Rim Chaabane rchaabane@ai.univ-paris8.fr

Les chaînes de caractères

Comme déjà précisé dans les premiers chapitres de ce cours, il n'existe pas de type chaîne de caractères prédéfinit en C. Il existe deux façons pour déclarer une chaîne de caractères en C, soit en utilisant un tableau de char dont la taille est fixée en avance, soit en utilisant un pointeur sur des char. Dans ce dernier cas, la taille de la chaîne ne peut être connue d'avance.

1. Règles générales d'écriture de constantes chaînes

Une chaîne de caractères s'écrit entre guillemets comme suit :

```
"Hello"
```

Cette chaîne contient 5 caractères.

Les caractères attendus entre les guillemets sont des lettres `a-Z`, des chiffres `1-9`, des caractères de ponctuation, mais aussi des caractères tels que `\n, \t, \a, \",\\.`

Le caractère `"` dans la chaîne indique la fin de la chaîne de caractères pour forcer l'apparition des guillemets dans la chaîne il faut les précéder du caractère `\`. Exemple :
Pour afficher la chaîne de caractères suivante :

Mr. Dupond a dit :
" J'ai été ravi de vous voir."

Il faut définir la chaîne de caractères suivante :

```
"Mr. Dupond a dit:\n\"J'ai été ravi de vous voir\"."
```

Il faut faire attention à ne pas placer n'importe comment les guillemets dans votre chaîne de caractères. Exemple :

```
"Il a dit : "Bonjour ! "."  /* incorrect car 'Bonjour !' n'est pas dans la
                               chaîne de caractères */
```

Première chaîne Deuxième chaîne

1.1. Déclarer une chaîne de caractère

- **Comme un tableau de char :**

Rim Chaabane rchaabane@ai.univ-paris8.fr

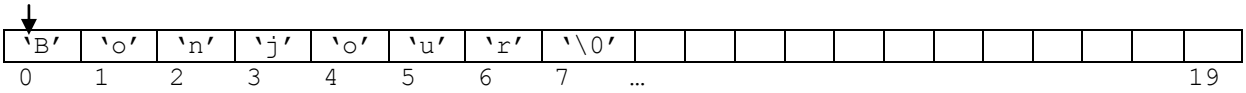
Pour déclarer une chaîne de caractère nous pouvons l'enregistrer sous forme d'un tableau de caractères. Pour délimiter la fin de la chaîne dans le tableau, le caractère fin de chaîne sera placé par défaut après le dernier caractère de la chaîne. Exemple :

```
#include <stdio.h>

int main(){
   int i;
   char tab[20] = "Bonjour";

   for (i=0; tab[i]!='\0'; i++){
      printf("%c", tab[i]);
   }
   printf("\n");
   return 0;
}
```

Pour afficher la chaîne initialisée dans tab, il faut la parcourir caractère par caractère jusqu'à rencontrer le caractère de fin de chaîne '\0'.

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

tab

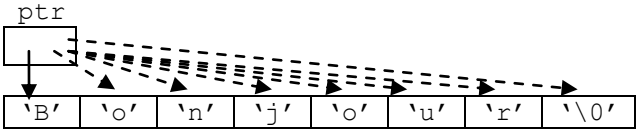



```
tab[0] = 'B'
tab[1]= 'o'
tab[2]= 'n'
…
tab[7]='\0'
```

- **comme un pointeur sur char :**

```
char * ptr = "Bonjour";
```

ptr




```
*ptr = 'B'
*(ptr+1) = 'o'
*(ptr+2) = 'n'
etc…
```

L'emplacement d'une chaîne est alloué une fois pour toutes avant l'exécution et il subsiste jusqu'à la fin de l'exécution.

Il vous est donc possible de manipuler au choix les chaines de caractères comme un tableau de char ou comme un pointeur sur char.

2. *Fonctions d'entrées/sorties liées aux chaînes*

- Ecriture de chaînes avec puts

Prototype : int puts (const char * s)

Cette fonction appartient à la librairie <stdio.h>
Elle renvoie les différents caractères de la chaîne sur l'écran aussi appelé unité de sortie standard stdout. Ensuite, elle transmet un caractère de fin de ligne ('\n').

Exemple d'utilisation :

```
char ch1[] = "Bonjour";
char ch2[] = "monsieur";
printf("essai de puts :");
puts(ch1);
puts(ch2);
```

On obtient sur la sortie standard :

```
essai de puts :Bonjour
monsieur
```

Si la chaîne ch1 avait contenu l'information ″Bonjour\n″ au lieu de ″Bonjour″, on obtiendrait une ligne vide supplémentaire :

```
essai de puts :Bonjour

monsieur
```

Cette fonction retourne une valeur entière positive lorsque l'opération s'est bien déroulée, et si une erreur survient la valeur EOF est retournée.

- Ecriture de chaînes avec %s dans printf

Puts et printf avec un %s permettent d'afficher une chaîne de caractère à la sortie standard. Contrairement à la fonction puts, printf avec un %s affiche la chaîne de caractères sans caractère de fin de chaîne et de retour à la ligne.

Les deux instructions suivantes sont équivalentes :

```
puts(ch1);
printf("%s\n", ch1);
```

- Lecture de chaînes avec gets

Prototype : char * gets(char * s)

Cette fonction permet de lire une chaîne de caractère depuis l'entrée standard stdin. La lecture s'arrête à la lecture d'un caractère de retour à la ligne. Il ne sera, cependant, pas enregistré en mémoire à la fin de chaine de caractères lue.

Par contre le caractère de fin de chaîne est enregistré en mémoire.
Le comportement de gets diffère de scanf avec un %s sur les points suivants :

- Pour gets, seule la fin de ligne sert de délimiteur alors que pour %s, il existe 5 caractères délimiteurs qu'on nomme des espaces blancs ; la chaine lue par gets peut donc contenir n'importe quel caractère espace blanc (excepté \n), en particulier des espaces.
- Ce caractère fin de ligne est consommé par la lecture, c'est-à-dire que le pointeur ne reste pas placé dessus comme il le ferait avec le code %s dans scanf. Cela signifie qu'une prochaine lecture, en particulier par gets, accédera bien au caractère de la ligne suivante.

La fonction gets fournit comme valeur de retour l'adresse de la chaine lue lorsque l'opération s'est bien déroulée, le pointeur NULL en cas d'erreur.

3. Fonctions de manipulation de chaîne

Ces fonctions travaillent toujours sur des adresses, ce qui signifie que l'on ne peut jamais transmettre à une fonction la valeur d'une chaine, mais seulement son adresse. On transmet ainsi un pointeur sur son premier caractère.

Les adresses son toujours de type char *.
Les arguments dans les prototypes des fonctions de manipulations de chaines de caractères sont de type char * ou const char *. Pour justifier cela, soit les deux fonctions fc et f :
fc (const char * adc);
f (char * ad) ;

Voyons ce que cela implique dans la définition de ces fonctions d'une part, et dans leur utilisation d'autre part.

- **Dans la définition des fonctions**

Il est impossible de modifier, dans le corps de la fonction fc, l'objet pointé par adc par des affectations de la forme :

adc = … ; / interdit car adc pointe sur un objet constant */

Qui plus est, il est impossible d'utiliser la valeur d'adc pour aller modifier des objets voisins, par des affectations de cette forme :

(adc+1) = … ; / interdit car l'expression adc +1 est, comme adc, de type pointeur sur un
adc[i]= … ; objet constant. Il en va de même pour adc+i équivalente à adc[i]. */

Ainsi la fonction fc est censée ne pas modifier la chaine pointée par adc, alors que la fonction f peut changer la chaine de caractère pointée par ad.

- **Dans l'utilisation des fonctions**

La fonction fc peut être appelée avec un argument de type const char* ou char *. Alors que pour la fonction f, elle ne peut être appelée qu'avec un argument de type char *.

Exemples :

```
char * ch; /* interdit car adc pointe sur un objet constant */

const char * chc= "salut";

fc(chc); /* correct : chc est du type attendu */
fc (ch); /* correct : ch de type char * est converti en type const char *  */

f(ch); /* correct : ch est du type attendu, char * */
f(chc); /* incorrect : chc es est de type const char * il ne peut être converti en char * */
```

Il n'est pas possible de traiter un objet constant comme un objet non constant puisqu'on risque de le modifier. Alors qu'un objet non constant peut être traité comme un objet constant car il n'y a alors aucun risque à ne pas le modifier.

Ainsi, déclarer votre chaine de caractères comme const char *, vous garantit que votre chaine ne sera modifiée par aucune fonction de la bibliothèque standard.

Toutes les fonctions qui suivent appartiennent à la librairie <string.h>.

a. La fonction strlen

La fonction strlen fournit la longueur de la chaine dont on lui a transmis l'adresse en argument. Cette longueur correspond au nombre de caractères trouvés depuis l'adresse indiquée jusqu'au premier caractère de code nul. Ce dernier n'étant pas pris en compte.
Par exemple :

strlen(″bonjour″) ; /* retournera 7 */

De même avec :

char * adr = ″salut″ ;
strlen(adr) ; /* retournera 5 */

- **prototype de strlen :**

size_t strlen(const char * chaine)	
chaine	Adresse de la chaine
Valeur de retour	Taille de la chaine

Remarque : le type ,size_t, de la valeur de retour de strlen() correspond à un type entier non signé.

- **Exemple d'implémentation de la fonction strlen :**

Soit la fonction longueur qui retourne tout comme strlen la longueur d'une chaine de caractères.

```
size_t longueur(const char * ad){
   size_t lgr = 0;
   while (*ad++) lgr++;
   return lgr;
}
```

Attention à l'utilisation de la valeur de retour de strlen dans un printf :

printf(″%u″, strlen(ch)) ; /* affichage erroné car size_t ne sera pas interprété comme un unsigned int */

Il est donc préférable de procéder ainsi :

unsigned long taille ;
taille = strlen(ch) ; /* conversion du type size_t en type unsigned long */
printf(″%u″, taille) ;

b. Fonctions de copie

➢ La fonction strcpy :

Cette fonction recopie à l'adresse d'une première chaine, l'adresse d'une seconde chaine de caractères avec son caractère de fin de chaine ('\0').

- **Prototype :**

char * strcpy(char* but, const char * source)	
but	Adresse à laquelle sera recopiée la chaine source
source	Adresse de la chaine à recopier
valeur de retour	Adresse *but*

Exemples d'utilisation :

```
strcpy (ch1, "") ; /* la chaine se trouvant à l'adresse ch1 est maintenant une chaine
                      vide */
```

- **les risques de la fonction strcpy :**

- aucun contrôle de longueur n'est effectué par cette fonction, il est donc nécessaire que l'emplacement réservé à *but* soit suffisant pour y reçevoir la chaine située à l'adresse source.
  Exemple :

```
char ch[5] ;
…
strcpy(ch, "bonjour") ; /* le caractère 'r' va écraser la valeur de l'emplacement
                           mémoire ch[5] */
```

  Pour recopier seulement les 5 premiers caractères de "bonjour" dans ch ; il faut utiliser la fonction strncpy qui permet de préciser le nombre de caractères à recopier.

- Elle ne s'assure pas de la présence du caractère de fin de chaine avant la copie.
  Exemple :

```
char ch1[10] ;
char ch2[7] ="bonjour"; /* pas de caractère de fin de chaine ici */
…
strcpy(ch1, ch2) ; /* strcpy va copier les caractères du mot bonjour puis va recopier
                      tous les caractères se trouvant après l'adresse ch2+7 jusqu'à ce
                      qu'elle rencontre un caractère de fin de chaine. Ce qui risque de
                      ne jamais arriver, de plus la chaine ch1 est limitée à 10
                      caractères, un débordement est donc à prévoir. */
```

  L'utilisation de la fonction strncpy() permettrait de limiter le nombre de caractères à recopier ce qui évitera tout débordement.

➢ La fonction strncpy :

Cette fonction recopie aussi à l'adresse d'une première chaine, l'adresse d'une seconde chaine de caractères mais en limitant la copie à un nombre de caractères précis. Ce nombre est donné en argument. Contrairement à strcpy, le caractère de fin de chaine ('\0') n'est pas rajouté automatiquement dans tous les cas.

- **Prototype :**

char * strncpy(char* but, const char * source, size_t longueur)	
But	Adresse à laquelle sera recopiée la chaine source
source	Adresse de la chaine à recopier
longueur	Nombre maximal de caractères recopies, y compris l'éventuel caractère '\0'
valeur de retour	Adresse *but*

Exemples d'utilisation :

char ch [10] ;
strcpy (ch, "abc", 10) ; /* ch contiendra a, b et c mais aussi 7 caractères nuls. */
strcpy (ch, "abc", 4) ; /* ch contiendra a, b, c et un caractère nul. */

strcpy (ch, "abc", 3) ; /* ch contiendra seulement a, b et c. */

strcpy (ch, "abc", 2) ; /* ch contiendra seulement a et b. */

- **Comment faire des copies fiables :**

L'inconvénient de stncpy est de ne pas toujours mettre le caractère de fin de chaine dans la chaine retournée en résultat (*but*). Pour éviter cela voici un exemple d'une façon de procéder pour remédier à ce défaut :

```
#include <stdio.h>
#define N 5

char ch1 [N];
...

strncpy(ch1, ch, N-1); /* on recopie au maximum N-1 caractères dans ch1 */
if (strlen(ch)>= (N-1)){ /* si la copie a été interrompue on ajoute     */
   ch1[N-1]='\0';        /* le caractère de fin de chaine ici           */
}
```

c. *Fonctions de concaténation*

Les fonctions de concaténation permettent de mettre bout à bout deux chaines de caractères afin de n'en former qu'une seule. Il existe en C deux fonctions pour réaliser ce travail :

- strcat
- strncat

Ces deux fonctions sont très similaires, la seule différence réside dans le fait que strncat possède un troisième argument qui permet de limiter le nombre de caractères à concaténer (identiquement à strncpy).

➢ La fonction strcat :

Cette fonction recopie à la fin d'une première chaine une seconde chaine.

- **Prototype :**

char * strcat(char * but, char * source)	
but	Adresse de la chaine réceptrice
source	Adresse de la chaine à concaténer
Valeur de retour	Adresse but

Cette fonction recopie la chaine située à l'adresse *source* à la fin de la chaine d'adresse *but*, c'est-à-dire à partir de son zéro de fin ; ce dernier se trouve donc remplacé par le premier caractère de la chaine d'adresse source.

Exemple :

```
char * ch1, * ch2 = "";
...

strcat(ch1, "");    /* la chaine à l'adresse ch1 est inchangée */
strcat(ch1, ch2);   /* la chaine à l'adresse ch1 est inchangée */
```

- **les risques de la fonction strcat :**

- aucun contrôle de longueur n'est effectué par cette fonction, il est donc nécessaire que l'emplacement réservé à la première chaine soit suffisant pour y recevoir la chaine à lui concaténer.
  Exemple :

```
char ch1[10] = "bonjour";
const char * ch2 = " monsieur";

...

strcat(ch1, ch2); /* débordement du tableau de caractères ch1 */
strcat(ch2, ch1); /*rejeté par le compilateur (erreur de compilation)*/
```

L'erreur de compilation est due au fait qu'à l'appel de strcat, ch2 sera convertie en char * au lieu de const char * ce qui est interdit.

- aucun contrôle sur la présence ou non du caractère de fin de chaine. Ainsi, si une des chaine ne dispose pas de caractère de fin de chaine, la recherche de caractères se poursuivra au-delà de la fin de zone concernée.
  Exemple :

```
char ch1[7] = "bonjour"; /* aucun caractère de fin de chaine dans ch1 */
const char * ch2 = " monsieur";

...

strcat(ch1, ch2);    /* recherche indéterminée dans ch1 du caractère */
                     /* de fin de chaine                             */
```

➢ La fonction strncat :

Cette fonction recopie à la fin d'une première chaine une seconde chaine en limitant cette dernière à un nombre de caractères donnés en argument.

- **Prototype :**

char * strncat(char * but, char * source, size_t longueur)	
but	Adresse de la chaine réceptrice
source	Adresse de la chaine à concaténer
longueur	Nombre maximal de caractères concaténés
Valeur de retour	Adresse but

Cette fonction recopie la chaine située à l'adresse *source* à la fin de la chaine d'adresse *but*, c'est-à-dire à partir de son zéro de fin ; ce dernier se trouve donc remplacé par le premier caractère de la chaine d'adresse source. Le nombre de caractères recopiés est limité à *longueur*.

A la différence de strncpy, strncat place un caractère de fin de chaine même si aucun caractère de fin de chaine n'a été trouvé parmi les caractères à concaténer.

Exemple :

```
char * ch1 = "salut";
char * ch2 = " Jack";
...

strncat(ch1, "", 10);      /* la chaine à l'adresse ch1 est inchangée */
strncat(ch1, ch2, 10);     /* la chaine à l'adresse ch1 est inchangée */
strncat(ch1, " Ben", 0);   /* la chaine à l'adresse ch1 est inchangée */
```

- **les risques de la fonction strncat :**

Le risque est le même que pour strcat à savoir qu'elle ne s'assure pas de la présence du caractère de fin de chaine

- **Comment faire une concaténation sans risques :**

Exemple de code :

```
#define N 50


char ch1 [N+1];
char * ch2 = " n'importe quoi ici";
...

strncat(ch1, ch2, (N - strlen(ch1)) );    /* concaténation de ch2 à ch1 */
                                          /* sans  dépasser la taille   */
                                          /* maximale de ch1            */
```

L'expression `(N - strlen(ch1))` ne doit jamais être négative.

Exemple d'utilisation de strncat :

```
#include <stdio.h>
#include <string.h>
#define CMAX 50
#define LMAX 80

int main(){
        char ch [CMAX] = "";
        char ligne [LMAX];

        printf("donnez une suite de chaines, a raison d\'une par ligne: \n");
        do{
                gets(ligne);
                strncat(ch, ligne, (CMAX - strlen(ch)) );

        } while (strlen(ch)< CMAX);

        printf("voici vos chaines concaténées : \n%s\n", ch);
        return 0;
}
```

d. Fonctions de comparaison

Pour comparer des chaines de caractères, nous devons utiliser des fonctions spécifiques car contrairement aux données de type de base les opérateurs relationnels classiques ne peuvent être employés.

Les fonctions que nous décrivons ci-dessous permettent de comparer des chaines de caractères selon leur ordre alphabétique (ordre de la table ASCII). Ces fonctions sont les suivantes :

- strcmp
- strncmp

➢ La fonction strcmp :

Cette fonction compare la première chaine à la seconde et retourne une valeur entière en retour qui indique l'ordre des deux chaines.

- **Prototype :**

int strcmp (const char * chaine1, const char * chaine2)	
chaine1	Adresse de la première chaine
chaine2	Adresse de la deuxième chaine
Valeur de retour	- < 0 : si la chaine d'adresse chaine1 arrive avant la chaine2 - > 0 : si la chaine d'adresse chaine1 arrive après la chaine2 - = 0 : si les deux chaines sont identiques

Exemple :

```
strcmp("bonjour", "monsieur"); /* négatif                          */
strcmp("paris2", "paris10");   /* positif, 1 vient avant le 2       */
strcmp("bonjour", "bonjour");  /* nul                               */
strcmp("Paris", "paris");      /* positif, les minuscules se placent */
                               /* avant les majuscules              */
strcmp("PARIS", "paris");      /* idem                              */
strcmp("ré", "rat");           /* positif                           */
strcmp("ré", "rue");           /* positif                           */
```

➢ La fonction strcnmp :

Même comportement que strncmp mais en limitant la comparaison des deux chaines à un nombre maximal de caractères.

- **Prototype :**

int strncmp (const char * chaine1, const char * chaine2, size_t longueur)	
chaine1	Adresse de la première chaine
chaine2	Adresse de la deuxième chaine
Longueur	Nombre maximal de caractères soumis à la comparaison
Valeur de retour	- < 0 : si la chaine d'adresse chaine1 arrive avant la chaine2 - > 0 : si la chaine d'adresse chaine1 arrive après la chaine2 - = 0 : si les deux chaines sont identiques

Exemples :

```
strcmp("bonjour", "bon", 12);  /* positif "bon" avant "bonjour"  */
strcmp("bonjour", "bon", 4);   /* idem                           */
strcmp("bonjour", "bon", 2);   /* idem                           */
```

e. Fonctions de recherche

La bibliothèque standard propose aussi des fonctions de recherche de la première occurrence d'un caractère ou sous-chaine dans une chaine de caractères et qui sont :

- Recherche d'un caractère : strchr et strrchr
- Recherche d'une sous-chaine : strstr
- Recherche d'un des caractères appartenant à un ensemble de caractères : strpbrk

Toutes ces fonctions retournent en résultat l'adresse de l'information cherchée si elle est trouvée, le pointeur nul autrement.

Ou encore des fonctions qui permettent d'éclater une chaine en plusieurs parties.
Exemple : strtok.

➢ La fonction strchr :

Recherche la première occurrence d'un caractère dans une chaine de caractères.

- **Prototype :**

char * strchr (const char * chaine, int c)	
chaine	Adresse de la première chaine
c	Caractère recherché après conversion en unsigned char
Valeur de retour	Adresse du premier caractère c trouvé s'il existe, pointeur NULL sinon

Exemple :
```
strchr("bonjour", 'o');  /* adresse du premier 'o' de la chaine "bonjour" */
strchr("bonjour", 'a');  /* fournit la valeur NULL                         */
```

➢ La fonction strrchr :

Même comportement que strrchr mais en effectuant la recherche à partir de la fin de la chaine et non à partir du début de la chaine comme strchr.

- **Prototype :**

char * strchr (const char * chaine, int c)	
chaine	Adresse de la première chaine
c	Caractère recherché après conversion en unsigned char
Valeur de retour	Adresse du premier caractère c trouvé s'il existe, pointeur NULL sinon

➢ La fonction strstr :

Recherche la première occurrence d'une sous chaine dans une chaine de caractères.

- **Prototype :**

char * strstr (const char * chaine1, const char * chaine2)	
chaine1	Adresse de la première chaine
chaine2	Adresse de la sous-chaine recherchée
Valeur de retour	Adresse de la première occurrence complète de la sous-chaine cherchée, si elle existe, sinon le pointeur NULL

Exemple :

```
strstr("recherche", 'ch');  /* retourne l'adresse du 3ème caractère de la chaine */
strstr("recherche", 'cha');  /* retourne la valeur NULL                            */
```

➢ La fonction strtok :

Eclate une chaine en plusieurs sous-chaines, sachant que ces dernières sont séparées par un ou plusieurs délimiteurs.

- **Prototype :**

char * strtok (const char * chaine, const char * delimiteurs)	
chaine	Adresse de la première chaine
delimiteurs	Adresse d'une chaine contenant les caractères délimiteurs
Valeur de retour	Adresse de la première sous-chaine de la chaine délimitée (avant et après) par des caractères délimiteurs si elle existe, le pointeur NULL sinon

Cette fonction recherche donc à partir de l'adresse chaine, le premier caractère différent des caractères appartenant à la chaine délimiteurs. Si aucun caractère n'est trouvé la fonction retourne NULL.

Si un caractère est trouvé, la fonction recherche alors, à partir de là, le premier caractère qui corresponde à l'un des délimiteurs. Si un délimiteur est trouvé, il est remplacé par un caractère de fin de chaine. La fonction retourne ainsi la première sous-chaine trouvée. Pour retrouver les autres sous-chaines il faut rappeler la fonction strtok mais avec la valeur NULL en premier argument de strtok. Le second argument correspond toujours aux caractères délimiteurs mais ils peuvent différer d'un appel au suivant. Avec un tel appel, la fonction strtok reprend son exploration non plus au début de la chaine, mais à la suite du caractère de fin de chaine qui vient d'être placé par le précédent appel.

Nous donnons dans ce qui suit un exemple d'utilisation de cette fonction :

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

int main(){
        char ch1[] = "12:30:25";
        char ch2[] = "12h30m42s";
        char ch3[] = "12hr 30mn 42s";
        char ch4[] = "une//autre?chaine$$un$peu?$bizarre";
        char ch5[] = "une//autre?chaine$$un$peu?$bizarre";
        char * adr;
        printf("eclatement de la chaine \"%s\" avec le seul delimiteur \":\"\n", ch1);
        adr = strtok(ch1, ":"); /* localisation du premier delimiteur */
        while (adr){
                printf("%s\n", adr);/* localisation du delimiteur suivant */
                adr = strtok(NULL, ":");
        }

        printf("eclatement de la chaine \"%s\" avec les delimiteurs \"hms\"", ch2);
        adr = strtok(ch2, "hms"); /* localisation du premier delimiteur */
        while (adr){
                printf("%s\n",adr); /* localisation du delimiteur suivant */
                adr = strtok(NULL, "hms");
        }

        printf("eclatement de la chaine \"%s\"avec les delimiteurs \"hr\" \"mn\" \"s\"", ch3);
        adr = strtok(ch3, "hrms"); /* localisation du premier delimiteur */
        while (adr){
                printf("%s\n",adr);
                adr = strtok(NULL, "hrmns"); /* localisation du delimiteur suivant */
        }

        printf("eclatement de la chaine \"%s\" \nen se servant de delimiteurs variables\n", ch4);

        adr = strtok(ch4, "/");
        printf("%s\n", adr);
        adr = strtok(NULL, "$");
        printf("%s\n", adr);
        adr = strtok(NULL, "?");
        printf("%s\n", adr);
        adr = strtok(NULL, "/");
        printf("%s\n", adr);

        printf("eclatement de la chaine \"%s\" \nen se servant de delimiteurs variables\n", ch5);
        adr = strtok(ch5, "/$?"); /* localisation du premier delimiteur */
        while (adr){
                printf("%s\n",adr);
                adr = strtok(NULL, "/$?"); /* localisation du delimiteur suivant */
        }

        return 0;
}
```

Trace d'exécution :

```
eclatement de la chaine "12:30:25" avec le seul delimiteur ":"
12
30
25
eclatement de la chaine "12h30m42s" avec les delimiteurs "hms"12
30
42
eclatement de la chaine "12hr 30mn 42s"avec les delimiteurs "hr" "mn" "s"12
 30
 42
```

```
eclatement de la chaine "une//autre?chaine$$un$peu?$bizarre"
en se servant de delimiteurs variables
une
/autre?chaine
$un$peu
$bizarre
eclatement de la chaine "une//autre?chaine$$un$peu?$bizarre"
en se servant de delimiteurs variables
une
autre
chaine
un
peu
bizarre
```

f. *Fonctions de conversion de chaîne en nombre*

Ces fonctions permettent de convertir une chaine (exemple : "21.6Degrés Celsius") ou une partie de la chaine en un nombre. Exemple :
- de type long (le résultat retourné est 21 dans le cas de notre exemple).
- de type double (le résultat retourné est 21,6 dans le cas de notre exemple).

Dans les deux cas les caractères qui ne peuvent pas être convertis sont ignorés.

Les fonctions de conversion sont :
Bibliothèque <string.h>
- strtol → chaine en long
- strtoul → chaine en unsigned long
- strtod → chaine en double

Bibliothèque <strlib.h>
- atoi → chaine en entier
- atol → chaine en long
- atof → chaine en flottant

Prototypes :

double atof(const char * chaine)	
chaine	Chaine à convertir
Valeur de retour	Résultat de la conversion de la chaine en double

long atol(const char * chaine)	
chaine	Chaine à convertir
Valeur de retour	Résultat de la conversion de la chaine en long

int atoi(const char * chaine)	
chaine	Chaine à convertir
Valeur de retour	Résultat de la conversion de la chaine en int