Classe string pour la représentation des chaînes de caractères

C'est un fait étonnant, la majorité des traités d'algorithmie n'étudient pas les chaînes en tant que telles. La structure de données s'en rapprochant le plus reste le tableau pour lequel on a imaginé une grande quantité de problèmes et de solutions.

Le langage C est resté fidèle à cette approche et considère les chaînes comme des tableaux de caractères. Ses concepteurs ont fait deux choix importants : la longueur d'une chaîne est limitée à celle allouée pour le tableau, et le codage est celui des caractères du C, utilisant la table ASCII. Dans la mesure où il n'existe pas de moyen de déterminer la taille d'un tableau autrement qu'en utilisant une variable supplémentaire, les concepteurs du langage C ont imaginé de terminer leurs chaînes par un caractère spécial, de valeur nulle. Il est vrai que ce caractère n'a pas de fonction dans la table ASCII, mais les chaînes du C sont devenues très spécialisées, donc très loin de l'algorithmie générale.

L'auteur de C++, Bjarne Stroustrup, a souhaité pour son langage une compatibilité avec le langage C mais aussi une amélioration du codage prenant en compte différents formats de codage, ASCII ou non.

1. Représentation des chaînes dans la STL

Pour la STL, une chaîne est un ensemble ordonné de caractères. Une chaîne s'apparente donc fortement au **`vector`** , classe également présente dans la bibliothèque. Toutefois, la chaîne développe des accès et des traitements qui lui sont propres, soutenant ainsi mieux les algorithmes traduits en C++.

Les chaînes de la bibliothèque standard utilisent une classe de caractères pour s'affranchir du codage. La STL fournit le support pour les caractères ASCII (`char`) et pour les caractères étendus (`wchar_t`), mais on pourrait très bien envisager de développer d'autres formats destinés à des algorithmes à base de chaînes. Le génie génétique emploie des chaînes composées de caractères spécifiques, A, C, G, T. Le codage avec un `char` est donc très coûteux en termes d'espace, puisque deux bits suffisent à exprimer un tel vocabulaire, d'autant plus que les séquences de gènes peuvent concerner plusieurs

centaines de milliers de bases. On peut aussi spécifier des caractères adaptés à des alphabets non latins, pour lesquels la table ASCII est inefficace.

La classe `basic_string` utilise un vecteur (`vector`) pour ranger les caractères en mémoire. L'implémentation ainsi que les performances peuvent varier d'un environnement à l'autre, suivant la qualité de la programmation. Toutefois, la conséquence la plus intéressante de l'utilisation du vecteur est que la longueur des chaînes est devenue variable. Voilà une limitation du langage C enfin dépassée.

En contrepartie, l'accès aux caractères n'est pas aussi direct. Des méthodes spécifiques ont été ajoutées, la classe `vector` ne proposant pas le nécessaire. Une chaîne propose également des méthodes spécifiques pour travailler sur des intervalles de caractères (extraction de sous-chaînes, recherche), alors que le vecteur est plutôt destiné à l'accès individuel aux éléments qu'il contient.

Pour le programmeur-utilisateur de la bibliothèque standard, la partie chaîne de caractères se résume peut-être à la classe `string`, destinée à améliorer l'antique `char*`. Mais il y a aussi le matériel nécessaire pour aller plus loin.

2. Mode d'emploi de la classe string

La classe `string` est une spécialisation du modèle `basic_string` pour les caractères habituels, `char`. Les caractéristiques exposées par la suite sont également valides pour d'autres formats de chaînes issus de `basic_string`.

a. Fonctions de base

Une chaîne se construit de différentes manières, à partir d'une littérale de chaîne ou caractère par caractère. Lorsque la chaîne est créée, on cherche souvent à accéder à certains de ses caractères.

Constructeurs

Différents constructeurs sont disponibles pour initialiser une chaîne de type `string`. Une chaîne peut être initialisée à partir d'une chaîne C (`char*`), à partir d'une autre chaîne ou

d'un morceau de chaîne :

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

int main(int argc, char* argv[])
{
  string s1;  // chaîne vide
  string s2 = ""; // vide aussi

  string s3 = "Bonjour";
  string s4(s3);  // copie s3 dans s4

  // copie tous les caractères de s3 dans s5
  string s5(s3.begin(),s3.end());
  string s6(s3,0,s3.length());  // idem
  cout << s1 << " " << s2 << endl;
  cout << s3 << " " << s4 << endl;
  cout << s5 << " " << s6 << endl;

  return 0;
}
```

Les méthodes `begin()` et `end()` expriment des positions de sous-chaîne. Il ne s'agit pas de valeurs entières, mais de références à des caractères. Le constructeur utilisé pour `s6` est donc différent de celui employé pour `s5` qui fonctionne avec des indices de caractères.

Itérateurs

La classe `string` fournit deux itérateurs destinés à l'itération ordinaire et à l'itération inverse. Toutefois, les méthodes spécifiques de la classe `string` donnent de meilleures implémentations pour les algorithmes spécifiques aux chaînes.

Il est cependant possible d'employer ces itérateurs, `begin/end` (respectivement `rbegin`, `rend`) :

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

int main(int argc, char* argv[])
{
  string s="Bonjour la STL";
  string::iterator p=find(s.begin(),s.end(),'n');
  if(p==s.end())
    cout << "Il n'existe pas de caractère 'n' dans s" << endl;
  else
    cout << "Le caractère 'n' se trouve dans la chaîne : " << *p;
  return 0;
}
```

Accès aux éléments

L'opérateur d'index `[]` a été surchargé pour la classe `string`. On accède alors aux éléments `s[0]` à `s[s.length() -1]`. En dehors de ces plages, l'accès déclenche une exception **`out_of_range`**.

```
string s="Bonjour la STL";
cout << s[0] << ' ' << s[1] << ' ' <<
s[s.length()-1] << endl;
```

Attention, car l'équivalence entre tableau et pointeur n'est plus vérifiée, la classe `string` ayant défini plusieurs champs. Ainsi, `s` est différent de `&s[0]`.

b. Intégration dans le langage C++

La chaîne est un type singulier pour tout langage de programmation, aussi important que peut l'être l'entier ou le booléen. La classe `string` a été conçue dans le but de rendre son emploi le plus naturel possible, c'est-à-dire qu'elle doit s'intégrer aussi bien que `char*` au sein des programmes C++.

Affectations

Nous avons vu au chapitre sur la programmation orientée objet qu'affectation et constructeur sont des notions liées. La classe `string` possède évidemment un constructeur de copie, dont trois arguments sur quatre reçoivent des valeurs par défaut.

L'opérateur `=` a été redéfini pour rendre l'emploi de la classe le moins singulier possible :

```
string m;
 m = 'a';    // initialise m à partir de 'a'
 m = "Bonjour"; // initialise m à partir de char*
 m = s;
```

Attention toutefois de ne pas commettre d'erreur de sémantique liée à la confusion entre table ASCII et valeur entière :

```
m = 65L;    // erreur de sémantique
```

En effet, `m` serait initialisée avec le caractère de code ASCII 65, donc 'A', et non avec la chaîne "65".

Comparaisons

La comparaison de chaînes est essentielle pour implémenter les algorithmes. La classe `basic_string` supporte les comparaisons entre deux chaînes et entre une chaîne et un tableau de caractères. La méthode `compare()` retourne -1, 0 ou 1 comme le ferait la fonction `strcmp()`.

De plus, les opérateurs `==`, `!=`, `>`, `<`, `<=` et `>=` ont été surchargés pour comparer des chaînes.

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

int main(int argc, char* argv[])
{
 string a="Piano";
```

```
    string b="Pianissimo";

    cout << boolalpha;
    cout << a.compare(b) << endl;    // affiche 1
    cout << a.compare("Forte") << endl;  // affiche 1

    cout << (a<b) << endl;       // affiche true

    return 0;
}
```

Conversion avec le C

Il existe trois méthodes pour appliquer les fonctions de traitement des chaînes du C sur des instances de la classe `string` : `data()`, `c_str()` et `copy()`.

La méthode `data()` copie les caractères de la chaîne dans un tableau avant de retourner un pointeur vers cette zone (`const char*`). L'instance de la classe `string` a la charge de libérer de la mémoire le tableau, aussi le programmeur ne doit pas tenter de le faire lui-même. Si la chaîne est modifiée pendant la période d'exposition de la fonction `data()`, des caractères risquent d'être perdus :

```
string a="bonjour";
const char*d=a.data();
a[0]='B';
char c=d[0];  // erreur, toujours la valeur 'b'
```

La méthode `c_str()` ajoute un caractère `0` à la fin de la chaîne, faisant coïncider cette représentation avec les chaînes du langage C :

```
const char*b = a.c_str();
int l_a = strlen(b);
```

Enfin, la méthode `copy()` recopie le contenu de la chaîne dans un tampon dont le programmeur a la responsabilité de libération :

```
char*t=new char[a.length()+1];  // 0 terminal
 a.copy(t,a.length());
 t[a.length()]=0;  // 0 terminal
 // ...
 delete t;
```

c. Fonctions spécifiques aux chaînes

Les chaînes de caractères de la bibliothèque standard offrent des méthodes spécifiques à l'écriture de certains algorithmes. Il s'agit de fonctions de haut niveau, difficilement programmables en langage C pour certaines d'entre elles.

Insertion

L'opérateur `+=` a été surchargé pour l'ajout d'un caractère ou d'une chaîne à la fin d'une chaîne. Cet opérateur a la même fonction que la méthode `append()` qui offre, elle, un registre de possibilités plus varié.

La méthode `insert()` présente l'avantage d'intercaler de nouveaux caractères à l'endroit souhaité par le programmeur.

```
string s="Fort";
 s+="isimo";
 cout << "s=" << s << endl;  // affiche Fortisimo
 s.insert(5,"s");
 cout << "s=" << s << endl;  // affiche Fortissimo
```

Concaténation

L'opérateur `+` a été surchargé pour réunir plusieurs chaînes en une seule. Il concatène aussi bien des caractères que des chaînes :

```
string tonalite;
string ton="la ";
ton=ton+'b';
string maj="majeur";
```

```
tonalite=ton+maj;
cout << tonalite << endl; // affiche la b majeur
```

Recherche et remplacement

Il existe plusieurs méthodes pour rechercher des éléments à l'intérieur d'une chaîne :

`find`	Recherche d'une sous-chaîne.
`rfind`	Recherche en partant de la fin.
`find_first_of`	Recherche de caractères.
`find_last_of`	Recherche de caractères en partant de la fin.
`find_first_not_of`	Recherche d'un caractère absent dans l'argument.
`find_last_not_of`	Recherche d'un caractère absent dans l'argument en partant de la fin.

Toutes ces méthodes renvoient un `string::size_type`, assimilable à un entier.

```
string s="BEADGCF";
string::size_type i1=s.find("A");

cout << "i1=" << i1 << endl;  // affiche 2
```

Si la recherche n'aboutit pas, `find()` renvoie `npos`, position de caractère illégale. La méthode `find()` renvoie d'ailleurs un résultat non signé :

```
string::size_type i2=s.find("Z");
```

```
cout << "i2=" << i2 << endl;  // affiche un nombre très grand
if(i2==string::npos)
  cout << "recherche de Z infructueuse";
```

Il n'est pas rare de combiner recherche et remplacement. C'est précisément le rôle de la méthode `replace()` que de modifier une séquence de chaîne en la remplaçant par une autre séquence :

```
string ton="la bémol majeur";
string::size_type p=ton.find("maj");
ton.replace(p,6,"mineur");
cout << ton << endl;    // affiche la bémol mineur
```

Il faut remarquer que la séquence remplaçante peut avoir une longueur différente de la séquence recouverte. Il existe aussi une méthode `erase()` pour supprimer un certain nombre de caractères :

```
ton.erase(ton.find("bémol"),5);
cout << ton << endl;    // affiche la mineur
```

Extraction de sous-chaîne

La méthode `substr()` réalise une extraction d'une sous-chaîne. Il est entendu qu'elle renvoie un résultat de type `string`. Voici un exemple très simple d'utilisation de cette méthode :

```
cout << ton.substr(0,3).c_str() << endl; // affiche la
```