = "abracarabra";
r = strstr(chaine, "bra");
```



Entrées/sorties

- `scanf("%s", chaine);`
- `prinft("%s", chaine);`
- `char *gets(char *chaine);`

  Place la chaîne entrée au clavier dans la tableau `chaine`; remplace le caractère retour par `'\0'`; a pour valeur `chaine`, ou `NULL` en fin de fichier ou en cas d'erreur.
- `int puts(const char *chaine);`

  Affiche à l'écran `chaine` suivie de retour; a pour valeur EOF en cas d'erreur, ou une valeur ≥ 0 sinon.

gets/puts

Entré au clavier (fichier `stdin`),

`abracadabra`retour

lu par

```
gets(chaine);
```

produit en mémoire

chaine	a	b	r	a	c	a	d	a	b	r	a	\0	

qui affiché par

```
puts(chaine);
```

produit à l'écran (fichier `stdout`)

`abracadabra`retour

Conversions

- Utilisation de `sscanf` de `<stdio.h>`
- Fonctions de `<stdlib.h>`

```
double strtod
     (const char *chaine, char **reste);
long strtol
     (const char *chaine, char **reste, int base);
unsigned long strtoul
     (const char *chaine, char **reste, int base);
```

- "string to ?"

  Conversion numérique d'un préfixe de `chaine` (aprés élimination des blancs préfixes) respectivement en `double`, `long`, et `unsigned long`
  `reste` pointe sur le reste de la chaîne;
  la base (2 à 36) est nécessaire pour les entiers.

```
double atof(const char *);
int atoi(const char *);
long atol(const char *);
```

- Anciennes fonctions conservées en C ANSI

  `atof(s)` équivaut à `strtod(s, (char**)NULL)`,
  `atoi(s)` à `(int)strtol(s, (char**)NULL, 10)`,
  `atol(s)` à `strtol(s, (char**)NULL, 10)`.

Exemple

```
# include <stdio.h>
# include <stdlib.h>

char ligne[81];
double d; long l; unsigned long u;

main(){
   char * * p;
   while ( *p = gets(ligne) ) {
      puts(ligne);
      d = strtod(*p, p);
      printf("DOUBLE = %g\n", d); puts(*p);
      l = strtol(*p, p, 10);
      printf("LONG = %ld\n", l); puts(*p);
      u = strtoul(*p, p, 10);
      printf("UNSIGNED LONG = %lu\n", u); puts(*p);
   }
}
```

```
12.34E-1 -1234 1234567taratata
DOUBLE = 1.234
-1234 1234567taratata
LONG = -1234
1234567taratata
UNSIGNED LONG = 1234567
taratata
```

Liste de chaînes

Deux représentations possibles

- **Tableau à deux dimensions**

`char table[3][10];`

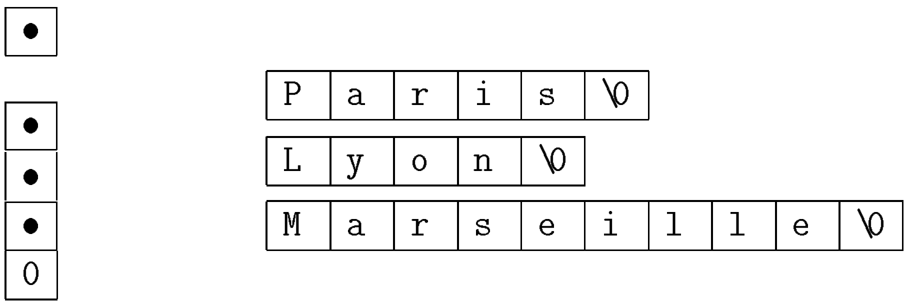
`table`

P	a	r	i	s	\0				
L	y	o	n	\0					
M	a	r	s	e	i	l	l	e	\0

- **Pointeurs sur chaînes**
  pour une gestion dynamique de la mémoire

`char * * liste;`

`liste`



Exemple

```
# include <stdio.h>
# include <stdlib.h>
# include <string.h>

main(){
   char * * liste = saisie();
   char * * p;
   int k;

   puts("DEBUT");
   for(p = liste, k = 1;  *p;  p++, k++)
      printf("%d -->%s<--\n", k, *p);
   puts("FIN");
}
```

```
DEBUT
1 -->Paris<--
2 -->Lyon<--
3 -->Marseille<--
FIN
```

Saisie de liste

Avec gestion dynamique du tas (mais sans contrôle du nombre de lignes, ni de la lecture d'une ligne, ni de l'allocation)

```
char * * saisie(void){
   char * * liste; char * * p;
   char ligne[81];

/* pour saisir au plus 10 lignes */
   liste = (char * *) calloc(11, sizeof(char *));

   for(p = liste; gets(ligne); p++){
      *p = (char *) malloc(strlen(ligne)+1);
      strcpy(*p, ligne);
   }
   *p = NULL;

   return liste;
}
```

Arguments de `main`

- "`main`" admet des paramètres (traditionnellement appelés `agrc` et `argv`) qui permettent de lire la ligne de commande qui lance le programme

```
int main(int argc, char * argv[]);
```

- Les deux arguments sont : le nombre de chaînes qui constituent la ligne de commande (séparées par des blancs), et leur liste

```
sort -u aclasser -o classe
```

argc `5`

argv

•	s	o	r	t	\0				
•	-	u	\0						
•	a	c	l	a	s	s	e	r	\0
•	-	o	\0						
•	c	l	a	s	s	e	\0		
0									

Exemple

Affichage de la ligne de commande, trois versions

```
int main(int argc, char * argv[]){
   int i;
   for(i = 0; i < argc; i++)
      printf("%s%s", argv[i], (i<argc-1)?" ":"\n");
}
```

```
int main(int argc, char * argv[]){
   while(argc-- > 0)
      printf((argc > 0)?"%s ":"%s\n", *argv++);
}
```

```
int main(int argc, char * argv[]){
   while(*argv != NULL)
      printf("%s ", *argv++);
   putchar('\n');
}
```

NOTE : les `"> 0"` et `"!= NULL` sont facultatifs.
La dernière version affiche un blanc en trop.

Macros avec chaînes

- Le préprocesseur ne fait pas de remplacement dans les chaînes

```
# define debogue(var, format) \
        printf("var = %format\n", var)
...
debogue(somme, f);
```

produit après pré-traitement

```
printf("var = %format\n", somme);
```

- Un paramètre précédé de "#" est remplacé (sans le "#") encadré d'apostrophes doubles (")

```
# define debogue(var, format) \
        printf(#var " = %" #format "\n", var)
...
debogue(somme, f);
```

produit après pré-traitement

```
printf("somme" " = %" "f" "\n", somme);
```

- La concaténation a lieu à la compilation

```
printf("somme = %f\n", somme);
```

Le type fonction

- Déclaration par prototypage :

```
double surf_rect(double a, double b);
```

- Valeur de retour : seuls types possibles : arithmétique, structure, union, pointeur, `void`
  Ni fonction, ni tableau
- Appel : par le nom suivi de la liste (éventuellement vide) des arguments *entre parenthèses*
- Adresse : non suivi de parenthèse dans une expression, le nom est un *pointeur constant* sur la première instruction du code (texte) de la fonction : son adresse

  Permet de manipuler les fonctions par pointeurs : fonctions en paramètres, en valeur de retour, en tableaux, ...
- Déclaration d'un pointeur sur fonction :

```
double (* ptr_fonc) (double, double);
```

Exemple

```
double surf_rect(double a, double b){
   return a*b;
}
double surf_tria(double a, double b){
   return a*b/2;
}
double surf_disq(double a, double b){
   return 3.14*a*a;
}
```

- Fonction qui gère le cas de figure pour calculer la surface

```
...
typedef enum {rectangle, triangle, disque} forme;

double surf1(forme x, double a, double b){
   switch(x){
      case rectangle : return surf_rect(a, b);
      case triangle  : return surf_tria(a, b);
      case disque    : return surf_disq(a, b);
   }
}
...
...
... surf1(triangle, x, y) ...
...
```

Pointeur sur fonction

```
...
double surf2(forme x, double a, double b){

   double (* fonc) (double, double);

   switch(x){
      case rectangle : fonc = surf_rect; break;
      case triangle  : fonc = surf_tria; break;
      case disque    : fonc = surf_disq; break;
   }
   return (* fonc)(a, b);
/* ou return fonc(a, b); */
}
...
...
... surf2(triangle, x, y) ...
...
```

Tableau de fonctions

```
double surf_rect(double a, double b){
   return a*b;
}
double surf_tria(double a, double b){
   return a*b/2;
}
double surf_disq(double a, double b){
   return 3.14*a*a;
}

typedef enum {rectangle, triangle, disque} forme;

double (* tab_fonc[3]) (double, double) =
               {surf_rect, surf_tria, surf_disq};

# define surf3(x, a, b) (* tab_fonc[x])((a), (b))
...
...
... surf3(triangle, x, y) ...
...
```

Fonction en paramètre

```
double surf_rect(double a, double b){
   return a*b;
}
double surf_tria(double a, double b){
   return a*b/2;
}
double surf_disq(double a, double b){
   return 3.14*a*a;
}

double surf4(double fonc(double, double),
             double a, double b){
   return fonc(a, b);
}
...
...
... surf4(surf_tria, x, y) ...
...
```

Ou Mieux

```
# define surf4(f, a, b) (* f)((a), (b))
```

EXERCICES

1. **Standards** : programmer les fonctions standards "`strlen`", "`strcmp`", et "`atoi`".

2. **Lecture de ligne** : programmer la fonction

   ```
   int getline(char * ligne, int max);
   ```

   qui mémorise au plus `max` caractères de la ligne d'entrée (clavier) dans le tebleau `ligne`, et prend pour valeur la longueur de la chaîne mémorisée.

3. **Liste** : écrire une fonction de saisie de liste de chaînes basée sur `getline`.
   Écrire une fonction d'affichage de liste de chaînes.

4. **Écart** : programmer la fonction `ecart` qui fournit le nombre de positions où deux chaînes diffèrent.
   Par exemple, `ecart("arche", "arrhes") = 2`.

5. **Recherche approchée** : écrire une fonction de recherche qui produit les chaînes d'une liste qui ont un écart inférieur à 2 avec une chaîne donnée.

6. **Miroir** : le miroir d'une chaîne x est la chaîne obtenue par lecture de x de droite à gauche.
   Par exemple, `miroir("fois") = "soif"`.

   Programmer la fonction `miroir`.

Standards

```
int longueur(char * chaine){
   int i = 0;
   while(*chaine++) i++;
   return i;
}

int comparaison(char * s, char * t){
   for( ; *s && *t; s++, t++)
      if(*s != *t) break;
   if(*s < *t) return -1;
   else if(*s == *t) return 0;
   else return 1;
}

int entier(char * chaine){
   int signe = 1, x = 0;
   while(isspace(*chaine)) chaine++;
   if(*chaine == '-') {
      signe = -1; chaine++;
   }
   else if(*chaine == '+') chaine++;
   while(isdigit(*chaine))
      x = x*10 + *chaine++ - '0';
   return signe*x;
}
```

Lecture de ligne

```
int getline(char ligne[], int max){
   int n, c;

   if((c=getchar()) == EOF) return -1;
   else ungetc(c, stdin);

   for(n=0;
       n<max && (c=getchar())!=EOF && c!='\n'; ++n)
      ligne[n] = c;
   while(c!='\n' && (c=getchar())!=EOF)
      ;
   ligne[n] = '\0';
   return n;
}
```

- Version basée sur `fgets` :

```
int getline(char *ligne, int max){
   if(fgets(ligne,max,stdin)==NULL) return -1;
   else return strlen(ligne);
}
```

Liste

```
void affichage(char * * liste){
   while(*liste)
      puts(*liste++);
}


char * * saisie(){
   char * * liste; char * * p;
   char ligne[81];
   int lg;

   liste = (char * *) calloc(11, sizeof(char *));

   for(p=liste; (lg=getline(ligne,80))!=-1; p++){
      *p = (char *) malloc(lg+1);
      strcpy(*p, ligne);
   }
   *p = NULL;

   return liste;
}
```

Recherche approchée

```
int ecart(char *x, char *y){
   int k = 0;
   while(*x && *y)
      if(*x++ != *y++) k++;
   while(*x++) k++;
   while(*y++) k++;
   return k;
}


char * * recherche(char * chaine, char * * liste){
   char * * proches =
            (char * *) calloc(11, sizeof(char *));
   char * * p = proches;

   for( ; *liste; liste++)
      if((k = ecart(chaine, *liste)) <= 2)
         *p++ = *liste;
   *p = NULL;

   return proches;
}
```

Miroir

```
# include <stdio.h>
# include <stdlib.h>

char * mir_rec(char * image, char * chaine){
   if(*chaine){
      image = mir_rec(image, chaine+1);
      *image++ = *chaine;
   }
   return image;
}

char * miroir(char * chaine){
   char * image = (char *)malloc(strlen(chaine)+1);
   *mir_rec(image, chaine) = '\0';
   return image;
}

main(){
   char chaine[81];
   while(gets(chaine)){
      puts(chaine);
      puts(miroir(chaine));
   }
}
```

Deux versions itératives

```
char * miroir(char * chaine){
   int n = strlen(chaine);
   char * image = (char *) malloc((size_t)n+1);
   char * p = image;

   for(n = strlen(chaine); n > 0; n--)
      *p++ = chaine[n-1];
   *p = '\0';

   return image;
}
```

```
char * miroir(char * chaine){
   int n = strlen(chaine);
   char * image = (char *) malloc((size_t)n+1);

   image = image+n;
   *image = '\0';

   for( ; *chaine; chaine++)
      *--image = *chaine;

   return image;
}
```