Plan

1 Les chaînes de caractères

Chaînes de caractères

Définition

Une **chaîne de caractères** est une structure de données complexe capable de stocker du texte.

Rappel :

- Un caractère est représentable en `C` par un entier stocké sur un `char` indiquant le code ASCII du caractère.
- Lors de la lecture clavier/sortie écran d'un caractère une conversion code ASCII/caractère est faite grâce au formateur `%c` des fonctions d'entrée/sortie formatée `printf` et `scanf`.

↪ Une chaîne de caractères est donc naturellement représentable par un **tableau de `char`**.

Chaînes de caractères C : *string*

La transmission de tableaux en C impose de transmettre en même temps leur taille. Pour facilite la manipulation des chaînes de caractères et ne pas avoir besoin de transmettre leur taille, C a fait le choix d'intégrer un **marqueur de fin de chaîne**.

Définition

Une chaîne de caractères C, appelée **string** (*null terminated string*) est un tableau de char contenant le caractère spécifique `'\0'` (de code ASCII 0) qui représente le marqueur de fin de chaîne.

Remarques :

- *string* n'est pas un type, le type d'une chaîne de caractères reste celui d'un tableau de `char` : `char *`
- *string* est une contrainte imposée aux tableaux de `char` de contenir le caractère `'\0'`

↪ Toute *string* est un tableau de `char` mais tout tableau de `char` n'est pas une *string* (il doit contenir un `'\0'`).

Contenu, longueur et taille des *string*

Définition

Le **contenu** d'une *string* est la séquence de caractères du tableau située avant le premier caractère `'\0'`.
La **longueur** d'une *string* est la longueur de cette séquence.
La **taille** d'une *string* reste la taille du tableau la contenant.

Remarques :

- Attention, la taille du tableau d'une *string* est donc strictement supérieure à la longueur de cette *string* (au moins un caract. en plus :`'\0'`).
- Différentes *strings* peuvent représenter la même séquence de caractères.

Exemple

`[h|e|l|l|o|\0]` représente la chaîne "hello" (stockée dans un tab. de taille 6).
`[h|e|l|l|o|\0|w|o|r|l|d|\0]` : chaîne "hello" (tab. de taille 12).
`[h|e|l|l|o]` : n'est pas une *string* seulement un tab. de taille 5.
`[\0]` : chaîne vide "" (tab. de taille 1).
`[\0|h|e|l|l|o|\0|\0|\0|\0]` : chaîne vide "" (tab. de taille 10).

Calcul de la longueur d'une *string*

La longueur d'une *string* peut être facilement calculée en parcourant son tableau de `char` jusqu'au premier caractère `'\0'`, sans avoir besoin de connaître la taille de son tableau.

```
/**
\brief calcule la longueur d'une string
\param s est une string (un tableau de char contenant '\0')
\return le nombre de caracteres de la chaine representee
*/
int long(char *s) {
    int i = 0;
    while(s[i]!='\0') i++;
    return i;
}
```

Exemple

```
char s[6]={'h','e','l','l','o','\0'}; printf("%d",long(s)); ⇒ 5
char s[6]={'h','\0','e','l','l','o'}; printf("%d",long(s)); ⇒ 1
char s[6]={'h','e','l','l','o'}; printf("%d",long(s)); ⇒ 5
char s[5]={'h','e','l','l','o'}; printf("%d",long(s)); ⇒ Erreur
```

Initialisation des *string*

Les *string* peuvent être initialisées de la même manière que les tableaux :

```
char s[6]; s[0]='h'; s[1]='e'; s[2]='l'; s[3]='l'; s[4]='o'; s[5]='\0';
char s[6] = {'h','e','l','l','o','\0'};
char s[6] = {'h','e','l','l','o',0};
char s[25] = {'h','e','l','l','o','\0'};
char s[6] = {'h','e','l','l','o'};
char s[] = {'h','e','l','l','o','\0'};

char *s; s = {'h','e','l','l','o','\0'};
```

On peut alternativement utiliser une **constante littérale spécifique** aux *string*, une chaîne entre guillemets :

`"chaine"` est équivalent à `{'c','h','a','i','n','e','\0'}`

```
char s[6] = "hello";
char s[25] = "hello";
char *s = "hello";
```

Attention : `char s[5] = "hello";` ne fait pas de `s` une *string* mais un simple tableau de caractères.

Saisie/Affichage de *string*

printf et **scanf** disposent d'un spécificateur de format spécifique aux *string* : `%s`.

- Lors de la lecture d'une *string*, il faut s'assurer que le tableau de `char` devant accueillir la chaîne contient suffisamment de place pour les caractères de la chaîne + le caractère `'\0'`.

Solution

On indique à la fonction **scanf** une limite n de remplissage pour que même si plus de caractères sont disponibles dans le buffer d'entrée, elle ne considère que les n premiers :

↪ On introduit ce n dans le spécificateur : `%ns`.
Le tableau receveur doit avoir une taille d'au moins $n+1$ caractères.

```
...
char nom[51];
scanf("%50s",nom);
printf("Bonjour %s !\n",nom);
...
```

Remarque : la chaîne saisie par `scanf` démarre au premier caractère du buffer différent de <espace>, <tabulation>, <entrée> et s'arrête dès que l'un de ces caractères est trouvé (ou si la limite est atteinte).

Protégeons les paramètres

Le C offre un moyen de "bloquer" la modification de paramètres : `const`

- Empêcher la modification d'un paramètre

```
void f(int const i; int *const p) {
    i = 2;              Erreur de compilation
    p = NULL;           Erreur de compilation
    *p = 5;             // On peut modifier l'objet pointe
    int *q = &i;        Warning
    *q = 5;             // mais on peut modifier !
}
```

Protégeons les paramètres

Le C offre un moyen de "bloquer" la modification de paramètres : `const`

- Empêcher la modification d'un paramètre

```
void f(int const i; int *const p) {
    i = 2;              Erreur de compilation
    p = NULL;           Erreur de compilation
    *p = 5;             // On peut modifier l'objet pointe
    int *q = &i;        Warning
    *q = 5;             // mais on peut modifier !
}
```

- Empêcher la modification d'un objet pointé

```
void f(const int * p) {
    *p = 5;         Erreur de compilation
    int *q = p;     Warning
    *q = 5;         // mais on peut modifier !
    p = NULL;       // On peut modifier le pointeur
}
```

Protégeons les paramètres

Le C offre un moyen de "bloquer" la modification de paramètres : `const`

- Empêcher la modification d'un paramètre

```
void f(int const i; int *const p) {
    i = 2;          Erreur de compilation
    p = NULL;       Erreur de compilation
    *p = 5;         // On peut modifier l'objet pointe
    int *q = &i;    Warning
    *q = 5;         // mais on peut modifier !
}
```

- Empêcher la modification d'un objet pointé

```
void f(const int * p) {
    *p = 5;      Erreur de compilation
    int *q = p; Warning
    *q = 5;      // mais on peut modifier !
    p = NULL;    // On peut modifier le pointeur
}
```

- Les deux

```
void f(const int *const p) {
    p = NULL;    Erreur de compilation
    *p = 5;      Erreur de compilation
    int *q = p; Warning
    *q = 5;      // mais on peut modifier !
}
```

Bibliothèque `string`

La bibliothèque `string.h` fournit un ensemble de fonctions de manipulation de `C string` dont :

- `size_t strlen(const char *s)` calcule la longueur de s.
- `int strcmp(const char *s1, const char *s2)` compare deux string dans l'ordre lexicographique et retourne un entier négatif, nul ou positif selon que la s1 précède, est identique ou suit s2.
- `char *strcat(char *s1, const char *s2)` concatène s2 à s1 (dont le tableau doit avoir suffisamment de place pour accueillir s2 en plus de s1) et retourne s1.
- `char *strncat(char *s1, const char *s2)` copie les n premiers caractères de s2 à la fin de s1 puis met le caractère fin de chaîne (s1 doit avoir suffisamment de place) puis retourne s1.
- `char *strstr(const char *s1, const char *s2)` retourne un pointeur sur la première occurrence de s2 dans s1 ou NULL si s2 n'apparait pas dans s1.