**Manipulez du texte avec les chaînes de caractères**

En programmation, une chaîne de caractères désigne… du texte, tout simplement ! Et on peut retenir ce texte sous forme de variable en mémoire.

Comme nous l'avons dit plus tôt, notre ordinateur ne peut retenir que des nombres. Les lettres sont exclues. Comment diable les programmeurs font-ils pour manipuler du texte, alors ? Eh bien ils sont malins, vous allez voir !

**Utilisez le type char pour stocker une lettre**

Si le type char permet de stocker des nombres (compris entre -128 et 127), on l'utilise rarement pour ça en C.

En général, même si le nombre est petit, on le stocke dans un int  . Certes, ça prend un peu plus de place en mémoire, mais aujourd'hui, la mémoire, ce n'est vraiment pas ce qui manque sur un ordinateur.

Le type char est en fait prévu pour stocker une lettre ! Attention, j'ai bien dit **UNE** lettre.

Comme la mémoire ne peut stocker que des nombres, on a inventé une table qui fait la conversion entre les nombres et les lettres. Ainsi, cette table indique par exemple que le nombre 65 équivaut à la lettre A.

Le langage C permet de faire très facilement la traduction lettre <=> nombre correspondant. Pour obtenir le nombre associé à une lettre, il suffit d'écrire cette lettre entre apostrophes, comme ceci : 'A'  . À la compilation, 'A' sera remplacé par la valeur correspondante.

Testons :

int main(int argc, char \*argv[])

{

char lettre = 'A';

printf("%d\n", lettre);

return 0;

}

65

On sait donc que la lettre A majuscule est représentée par le nombre 65. B vaut 66, C vaut 67, etc.

Testez avec des minuscules, et vous verrez que les valeurs sont différentes. 'a' n'est pas identique à 'A'  : l'ordinateur fait la différence entre les majuscules et les minuscules (on dit qu'il "respecte la casse").

La plupart des caractères « de base » sont codés entre les nombres 0 et 127. Une table fait la conversion entre les nombres et les lettres : la table ASCII (prononcez « Aski »). Le site [AsciiTable.com](http://www.asciitable.com" \t "_blank) propose cette table mais ce n'est pas le seul, on peut aussi la retrouver sur Wikipédia et bien d'autres sites encore.

**Affichez un caractère avec la fonction printf**

La fonction printf  peut aussi afficher un caractère via le symbole %c(c comme *caractère*) :

int main(int argc, char \*argv[])

{

char lettre = 'A';

printf("%c\n", lettre);

return 0;

}

A

Hourra, nous savons afficher une lettre !

On peut aussi demander à l'utilisateur d'entrer une lettre en utilisant le %c dans un scanf  :

int main(int argc, char \*argv[])

{

char lettre = 0;

scanf("%c", &lettre);

printf("%c\n", lettre);

return 0;

}

Si je tape la lettre B, je verrai :

B

B

Le premier des deux B est celui que j'ai tapé au clavier, le second est celui affiché par le printf.

Voici à peu près tout ce qu'il faut savoir sur le type char :

* le type char permet de stocker des nombres allant de -128 à 127, (le type unsigned char permet de stocker les nombres de 0 à 255) ;
* votre ordinateur utilise la table ASCII pour convertir les lettres en nombres et inversement ;
* on peut alors utiliser le type char pour stocker UNE lettre ;
* 'A'  est remplacé à la compilation par la valeur correspondante (65, en l'occurrence). On utilise donc les apostrophes pour obtenir la valeur d'une lettre.

**Voyez les chaînes comme des tableaux de type  char**

Une chaîne de caractères n'est en fait rien d'autre qu'un tableau de type char.

Si on crée un tableau :

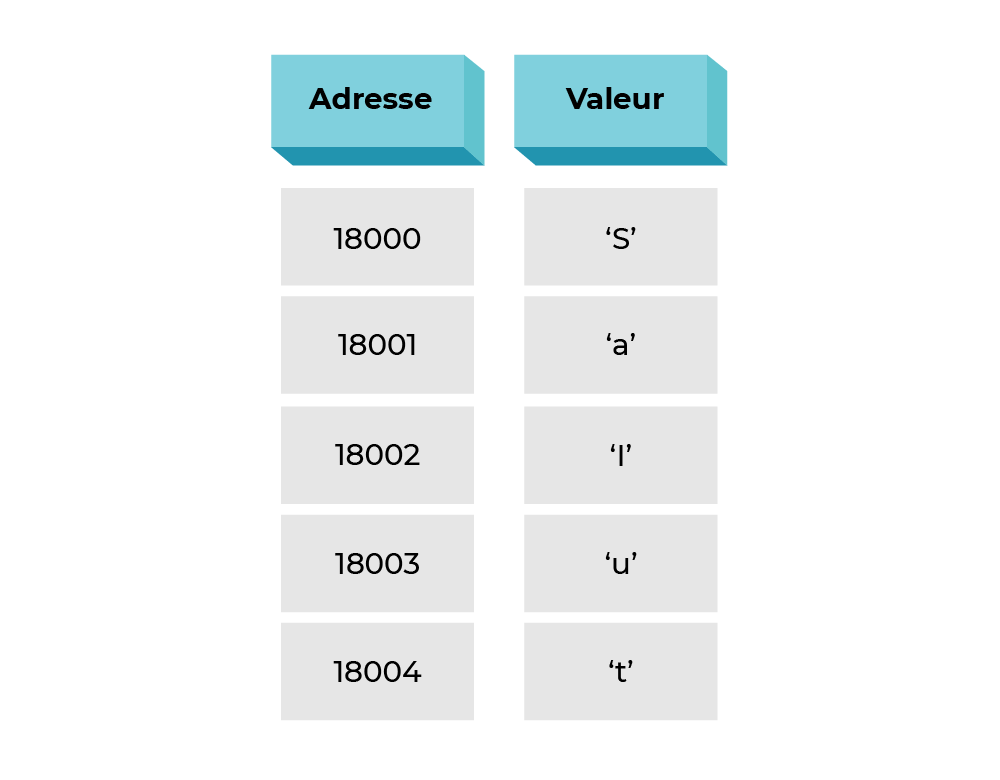
char chaine[5];

et qu'on met :

* dans chaine[0] la lettre 'S'  ;
* dans chaine[1] la lettre 'a'  ,

…  on peut ainsi former une chaîne de caractères, c'est-à-dire du texte.

Voici, de manière simplifiée, comment la chaîne est stockée en mémoire :

Une chaîne de caractères en mémoire (simplifiée)

Ce tableau prend 5 cases en mémoire pour représenter le mot "Salut".

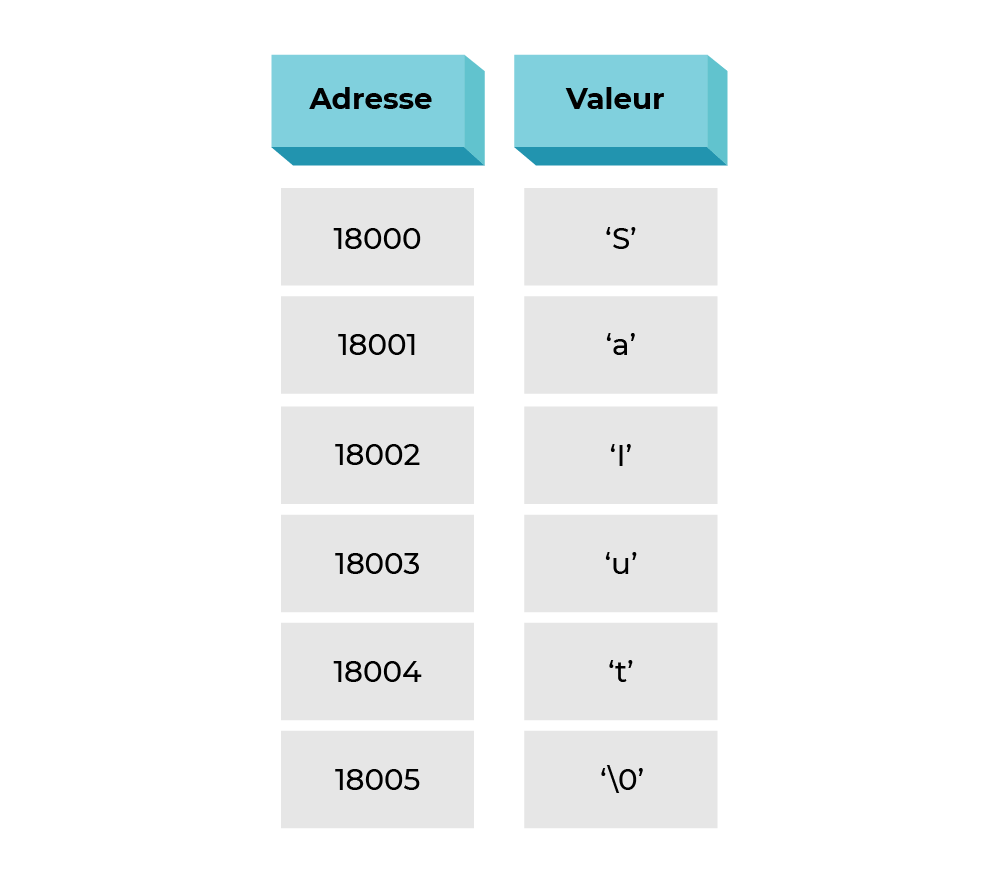
Pour la valeur, j'ai volontairement écrit sur le schéma les lettres entre apostrophes pour indiquer que c'est un nombre qui est stocké, et non une lettre. En réalité, dans la mémoire, ce sont bel et bien les nombres correspondant à ces lettres qui sont stockés.

Toutefois, une chaîne de caractères ne contient pas que des lettres ! Le schéma est en fait incomplet. Une chaîne de caractère **doit impérativement contenir un caractère spécial à la fin de la chaîne**, appelé "caractère de fin de chaîne". Ce caractère s'écrit'\0'  .

Pourquoi faut-il que la chaîne de caractères se termine par un \0  ?

Tout simplement pour que votre ordinateur sache quand s'arrête la chaîne ! Le caractère\0permet de dire : "Stop, c'est fini, y'a plus rien à lire après, circulez !".

Par conséquent, pour stocker le mot "Salut" (qui comprend 5 lettres) en mémoire, il ne faut pas un tableau de 5 char, mais de 6 :

Une chaîne de caractères en mémoire

Chaque fois que vous créez une chaîne de caractères, vous allez donc devoir penser à prévoir de la place pour le caractère de fin de chaîne. Il faut **toujours** ajouter un bloc de plus dans le tableau pour stocker ce caractère \0  , c'est impératif !

Oublier le caractère de fin \0  est une source d'erreurs impitoyable du langage C. Je le sais pour en avoir fait les frais plus d'une fois.

Grâce à \0  :

* vous n'avez pas à retenir la taille de votre tableau car il indique que le tableau s'arrête à cet endroit ;
* vous pouvez passer votre tableau de char à une fonction sans avoir à ajouter à côté une variable indiquant la taille du tableau.

Cela n'est valable que pour les chaînes de caractères (c'est-à-dire le type char\*  , qu'on peut aussi écrire char[]  ). Pour les autres types de tableaux, vous êtes toujours obligé de retenir la taille du tableau quelque part.

**Créez et initialisez une chaîne**

Si on veut initialiser notre tableau chaine avec le texte "Salut", on peut utiliser la méthode manuelle mais peu efficace :

char chaine[6]; // Tableau de 6 char pour stocker S-a-l-u-t + le \0

chaine[0] = 'S';

chaine[1] = 'a';

chaine[2] = 'l';

chaine[3] = 'u';

chaine[4] = 't';

chaine[5] = '\0';

Cette méthode marche.

On peut le vérifier avec un printf en utilisant le symbole %s   ( s pour *string*en anglais, qu'on traduit par "chaîne").

Voici le code complet qui crée une chaîne "Salut" en mémoire, et qui l'affiche :

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[])

{

char chaine[6]; // Tableau de 6 char pour stocker S-a-l-u-t + le \0

// Initialisation de la chaîne (on écrit les caractères un à un en mémoire)

chaine[0] = 'S';

chaine[1] = 'a';

chaine[2] = 'l';

chaine[3] = 'u';

chaine[4] = 't';

chaine[5] = '\0';

// Affichage de la chaîne grâce au %s du printf

printf("%s", chaine);

return 0;

}

Résultat :

Salut

Vous remarquerez que c'est un peu fatigant et répétitif de devoir écrire les caractères un à un comme on l'a fait dans le tableau chaine  .

Pour initialiser une chaîne, il existe heureusement une méthode plus simple :

int main(int argc, char \*argv[])

{

char chaine[] = "Salut"; // La taille du tableau chaine est automatiquement calculée

printf("%s", chaine);

return 0;

}

Salut

À la première ligne, je crée une variable de type char[]  . J'aurais pu écrire aussi char\*  , le résultat aurait été le même.

En tapant entre guillemets la chaîne à mettre dans votre tableau, le compilateur C calcule automatiquement la taille nécessaire. C'est-à-dire qu'il compte les lettres et ajoute 1 pour placer le caractère \0  .

Il écrit ensuite, une à une, les lettres du mot "Salut" en mémoire, et ajoute le \0  comme on l'a fait nous-mêmes manuellement quelques instants plus tôt. Bref, c'est bien plus pratique.

Il y a toutefois un défaut : ça ne marche que pour l'initialisation ! Vous ne pouvez pas écrire plus loin dans le code :

chaine = "Salut";

Cette technique est donc à réserver à l'initialisation. Après cela, il faudra écrire les caractères manuellement un à un en mémoire, comme on l'a fait au début.

**Récupérez une chaîne via un scanf**

Vous pouvez enregistrer une chaîne entrée par l'utilisateur via un scanf  , via le symbole %s  .

Seul problème : vous ne savez pas combien de caractères l'utilisateur va entrer. Si vous lui demandez son prénom, il s'appelle peut-être Luc (3 caractères), mais qui vous dit qu'il ne s'appelle pas Jean-Édouard (beaucoup plus de caractères) ?

Pour ça, il n'y a pas 36 solutions. Il va falloir créer un tableau de char suffisamment grand pour pouvoir stocker le prénom. On va donc créer un char[100] :

int main(int argc, char \*argv[])

{

char prenom[100];

printf("Comment t'appelles-tu ? ");

scanf("%s", prenom);

printf("Salut %s, je suis heureux de te rencontrer !", prenom);

return 0;

}

Comment t'appelles-tu ? Mateo21

Salut Mateo21, je suis heureux de te rencontrer !

**Manipulez des chaînes à l'aide de fonctions**

La bibliothèque string.h fournit de nombreuses fonctions dédiées aux calculs sur des chaînes.

Je vais vous présenter les principales :

1. strlen pour calculer la longueur d'une chaîne.
2. strcpy pour copier une chaîne dans une autre.
3. strcat pour concaténer 2 chaînes.
4. strcmp pour comparer 2 chaînes.
5. strchr pour rechercher un caractère.
6. strpbrk pour rechercher le premier caractère de la liste.
7. strstr pour rechercher une chaîne dans une autre.
8. sprintf pour écrire dans une chaîne.

Comme on va utiliser une nouvelle bibliothèque appelée string.h  , vous devez l'inclure !

Rajoutez cette ligne en haut des fichiers .c où vous en avez besoin :

#include <string.h>

Si vous ne le faites pas, l'ordinateur ne connaîtra pas les fonctions que je vais vous présenter car il n'aura pas les prototypes, et la compilation plantera.

**1. Calculez la longueur d'une chaîne avec  strlen**

strlen (pour "string length") est une fonction qui calcule la longueur d'une chaîne de caractères (sans compter le caractère\0  ) :

1. Vous lui envoyez un seul paramètre (votre chaîne de caractères) ;
2. Elle vous retourne la longueur de la chaîne.

Maintenant que vous savez ce qu'est un prototype, je vous donne celui de la fonction  strlen  :

size\_t strlen(const char\* chaine);

size\_t est un type spécial qui signifie que la fonction renvoie un nombre correspondant à une taille.

Ce n'est pas un type de base comme int  , long ou char  , c'est un type "inventé".

Nous apprendrons nous aussi à créer nos propres types de variables quelques chapitres plus loin. Pour le moment, on va se contenter de stocker la valeur renvoyée par strlen dans une variable de type int  (l'ordinateur convertira de size\_t en int automatiquement).

En toute rigueur, il faudrait plutôt stocker le résultat dans une variable de type size\_t  , mais en pratique un int est suffisant pour cela.

La fonction prend un paramètre de type const char\*  .

Le const(  qui signifie *constante*) fait que la fonction strlen "s'interdit" en quelque sorte de modifier votre chaîne. Quand vous voyez un const  , vous savez que la variable n'est pas modifiée par la fonction, elle est juste lue.

Testons la fonction strlen  :

int main(int argc, char \*argv[])

{

char chaine[] = "Salut";

int longueurChaine = 0;

// On récupère la longueur de la chaîne dans longueurChaine

longueurChaine = strlen(chaine);

// On affiche la longueur de la chaîne

printf("La chaine %s fait %d caracteres de long", chaine, longueurChaine);

return 0;

}

La chaine Salut fait 5 caracteres de long

Pour écrire la fonction strlen   , il suffit de faire une boucle sur le tableau de char qui s'arrête quand on tombe sur le caractère \0  . Un compteur s'incrémente à chaque tour de boucle, et c'est ce compteur que la fonction retourne.

Tiens, tout ça m'a donné envie d'écrire moi-même une fonction similaire à strlen  . Ça vous permettra en plus de bien comprendre comment la fonction marche :

int longueurChaine(const char\* chaine);

int main(int argc, char \*argv[])

{

char chaine[] = "Salut";

int longueur = 0;

longueur = longueurChaine(chaine);

printf("La chaine %s fait %d caracteres de long", chaine, longueur);

return 0;

}

int longueurChaine(const char\* chaine)

{

int nombreDeCaracteres = 0;

char caractereActuel = 0;

do

{

caractereActuel = chaine[nombreDeCaracteres];

nombreDeCaracteres++;

}

while(caractereActuel != '\0'); // On boucle tant qu'on n'est pas arrivé à l'\0

nombreDeCaracteres--; // On retire 1 caractère de long pour ne pas compter le caractère \0

return nombreDeCaracteres;

}

La fonction longueurChaine fait une boucle sur le tableau chaine :

1. Elle stocke les caractères un par un dans caractereActuel  .
2. Dès que caractèreActuel vaut '\0'  , la boucle s'arrête.
3. À chaque passage dans la boucle, on ajoute 1 au nombre de caractères qu'on a analysés.
4. À la fin de la boucle, on retire 1 caractère au nombre total de caractères qu'on a comptés. Cela permet de ne pas compter le caractère \0  dans le lot.
5. Enfin, on retourne nombreDeCaracteres   , et le tour est joué !

**2. Copiez une chaîne dans une autre avec strcpy**

La fonction strcpy(pour "string copy") permet de copier une chaîne à l'intérieur d'une autre.

Voici son prototype :

char\* strcpy(char\* copieDeLaChaine, const char\* chaineACopier);

Cette fonction prend deux paramètres :

1. copieDeLaChaine  : c'est un pointeur vers un char\*  (tableau de char  ). C'est dans ce tableau que la chaîne sera copiée.
2. chaineACopier  : c'est un pointeur vers un autre tableau de char  . Cette chaîne sera copiée dans copieDeLaChaine  .

La fonction renvoie un pointeur sur copieDeLaChaine  , ce qui n'est pas très utile. En général, on ne récupère pas ce que cette fonction renvoie.

Testons cela :

int main(int argc, char \*argv[])

{

/\* On crée une chaîne "chaine" qui contient un peu de texte

et une copie (vide) de taille 100 pour être sûr d'avoir la place

pour la copie \*/

char chaine[] = "Texte", copie[100] = {0};

strcpy(copie, chaine); // On copie "chaine" dans "copie"

// Si tout s'est bien passé, la copie devrait être identique à chaine

printf("chaine vaut : %s\n", chaine);

printf("copie vaut : %s\n", copie);

return 0;

}

chaine vaut : Texte

copie vaut : Texte

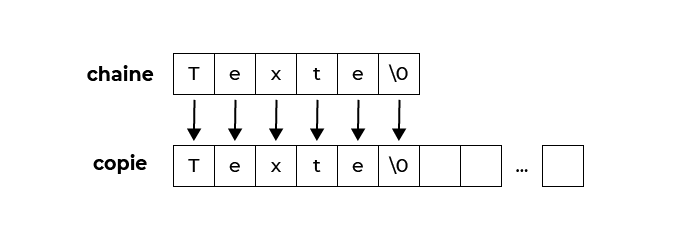
On voit que chaine vaut "Texte". Jusque-là, c'est normal.

Par contre, on voit aussi que la variable copie  , qui était vide au départ, a été remplie par le contenu de chaine  . La chaîne a donc bien été copiée dans copie  .

Vérifiez que la chaîne copie est assez grande pour accueillir le contenu de chaine  .

Si, dans mon exemple, j'avais défini copie[5]  (ce qui n'est pas suffisant car il n'y aurait pas eu de place pour le \0  ), la fonction strcpy aurait "débordé en mémoire" et probablement fait planter votre programme. À éviter à tout prix !

Schématiquement, la copie a fonctionné comme ceci :

Copie d'une chaîne de caractères

Chaque caractère de chaine a été placé dans copie  .

La chaîne copie contient de nombreux caractères inutilisés. Je lui ai donné la taille 100 par sécurité, mais en toute rigueur, la taille 6 aurait suffit. L'avantage de créer un tableau un peu plus grand, c'est que de cette façon la chaîne copie sera capable de recevoir d'autres chaînes peut-être plus grandes dans la suite du programme.

**3. Concaténez deux chaînes avec  strcat**

La fonction  strcat  (pour "string catenate") ajoute une chaîne à la suite d'une autre. On appelle cela la concaténation.

Supposons que l'on ait les variables suivantes :

1. chaine1 = "Salut "
2. chaine2 = "Mateo21"

Si je concatène chaine2 dans chaine1  , alors chaine1 vaudra "Salut Mateo21"  .

Quant à chaine2  , elle n'aura pas changé et vaudra donc toujours "Mateo21"  .

Seule chaine1 est modifiée.

C'est exactement ce que fait strcat  , dont voici le prototype :

char\* strcat(char\* chaine1, const char\* chaine2);

chaine2 ne peut pas être modifiée car elle est définie comme constante dans le prototype de la fonction.

La fonction retourne un pointeur vers chaine1  , ce qui, comme pour strcpy  , ne sert pas à grand-chose dans le cas présent : on peut donc ignorer ce que la fonction nous renvoie.

La fonction ajoute à chaine1 le contenu de chaine2  .

Regardons-y de plus près :

int main(int argc, char \*argv[])

{

/\* On crée 2 chaînes. chaine1 doit être assez grande pour accueillir

le contenu de chaine2 en plus, sinon risque de plantage \*/

char chaine1[100] = "Salut ", chaine2[] = "Mateo21";

strcat(chaine1, chaine2); // On concatène chaine2 dans chaine1

// Si tout s'est bien passé, chaine1 vaut "Salut Mateo21"

printf("chaine1 vaut : %s\n", chaine1);

// chaine2 n'a pas changé :

printf("chaine2 vaut toujours : %s\n", chaine2);

return 0;

}

chaine1 vaut : Salut Mateo21

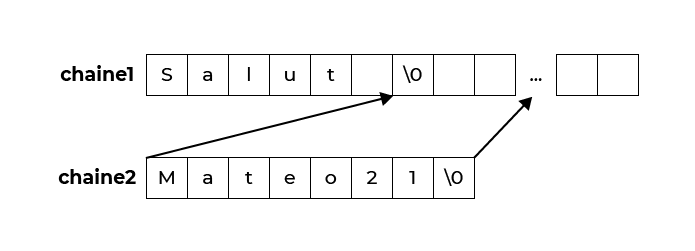
chaine2 vaut toujours : Mateo21

Vérifiez que chaine1 est assez grande pour qu'on puisse lui ajouter le contenu de chaine2   , sinon vous ferez un débordement en mémoire qui peut conduire à un plantage.

C'est pour cela que j'ai défini chaine1 de taille 100.

Quant à chaine2  , j'ai laissé l'ordinateur calculer sa taille (je n'ai donc pas précisé la taille) car cette chaîne n'est pas modifiée, il n'y a donc pas besoin de la rendre plus grande que nécessaire.

Voici comment fonctionne la concaténation :

Concaténation de chaînes

Le tableau chaine2 a été ajouté à la suite de chaine1  (qui comprenait une centaine de cases).

Le \0 de chaine1 a été supprimé (en fait, il a été remplacé par le M de Mateo21).

En effet, il ne faut pas laisser un \0  au milieu de la chaîne, sinon celle-ci aurait été "coupée" au milieu ! On ne met un \0  qu'à la fin de la chaîne, une fois qu'elle est finie.

**4. Comparez deux chaînes avec strcmp**

strcmp (pour "string compare") compare 2 chaînes entre elles.

Voici son prototype :

int strcmp(const char\* chaine1, const char\* chaine2);

Les variables chaine1 et chaine2 sont comparées. Aucune d'elles n'est modifiée car elles sont indiquées comme constantes.

Il est important de récupérer ce que la fonction strcmp renvoie :

* 0 si les chaînes sont identiques ;
* une autre valeur (positive ou négative) si les chaînes sont différentes.

Il aurait été plus logique que la fonction renvoie 1 si les chaînes avaient été identiques, pour dire "vrai" (rappelez-vous des booléens).

La raison est simple : la fonction compare les valeurs de chacun des caractères un à un :

* si tous les caractères sont identiques, elle renvoie 0 ;
* si les caractères de la chaine1 sont supérieurs à ceux de la chaine2  , la fonction renvoie un nombre positif ;
* si c'est l'inverse, la fonction renvoie un nombre négatif.

Dans la pratique, on se sert surtout de strcmp pour vérifier si deux chaînes sont identiques ou non.

Voici un code de test :

int main(int argc, char \*argv[])

{

char chaine1[] = "Texte de test", chaine2[] = "Texte de test";

if (strcmp(chaine1, chaine2) == 0) // Si chaînes identiques

{

printf("Les chaines sont identiques\n");

}

else

{

printf("Les chaines sont differentes\n");

}

return 0;

}

Les chaines sont identiques

Les chaînes étant identiques, la fonction strcmp a renvoyé le nombre 0.

Notez que j'aurais pu stocker ce que renvoie strcmp dans une variable de type int  . Toutefois, ce n'est pas obligatoire, on peut directement mettre la fonction dans le if comme je l'ai fait.

N'oubliez pas :

* 0 signifie "identique" ;
* et une autre valeur signifie "différent".

C'est la seule source d'erreur possible ici.

**5. Recherchez un caractère avec  strchr**

La fonction strchr recherche un caractère dans une chaîne.

Voici son prototype :

char\* strchr(const char\* chaine, int caractereARechercher);

La fonction prend 2 paramètres :

1. chaine  : la chaîne dans laquelle la recherche doit être faite.
2. caractereARechercher  : le caractère que l'on doit rechercher dans la chaîne.

caractereARechercher est de type int et non de typechar  .

Ce n'est pas réellement un problème car, au fond, un caractère est et restera toujours un nombre. Néanmoins, on utilise quand même plus souvent un char qu'un int pour stocker un caractère en mémoire.

La fonction renvoie un pointeur vers le premier caractère qu'elle a trouvé, c'est-à-dire qu'elle renvoie l'adresse de ce caractère dans la mémoire. Elle renvoie NULL si elle n'a rien trouvé.

Dans l'exemple suivant, je récupère ce pointeur dans suiteChaine  :

int main(int argc, char \*argv[])

{

char chaine[] = "Texte de test", \*suiteChaine = NULL;

suiteChaine = strchr(chaine, 'd');

if (suiteChaine != NULL) // Si on a trouvé quelque chose

{

printf("Voici la fin de la chaine a partir du premier d : %s", suiteChaine);

}

return 0;

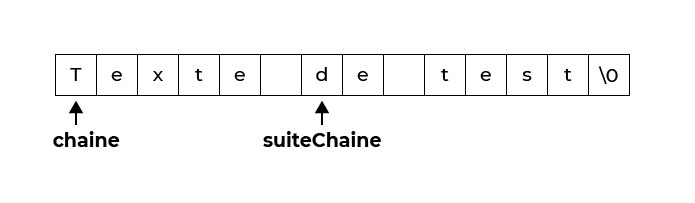
}

Voici la fin de la chaine a partir du premier d : de test

Avez-vous bien compris ce qu'il se passe ici ? C'est un peu particulier.

En fait, suiteChaine est un pointeur comme chaine  , sauf que chaine pointe sur le premier caractère (le 'T'  majuscule), tandis que suiteChaine pointe sur le premier caractère 'd' qui a été trouvé dans chaine  .

Ce schéma montre où pointe chaque pointeur :

Pointeurs et chaînes

chaine commence au début de la chaine ( 'T'  majuscule), tandis que suiteChaine pointe sur le 'd' minuscule.

Lorsque je fais un printf de suiteChaine  , il est donc normal que l'on m'affiche juste "de test".

printf affiche tous les caractères qu'elle rencontre (d,e, ,t,e,s,t) jusqu'à ce qu'elle tombe sur \0 qui lui dit que la chaîne s'arrête là.

Il existe une fonction strrchr strictement identique à strchr  , sauf que celle-là renvoie un pointeur vers **le dernier caractère** qu'elle a trouvé dans la chaîne, plutôt que vers le premier.

**6. Recherchez le premier caractère d'une liste avec strpbrk**

Cette fonction recherche un des caractères dans la liste que vous lui donnez sous forme de chaîne, contrairement à strchr qui ne peut rechercher qu'un seul caractère à la fois.

Si on forme la chaîne "xds" et qu'on en fait une recherche dans "Texte de test"  , la fonction renvoie un pointeur vers le premier de ces caractères qu'elle y a trouvés. En l'occurrence, le premier caractère de "xds"  qu'elle trouve dans "Texte de test"  est le x  , donc strpbrk renverra un pointeur sur 'x'  .

Voici son prototype :

char\* strpbrk(const char\* chaine, const char\* lettresARechercher);

Testons la fonction :

int main(int argc, char \*argv[])

{

char \*suiteChaine;

// On cherche la première occurrence de x, d ou s dans "Texte de test"

suiteChaine = strpbrk("Texte de test", "xds");

if (suiteChaine != NULL)

{

printf("Voici la fin de la chaine a partir du premier des caracteres trouves : %s", suiteChaine);

}

return 0;

}

Voici la fin de la chaine a partir du premier des caracteres trouves :

xte de test

Pour cet exemple, j'ai directement écrit les valeurs à envoyer à la fonction (entre guillemets). Rien ne nous oblige à employer une variable à tous les coups, on peut très bien écrire la chaîne directement. Il faut simplement retenir la règle suivante :

* si vous utilisez les guillemets ""  , cela signifie **chaîne**;
* si vous utilisez les apostrophes ''  , cela signifie **caractère**.

**7. Recherchez une chaîne dans une autre avec strstr**

La fonction  strstr  (pour "string string") recherche la première occurrence d'une chaîne dans une autre chaîne.

Son prototype est :

char\* strstr(const char\* chaine, const char\* chaineARechercher);

Le prototype est similaire à strpbrk  , mais attention à ne pas confondre :

* strpbrk recherche UN des caractères ;
* strstr recherche toute la chaîne.

Exemple :

int main(int argc, char \*argv[])

{

char \*suiteChaine;

// On cherche la première occurrence de "test" dans "Texte de test" :

suiteChaine = strstr("Texte de test", "test");

if (suiteChaine != NULL)

{

printf("Premiere occurrence de test dans Texte de test : %s\n", suiteChaine);

}

return 0;

}

Premiere occurrence de test dans Texte de test : test

La fonction strstr recherche la chaîne "test" dans "Texte de test" :

* elle renvoie, comme les autres, un pointeur quand elle a trouvé ce qu'elle cherchait ;
* elle renvoieNULL si elle n'a rien trouvé.

Jusqu'ici, je me suis contenté d'afficher la chaîne à partir du pointeur retourné par les fonctions. Dans la pratique, ça n'est pas très utile. Vous ferez juste un if (resultat != NULL)  pour savoir si la recherche a ou non donné quelque chose, et vous afficherez "Le texte que vous recherchiez a été trouvé".