Introduction au Langage C

Vincent Vidal

Maître de Conférences
Enseignements : IUT Lyon 1 - pôle AP - Licence ESSIR - bureau 2ème étage
Recherche : Laboratoire LIRIS - bât. Nautibus
E-mail : vincent.vidal@univ-lyon1.fr

Supports de cours et TPs : `http://claroline connect.univ-lyon1.fr` espace d'activités "M1102 M1103 C - Introduction au langage C"

46H prévues $\approx$ 42H de cours+TPs, 2H - interros, et 2H - examen final

Évaluation : Contrôle continu + examen final + Bonus/Malus TP

Plan

Plan

En mémoire

En C, il n'existe pas de type chaîne, on travaille sur des **tableaux de caractères terminés par un caractère particulier** '\0' :

- La taille du tableau est fixe, ce qui impose une longueur max aux chaînes et une perte de place mémoire si la chaîne n'utilise pas toute la longueur du tableau.
- La manipulation des informations est réalisée caractère par caractère.

Chaîne de caractères en C

En C, une chaîne de $n \geq 0$ caractère(s) est une suite contiguë de $n+1$ octet(s) en mémoire, les n premiers octets correspondant aux n caractères de la chaîne, et le dernier octet supplémentaire au code nul ('\0') qui symbolise la fin de chaîne.

En mémoire

En C, il n'existe pas de type chaîne, on travaille sur des **tableaux de caractères terminés par un caractère particulier** '\0' :

- La taille du tableau est fixe, ce qui impose une longueur max aux chaînes et une perte de place mémoire si la chaîne n'utilise pas toute la longueur du tableau.
- La manipulation des informations est réalisée caractère par caractère.

Chaîne de caractères en C

En C, une chaîne de $n \geq 0$ caractère(s) est une suite contiguë de $n + 1$ octet(s) en mémoire, les n premiers octets correspondant aux n caractères de la chaîne, et le dernier octet supplémentaire au code nul ('\0') qui symbolise la fin de chaîne.

En mémoire

En C, il n'existe pas de type chaîne, on travaille sur des **tableaux de caractères terminés par un caractère particulier** '\0' :

- La taille du tableau est fixe, ce qui impose une longueur max aux chaînes et une perte de place mémoire si la chaîne n'utilise pas toute la longueur du tableau.
- La manipulation des informations est réalisée caractère par caractère.

Chaîne de caractères en C

En C, une chaîne de $n \geq 0$ caractère(s) est une suite contiguë de $n + 1$ octet(s) en mémoire, les n premiers octets correspondant aux n caractères de la chaîne, et le dernier octet supplémentaire au code nul ('\0') qui symbolise la fin de chaîne.

Les chaînes constantes

En C, une suite de caractères entre guillemets est une **chaîne de caractères constante**, par exemple **"**ma chaine constante**"** :

- Le compilateur crée en mémoire la suite d'octets correspondant à la chaîne (sans oublier le caractère nul en fin de chaîne).
- La valeur de "ma chaine constante" n'est pas la valeur de la chaîne elle-même, mais l'adresse de son premier octet en mémoire.
  Ainsi l'instruction "chaine" == "chaine" retournera 0 (faux) en C.
  Pour comparer 2 chaînes il faudra utiliser une fonction (e.g. strcmp).

Les chaînes constantes

En C, une suite de caractères entre guillemets est une **chaîne de caractères constante**, par exemple **"**ma chaine constante**"** :

- Le compilateur crée en mémoire la suite d'octets correspondant à la chaîne (sans oublier le caractère nul en fin de chaîne).
- La valeur de **"**ma chaine constante**"** n'est pas la valeur de la chaîne elle-même, mais l'adresse de son premier octet en mémoire.

  Ainsi l'instruction "chaine" == "chaine" retournera 0 (faux) en C.

  Pour comparer 2 chaînes il faudra utiliser une fonction (e.g. strcmp).

Les chaînes constantes

En C, une suite de caractères entre guillemets est une **chaîne de caractères constante**, par exemple **"**ma chaine constante**"** :

- Le compilateur crée en mémoire la suite d'octets correspondant à la chaîne (sans oublier le caractère nul en fin de chaîne).
- La valeur de **"**ma chaine constante**"** n'est pas la valeur de la chaîne elle-même, mais l'adresse de son premier octet en mémoire.
  Ainsi l'instruction "chaine" == "chaine" retournera 0 (faux) en C.
  Pour comparer 2 chaînes il faudra utiliser une fonction (e.g. strcmp).

Les chaînes constantes

En C, une suite de caractères entre guillemets est une **chaîne de caractères constante**, par exemple **"**ma chaine constante**"** :

- Le compilateur crée en mémoire la suite d'octets correspondant à la chaîne (sans oublier le caractère nul en fin de chaîne).
- La valeur de **"**ma chaine constante**"** n'est pas la valeur de la chaîne elle-même, mais l'adresse de son premier octet en mémoire.
  Ainsi l'instruction "chaine" == "chaine" retournera 0 (faux) en C.
  Pour comparer 2 chaînes il faudra utiliser une fonction (e.g. strcmp).

Exemple : afficher une chaîne de caractères

```
#include <stdio.h>
int main(void) {
    char * adr ;
    adr = "bonjour" ;
    while ( *adr != '\0' )
    {
        printf("%c", *adr) ;
        adr++ ;
    }

    return 0;
}
```

Convention

Initialisation d'un tableau de caractères à l'aide d'une chaîne ?

```c
int main(void) {
   /* voici 2 init autorisées et équivalentes en C : */
   char tab_car[15] = "Salut!";
   char tab_car2[15] = { 'S', 'a', 'l', 'u', 't', '!', '\0' };
   /* Que se passe-t-il pour les cases non initialisées? */

   /* init optimale en mémoire : */
   char tab_car3[] = "Salut!"; /* un tableau de 7 caractères est réservé */

   /* exemple non autorisé en C : */
   char tab_car4[15];
   tab_car4 = "Salut!"; /* engendre une erreur car tab_car4 est une @ constante! */

   return 0;
}
```

Convention

Initialisation d'un tableau de pointeurs sur des chaînes ?

```c
#include <stdio.h>
int main(void) {
   /* déclaration et init d'un tableau de 7 chaînes */
   char * jour[7] = { "lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi", "dimanche" } ;
   int j ;
   printf( "Donnez un entier entre 1 et 7 : ") ;
   scanf("%d", &j) ;
   if(j<1 || j > 7)
   {
      printf( "Erreur de saisie.\n") ;
      return -1;
   }

   printf("Le jour de la semaine numero %d est %s.\n", j, jour[j-1]) ;

   return 0;
}
```

Possibilités

- Utilisation du code format %s dans les fonctions **scanf** et **printf**.
- Utilisation des fonctions de lecture **gets** et d'affichage **puts**.

Les fonctions **printf** et **scanf** permettent d'afficher ou de lire plusieurs chaînes, tandis que les fonctions **puts** et **gets** ne traitent qu'**une chaîne à la fois**.

Exemple avec scanf et printf

```c
#include <stdio.h>
#define TAILLE_MAX 256 /* longueur max de chaîne sans le '\0' */
int main(void) {
   char ma_chaine[TAILLE_MAX+1], ma_chaine2[TAILLE_MAX+1];
   printf("Saisissez une chaine de caracteres : ");
   scanf("%s", ma_chaine);  /* scanf("%s", &ma_chaine); fonctionne aussi
                               car &ma_chaine==ma_chaine ; ma_chaine est déjà une @ */
   printf("Vous avez entre %s.\n", ma_chaine);

   printf("Saisissez deux chaines de caracteres : ");
   scanf("%s%s", ma_chaine, ma_chaine2); /* une chaine est délimitée par un ' ' ou un '\n' ;
                                            le caractère delimiteur n'est pas consommé */
   printf("Vous avez entre %s et %s.\n", ma_chaine, ma_chaine2);

   return 0;
}
```

Exemple avec gets et puts

```c
#include <stdio.h>
#define TAILLE_MAX 256 /* longueur max de chaîne sans le '\0' */
int main(void) {
   char ma_chaine[TAILLE_MAX+1];
   printf("Saisissez une chaine de caracteres : ");
   gets(ma_chaine); /* lit tout jusqu'au 1er '\n' rencontré (les ' ' sont conservés) */

   puts(ma_chaine); /* puts(ma_chaine); équivaut à printf("%s\n", ma_chaine); */

   return 0;
}
```

Que se passe-t-il si la chaîne à lire est trop longue ?

```c
#include <stdio.h>
#define TAILLE_MAX 5 /* longueur max de chaîne sans le '\0' */
int main(void) {
   char ma_chaine[TAILLE_MAX+1];
   printf("Saisissez une chaine de caracteres : ");
   scanf("%s", ma_chaine);  /* scanf("%s", &ma_chaine); fonctionne aussi
                               car &ma_chaine==ma_chaine ; ma_chaine est déjà une @ */
   printf("Vous avez entre %s.\n", ma_chaine);

   return 0;
}
```

scanf et gets écrivent au-delà du buffer (tableau de char) alloué : **dépassement de buffer** ! Utiliser %5s pour scanf ou une TAILLE_MAX >= à la longueur maximale d'une ligne tapée au clavier.

Solution portable : fgets(ma_chaine, TAILLE_MAX+1, stdin) ;

Que se passe-t-il si la chaîne à lire est trop longue ?

```c
#include <stdio.h>
#define TAILLE_MAX 5 /* longueur max de chaîne sans le '\0' */
int main(void) {
   char ma_chaine[TAILLE_MAX+1];
   printf("Saisissez une chaine de caracteres : ");
   scanf("%s", ma_chaine);  /* scanf("%s", &ma_chaine); fonctionne aussi
                               car &ma_chaine==ma_chaine ; ma_chaine est déjà une @ */
   printf("Vous avez entre %s.\n", ma_chaine);

   return 0;
}
```

scanf et gets écrivent au-delà du buffer (tableau de char) alloué : **dépassement de buffer** ! Utiliser %5s pour scanf ou une TAILLE_MAX >= à la longueur maximale d'une ligne tapée au clavier.

Solution portable : fgets(ma_chaine, TAILLE_MAX+1, stdin) ;

Que se passe-t-il si la chaîne à lire est trop longue ?

```c
#include <stdio.h>
#define TAILLE_MAX 5 /* longueur max de chaîne sans le '\0' */
int main(void) {
   char ma_chaine[TAILLE_MAX+1];
   printf("Saisissez une chaine de caracteres : ");
   scanf("%s", ma_chaine);  /* scanf("%s", &ma_chaine); fonctionne aussi
                               car &ma_chaine==ma_chaine ; ma_chaine est déjà une @ */
   printf("Vous avez entre %s.\n", ma_chaine);

   return 0;
}
```

scanf et gets écrivent au-delà du buffer (tableau de char) alloué : **dépassement de buffer** ! Utiliser %5s pour scanf ou une TAILLE_MAX >= à la longueur maximale d'une ligne tapée au clavier.

Solution portable : **fgets**(ma_chaine, TAILLE_MAX+1, stdin) ;

Solution satisfaisante

```c
#include <stdio.h>
#define TAILLE_MAX 256 /* longueur max de chaîne sans le '\0' */
int main(void) {
   char ligne_saisie[TAILLE_MAX+1];
   int nb_param_bien_saisis=0;
   int n; /* entier à saisir */
   char c; /* caractère à saisir différent de ' ' */

   do
   {
      printf("Saisir un entier et un caractere au clavier : ");
      gets(ligne_saisie);
      nb_param_bien_saisis = sscanf(ligne_saisie, "%d %c", &n, &c);
      if(nb_param_bien_saisis<2) printf("\n Erreur de Saisie!\n");
   }while(nb_param_bien_saisis<2) ;

   printf("Merci, vous avez entre une valeur valide pour l'entier (%d) et pour le caractere
      (%c).\n", n, c);

   return 0;
}
```

Types des paramètres des fonctions de manipulation de chaînes C dans string.h

- Pour une chaîne de caractères fournie en donnée/entrée : **const char ***
- Pour une chaîne de caractères fournie en résultat/sortie ou en donnée-résultat/entrée-sortie : **char ***

Types des paramètres des fonctions de manipulation de chaînes C dans string.h

- Pour une chaîne de caractères fournie en donnée/entrée : **const char ***
- Pour une chaîne de caractères fournie en résultat/sortie ou en donnée-résultat/entrée-sortie : **char ***

Les fonctions à connaître (1/2)

Définies dans de fichier d'en-tête string.h (**#include** <string.h>) :

- Longueur d'une chaîne : size_t **strlen**(const char*) ;
- Copie d'une chaîne dans une autre : **strcpy**,
  char * **strncpy**(char * dest, const char * src, size_t n) ;
- Copie d'une chaîne à la suite de la première (concaténation) : **strcat**,
  char * **strncat**(char * dest, const char * src, size_t n) ;
- Comparaison de 2 chaînes au sens de l'ordre du code ASCII de leurs caractères : sensible à la casse : **strcmp**, **strncmp** ; pas sensible : **stricmp**, **strnicmp**
- Recherche dans une chaîne : d'un caractère : **strchr**, **strrchr** ; d'une sous-chaîne : **strstr**

Les fonctions à connaître (1/2)

Définies dans de fichier d'en-tête string.h (**#include** <string.h>) :

- Longueur d'une chaîne : size_t **strlen**(const char*) ;
- Copie d'une chaîne dans une autre : **strcpy**,
  char * **strncpy**(char * dest, const char * src, size_t n) ;
- Copie d'une chaîne à la suite de la première (concaténation) : **strcat**,
  char * **strncat**(char * dest, const char * src, size_t n) ;
- Comparaison de 2 chaînes au sens de l'ordre du code ASCII de leurs caractères : sensible à la casse : **strcmp**, **strncmp** ; pas sensible : **stricmp**, **strnicmp**
- Recherche dans une chaîne : d'un caractère : **strchr**, **strrchr** ; d'une sous-chaîne : **strstr**

Les fonctions à connaître (1/2)

Définies dans de fichier d'en-tête string.h (**#include** <string.h>) :

- Longueur d'une chaîne : size_t **strlen**(const char*) ;
- Copie d'une chaîne dans une autre : **strcpy**,
  char * **strncpy**(char * dest, const char * src, size_t n) ;
- Copie d'une chaîne à la suite de la première (concaténation) : **strcat**,
  char * **strncat**(char * dest, const char * src, size_t n) ;
- Comparaison de 2 chaînes au sens de l'ordre du code ASCII de leurs caractères : sensible à la casse : **strcmp**, **strncmp** ; pas sensible : **stricmp**, **strnicmp**
- Recherche dans une chaîne : d'un caractère : **strchr**, **strrchr** ; d'une sous-chaîne : **strstr**

Les fonctions à connaître (1/2)

Définies dans de fichier d'en-tête string.h (**#include** <string.h>) :

- Longueur d'une chaîne : size_t **strlen**(const char*) ;
- Copie d'une chaîne dans une autre : **strcpy**,
  char * **strncpy**(char * dest, const char * src, size_t n) ;
- Copie d'une chaîne à la suite de la première (concaténation) : **strcat**,
  char * **strncat**(char * dest, const char * src, size_t n) ;
- Comparaison de 2 chaînes au sens de l'ordre du code ASCII de leurs caractères : sensible à la casse : **strcmp**, **strncmp** ; pas sensible : **stricmp**, **strnicmp**
- Recherche dans une chaîne : d'un caractère : **strchr**, **strrchr** ; d'une sous-chaîne : **strstr**

Les fonctions à connaître (1/2)

Définies dans de fichier d'en-tête string.h (**#include** <string.h>) :

- Longueur d'une chaîne : size_t **strlen**(const char*) ;
- Copie d'une chaîne dans une autre : **strcpy**,
  char * **strncpy**(char * dest, const char * src, size_t n) ;
- Copie d'une chaîne à la suite de la première (concaténation) : **strcat**,
  char * **strncat**(char * dest, const char * src, size_t n) ;
- Comparaison de 2 chaînes au sens de l'ordre du code ASCII de leurs caractères : sensible à la casse : **strcmp**, **strncmp** ; pas sensible : **stricmp**, **strnicmp**
- Recherche dans une chaîne : d'un caractère : **strchr**, **strrchr** ; d'une sous-chaîne : **strstr**

Les fonctions à connaître (2/2)

Définies dans de fichier d'en-tête stdlib.h (**#include** <stdlib.h>) :

- **atoi** : convertit une chaîne en **int**
- **atol** : convertit une chaîne en **long**
- **atof** : convertit une chaîne en **double**

Définies dans le fichier d'en-tête stdio.h (**#include** <stdio.h>) :

- De valeur numérique vers chaîne ?
  int **sprintf**(char* dest, const char* format, ...)
  int **snprintf**(char* dest, int n, const char* format, ...) depuis C++11

Les fonctions à connaître (2/2)

Définies dans de fichier d'en-tête stdlib.h (**#include** <stdlib.h>) :

- **atoi** : convertit une chaîne en **int**
- **atol** : convertit une chaîne en **long**
- **atof** : convertit une chaîne en **double**

Définies dans le fichier d'en-tête stdio.h (**#include** <stdio.h>) :

- De valeur numérique vers chaîne ?
  int **sprintf**(char* dest, const char* format, ...)
  int **snprintf**(char* dest, int n, const char* format, ...) depuis C++11

Les fonctions à connaître (2/2)

Définies dans de fichier d'en-tête stdlib.h (**#include** <stdlib.h>) :

- **atoi** : convertit une chaîne en **int**
- **atol** : convertit une chaîne en **long**
- **atof** : convertit une chaîne en **double**

Définies dans le fichier d'en-tête stdio.h (**#include** <stdio.h>) :

- De valeur numérique vers chaîne ?
  int **sprintf**(char* dest, const char* format, ...)
  int **snprintf**(char* dest, int n, const char* format, ...) depuis C++11

Remarque sur les 2 derniers slides

Lorsqu'une fonction a un argument résultat (*dest* dans les 2 derniers slides), alors on doit obligatoirement passer un argument qui contient une adresse d'un bloc d'octets modifiables.

Ainsi, on ne peut ni utiliser un pointeur ne pointant sur aucune adresse mémoire valide, ni utiliser une chaîne de caractères constante.

`char* p; strcpy(p, "la");`, `const char ch[3]; strcpy(ch, "la");` ou `strcpy("ici", "la");` ne sont pas des appels valides.

`char ch[3]; strcpy(ch, "la");` est valide.

Remarque sur les 2 derniers slides

Lorsqu'une fonction a un argument résultat (*dest* dans les 2 derniers slides), alors on doit obligatoirement passer un argument qui contient une adresse d'un bloc d'octets modifiables.

Ainsi, on ne peut ni utiliser un pointeur ne pointant sur aucune adresse mémoire valide, ni utiliser une chaîne de caractères constante.

`char* p; strcpy(p, "la");`, `const char ch[3]; strcpy(ch, "la");` ou `strcpy("ici", "la");` ne sont pas des appels valides.

`char ch[3]; strcpy(ch, "la");` est valide.

Remarque sur les 2 derniers slides

Lorsqu'une fonction a un argument résultat (*dest* dans les 2 derniers slides), alors on doit obligatoirement passer un argument qui contient une adresse d'un bloc d'octets modifiables.

Ainsi, on ne peut ni utiliser un pointeur ne pointant sur aucune adresse mémoire valide, ni utiliser une chaîne de caractères constante.

`char* p; strcpy(p, "la");`, `const char ch[3]; strcpy(ch, "la");` ou `strcpy("ici", "la");` ne sont pas des appels valides.

`char ch[3]; strcpy(ch, "la");` est valide.

Dans le 1er cours, on a dit que **int** main(**int** nbarg, **char** * argv[]) est une en-tête possible du main. nbarg est le nombre d'arguments sur la ligne de commande (nom du programme inclu) et **char** * argv[] est un tableau de chaînes de caractères de longueur variable : il contiendra tous les arguments (1 argument = une chaîne) passés par l'utilisateur du programme.

```
#include <stdio.h>
#include <stdarg.h>
int main(int nbarg, char * argv[]) {
   int i ;
   printf("Le nom du programme est : %s\n", argv[0]) ;

   if(nbarg>1) for(i=1; i<nbarg; i++) printf("L'argument numero %d est %s\n", i, argv[i]);
   else printf("Aucun argument n'a ete passe au programme.\n");

   return 0;
}
```