Chaînes

Octobre 2015

Chaînes

En programmation, nous avons souvent besoin de lire ou de manipuler des séquences de caractères. Pour compléter les chaînes littérales décrites précédemment, la bibliothèque standard fournit le type stringxe "string (bibliothèque)"xe "bibliothèque standard:string". Avec ce type, vous disposez de diverses opérations très utiles pour les chaînes comme la concaténation, la lecture d'un caractère ou d'une ligne complète, l'écriture d'un caractère ou d'une chaîne, etc. Les exemples qui suivent illustrent quelques-unes de ces opérations. Code 2.3 : lecture et écriture via les flux standard

```
1 : #include<iostream>
2 : #include<string>
3 : using namespace std;
4 : int main()
5 : {
6 :   char nom[50];   //le nom est stocké dans un tableau de caractères
7 :   string prenom; //le prénom est stocké dans une
8 :                 //variable de type string
9 :   string metier;
10:
11:   cout << "Saisissez votre nom:t";
12:   cin >> nom;
13:   cout << "nSaisissez votre prénom:t";
14:   cin >> prenom;
15:
16:   string resultat = "nnBonjour " + prenom + " " + nom;
17: resultat += 'n'; // équivalent à resultat=resultat+'n'
18:
19:   cout << resultat;
20:
21: }
```

Voici un exemple d'exécution de ce programme :

```
Saisissez votre nom:    Dupont
Saisissez votre prénom: Antoine

 Bonjour Antoine Dupont
```

À savoir

Le langage C++ a hérité du C la notion de chaîne sous la forme d'un tableau de charxe "chaîne:tableau de char (format C)" terminé par le caractère nul ainsi que d'un ensemble de fonctions permettant de manipuler ces chaînes. Ces fonctions se trouvent dans les en-têtes <string.h> et <cstring>. Lorsque c'est possible, il est préférable d'éviter les chaînes de style C et de choisir des chaînes string pour bénéficier du contrôle des types plus rigoureux du C++. cin accepte les arguments de type pointeur de caractères (char*xe "char*,

représentation d'une chaîne"), c'est pourquoi vous pouvez créer un tableau de caractères (ligne 6) et l'alimenter avec ce flux. Si vous saisissez Dupont, la variable nom va recevoir les caractères 'D', 'u', 'p', 'o', 'n', 't', suivis du caractère nul de fin '' que vous ne devez pas oublier de prendre en compte dans la taille du tableau. Pour enregistrer le prénom, nous utilisons cette fois le type string de la bibliothèque standard. Les chaînes standard présentent l'avantage de s'adapter à la taille des données que vous voulez y stocker. La déclaration de variable (ligne 7) est très simple, vous n'avez pas besoin de calculer la taille de cette dernière. N'oubliez pas que le flux cin lit tous les caractères jusqu'au premier espace rencontré. Si vous saisissez « Marie Charlotte » comme prénom, l'instruction de la ligne 14 n'enregistrera que « Marie » et « Charlotte » sera affiché avec l'instruction cout suivante. Vous pouvez résoudre ce problème avec la fonction getline() qui lit la ligne complète (voir exemple suivant). La ligne 16 est particulièrement intéressante. En C++, vous pouvez déclarer vos variables dans l'instruction qui les utilise. Vous obtenez ainsi une meilleure gestion de la mémoire et donc de meilleures performances. Cela limite aussi les erreurs dues à des variables non initialisées. La variable resultat est donc déclarée puis initialisée en concaténant deux chaînes littérales et deux objets string. L'opérateur de concaténation des objets string en C++ est +. La ligne suivante présente une autre forme de concaténation : on ajoute un retour à la ligne à la fin de la chaîne avec l'opérateur +=. Enfin, à la ligne 19, vous affichez la chaîne obtenue.

Astuce

La ligne 17 sert uniquement à illustrer l'opérateur += et le code n. Dans un programme standard, les lignes 17 et 19 seront regroupées en une seule :

```
cout << resultat << endl;
```

endl est expliqué dans la section consacrée aux manipulateurs de flux. Examinez maintenant le programme suivant. Nous lisons toujours les nom et prénom saisis au clavier mais en utilisant cette fois la fonction getline(). Code 2.4 : utilisation de getline()

```
1 : #include<iostream>
2 : #include<string>
3 : using namespace std;
4 :
5 : int main()
6 : {
7 :  string nom, prenom;   //le nom complet est stocké dans
8 :                        //2 variables de type string
9 :  cout << "Saisissez votre nom:t";
10:  getline (cin, nom);
11:  cout << "nSaisissez votre prénom:t";
12:  getline (cin, prenom);
13:
14:  cout << "Bonjour " << prenom << " " << nom <<endl;
15:
16:  string message = "Bienvenue";
17:
18:  unsigned int i;
19:  for (i=0; i<message.length(); i++) {
20:    cout << "Lettre " << i << "= " << message[i] << endl; }
21: }
```

Voici un exemple d'exécution de ce programme :

```
Saisissez votre nom:    Dupont de nemour
Saisissez votre prénom: Charles Antoine

 Bonjour Charles Antoine Dupont de nemour
Lettre 1= B
Lettre 2= i
Lettre 3= e
Lettre 4= n
Lettre 5= v
Lettre 6= e
Lettre 7= n
Lettre 8= u
Lettre 9= e
```

getline() prend en premier paramètre le flux d'entrée sur lequel elle doit lire la ligne, et en deuxième paramètre l'objet string dans lequel elle doit stocker cette ligne. Elle accepte un troisième paramètre pour indiquer le caractère séparateur de ligne. Ce paramètre est facultatif, car il dispose d'une valeur par défaut qui correspond au caractère de fin de ligne classique 'n'. La différence entre ce programme et le précédent est que celui-ci fonctionne quel que soit le nombre de mots composant les nom et prénom. Le mot clé endl ligne 19 insère un retour à la ligne dans le flux de sortie. L'affichage du message "Bienvenue" aux lignes 19 à 21 illustre un autre aspect des objets de type string : ils sont assimilés à des tableaux de caractères, ce qui permet d'accéder à chaque caractère d'une chaîne comme s'il s'agissait d'un tableau. Pour ce qui concerne la longueurxe "longueur de chaîne"xe "chaîne:longueur (length)", vous l'obtenez en la suffixant par .lengthxe "fonction:length"xe "length (fonction)"(). La valeur obtenue est de type unsigned int (ligne 19). De nombreuses autres fonctions ou méthodes déclarées dans <string> sont disponibles. Voici un échantillon de ces fonctions :

```
string s1 = "une chaîne de caractères";
string s2 = s1.substr(14, 23);  // s2 = "caractères"xe "fonction:substr"xe "substr (fonction)"
string s3 = s1.replace(14, 23, "type string");xe "fonction:replace"xe "replace:fonction"
   // s3 = "une chaîne de type string"
int n1 = s3.find("ne");                    // n1 = 1xe "fonction:find"xe "find (fonction)"
int n2 = s3.rfind("ne");                   // n2 = 8xe "fonction:rfind"xe "rfind (fonction)"
```

À savoir

En C, une chaîne de caractères est représentée en mémoire comme un tableau de char dont le dernier élément est le caractère nul noté \0. Il existe des méthodes de conversion entre le type string et les chaînes de C. Ces méthodes sont notamment utilisées quand un type de chaîne doit être transmis en argument à une fonction qui s'attend à recevoir l'autre type de chaîne. Si cs est une chaîne de C, alors string(cs) est un objet de type string. Inversement, si s est un type string, alors s.c_str() est une chaîne de C. Pour terminer avec les fonctions simples liées au flux d'entrée cin, get()xe "fonction:get"xe "get (fonction)" est l'équivalent de getline() pour un seul caractère. Voici un exemple d'utilisation de cette fonction :

Code 2.5 : lecture caractère par caractère avec get()

```
#include<iostream>
#include<string>
using namespace std;

int main()
{
  char car1, car2;

  cout << "Saisissez votre nom: ";
  cin.get(car1).get(car2);  // cin.get est appelée 2 fois
  cout << "Votre nom commence par: " << car1 ;
  cout << car2 << endl;     //endl purge le flux
  cin.ignore(80, 'n');
 //on ignore les caractères suivants (80 maxi) sur la ligne
}
```

L'exécution de ce code donne le résultat suivant :

```
Saisissez votre nom: Dupont
Votre nom commence par: Du
```

La fonction get() renvoie un objet de type iostream, comme le flux cin. Vous pouvez donc concaténer cette fonction comme illustré dans cet exemple. Examinons maintenant les fonctions associées au flux de sortie. Comme l'opérateur de réception << avec get() et getline(), l'opérateur d'émission se complète avec les fonctions put() et write()xe "fonction:put"xe "put (fonction)"xe "fonction:write"xe "write (fonction)". La première écrit un caractère dans le flux de sortie et peut être concaténée comme get(). L'instruction qui suit, par exemple, affiche "Bonjour" sur l'écran :

```
cout.put('B').put('o').put('n').put('j').put('o').put('u').put('r');
```

La fonction write() reçoit deux arguments : la chaîne et le nombre maximal de caractères à écrire. La fonction strlen() permet d'obtenir la longueur de la chaîne.

Code 2.6 : utilisation de put() et de write()

```
#include <iostream>
using namespace std;

int main() {

  char* chaine = "Bonjour les amis!";
  cout.write(chaine,7);    // affiche "Bonjour"
  cout.put('n');          //on écrit un retour à la ligne
  cout.write(chaine, strlen(chaine)) ; // affiche toute la chaîne
  return 0;
}
```

Résultat obtenu :

```
Bonjour
Bonjour les amis!
```

Le texte original de cette fiche pratique est extrait de «Tout sur le C++» (Christine EBERHARDT, Collection CommentCaMarche.net, Dunod, 2009)

‹ Précédent

- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

Suivant ›

Réalisé sous la direction de Jean-François PILLOU,
fondateur de CommentCaMarche.net.