Classe string

(Tiré du livre de Deitel et Deitel, « Comment programmez en C++ », chap. 19)

INTRODUCTION

La classe string appartient au standard C++. Pour l'employer, incluez simplement le fichier d'entête suivant : #include <string>. Les définitions des modèles et toutes les fonctionnalités sont définies dans l'espace de nom std.

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

. . .
```

USAGE DE CONSTRUCTEURS

Il existe plusieurs constructeurs pour la classe string.

```
string c1("Allo");      //  Crée un objet de la classe string renfermant les
                        //  caractères inclus dans "Allo".

string c2(8, 'x');      //  Crée un objet de la classe string renfermant les
                        //  caractères "xxxxxxxx".

string c3;              //  Crée un objet de la classe string renfermant une chaîne
                        //  vide.

string c4 = "Mai";      //  Constructeur de copie.
```

Par contre, la classe string n'offre aucune conversion d'un type int ou char en string.

```
string erreur1 = 'c';
string erreur2 ( 'u' );
string erreur3 = 22;
string erreur4(8);
```

Erreur de syntaxe

PARALLÈLE AVEC LES CHAÎNES DE CARACTÈRES CHAR *

Un string ne représente pas un pointeur vers une chaîne de caractères mais un objet. Contrairement aux chaînes de caractères char * propres au C, les objets de la classe string ne se terminent pas nécessairement par un caractère null.

La longueur d'un objet de la classe string est stockée dans la classe string; elle peut être recouvrée avec la fonction membre length : Ex. : int n = c1.length();

Une chaîne de longueur 0, ne contenant aucun caractère, porte le nom de chaîne vide. Elle est écrite "". Contrairement aux variables numériques, les variables de type string sont toujours initialisées par défaut avec une chaîne vide : Ex. : string chaine;

L'opérateur d'indice [] peut être utilisé avec des objets de la classe string pour accéder à des caractères individuels. Les objets de la classe string possèdent un premier indice de 0 et un dernier égal à la longueur de l'objet – 1.

La plupart des fonctions membres d'une classe string prennent comme arguments un emplacement d'indice de départ et le nombre de caractères pour l'opération.

Si la quantité de caractères à traiter dépasse la longueur de l'objet de la classe string, un ajustement de la quantité est effectué pour ne pas déborder de la limite de la chaîne.

Caractéristiques des objets de la classe string

Fonction membre	**Fonctionnalité**
capacity	le nombre total de caractères pouvant être stockés sans devoir augmenter la capacité de mémoire de la chaîne.
max_size	la longueur de la plus grande chaîne pouvant être stockée dans un objet de la classe string.
size ou length	le nombre de caractères stockés dans une chaîne.
empty	retourne true si la chaîne est vide, false autrement.

Exemple :

```
#include <iostream>
#include <string>
using namespace std;
void main()
{
        string s("OUF");
        cout    << "capacite : "          << s.capacity()
                << "\ntaille max. : "      << s.max_size()
                << "\ntaille : "           << s.size()
                << "\nlongueur : "         << s.length()
                << "\nvide : "             << (s.empty() ? "true" : "false");

}
```

SORTIE :

```
capacite : 31
taille max. : 4294967293
taille : 3
longueur : 3
vide : false
```

****

Exemple de programme :

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

void main()
{
	string s1("cat"), s2, s3;		// Création de s1, s2 et s3.

	s2 = s1;				// s2 devient une copie de s1, mais n'est pas lié à s1.

	s3.assign(s1);				// Copie s1 dans s3 grâce à la
						// fonction membre assign.
						// s1 et s3 ne sont pas liés.

	cout	<< "s1: " << s1
		<< "\ns2: " << s2
		<< "\ns3: " << s3	<< "\n\n";

	s2[0] = s3[2] = 'r';			// Il n'y a pas de vérification de plage.

	cout	<< "Apres modification de s2 et s3:\n"
		<< "s1: "	<< s1
		<< "\ns2: "	<< s2	<< "\ns3: ";

	//	Fonction membre at() :	permet de sortir le contenu d'une chaîne un caractère
	//				à la fois (une vérification de plage est faite).

	int longueur = s3.length();
	for (int x = 0; x < longueur; ++x) cout << s3.at(x);

	//	Concaténation

	string s4(s1 + "acombes"), s5;		// Déclaration de s4 et s5.
```

```
                                        // Surcharge de l'opérateur + qui représente la
                                        // concaténation.
                                        // Attention! Une ou les deux chaînes entourant +
                                        // doivent être une variable de type string.

        //      Concaténation de s3 et "pette" grâce à la surcharge de +=.

        s3 += "pette";

        //      Fonction membre append :    permet de concaténer à l'objet courant la chaîne
        //                                  passée en paramètre.

        s1.append("atonie");

        //      Concaténer une sous-chaîne de s1 (les 4e et 5e éléments de s1) à la chaîne s5.
        //      La fonction size renvoie le nombre de caractères contenus dans s1.

        s5.append(s1, 3, s1.size());

        cout    << "\n\nAprès concatenation:\n"      << "s1: "        << s1
                << "\ns2: "     << s2   << "\ns3: "     << s3   << "\ns4: "
                << s4   << "\ns5: "     << s5   << endl;
}
```

SORTIE :

```
s1: cat
s2: cat
s3: cat

Apres modification de s2 et s3:
s1: cat
s2: rat
s3: car

Apres concatenation:
s1: catatonie
s2: rat
s3: carpette
s4: catacombes
s5: atonie
```

Comparaison d'objets de la classe string

```
#include <iostream>
#include <string>

using namespace std;

void main()
{
        string  s1("Test des fonctions de comparaison."),
                s2("Allo"),
                s3("compara"),
                z1(s2);

        //      Comparaison de s1 avec z1.
        //      Toutes les fonctions d'opérateurs surchargés (==, !=, <, >, <=, >=)
        //      renvoient des valeurs booléennes.

        if ( s1 == z1) cout << "s1 == z1\n";
                else    if (s1 > z1) cout << "s1 > z1\n";
                                else cout << "s1 < z1\n";

        //      Comparaison de s1 avec s2.
        //      La fonction membre compare vérifie s1 p/r à s2;
        //      elle retourne 0 si s1 et s2 sont équivalentes,
        //      un nombre positif si s1 > s2 et un nombre négatif
        //      si s1 < s2.

        int f = s1.compare(s2);
        if ( f == 0) cout << "s1.compare(s2) == 0\n";
                else    if (f > 0) cout << "s1.compare(s2) > 0\n";
                                else cout << "s1.compare(s2) < 0\n";

        //      Comparaison de s1 (éléments 22 à 28) avec s3 (éléments 0 à 6).
        //      Version surchargée de la fonction compare pour comparer
        //      des portions de s1 et de s3.
        //      Les paramètres de la fonction compare sont respectivement :
        //      indice de départ de s1, longueur de la portion de s1,
        //      s3, indice de départ de s3, longueur de la portion de s3.

        f = s1.compare(22, 7, s3, 0, 7);
        if ( f == 0) cout << "s1.compare(22, 7, s3, 0, 7) == 0\n";
                else    if (f > 0) cout << "s1.compare(22, 7, s3, 0, 7) > 0\n";
                                else cout << "s1.compare(22, 7, s3, 0, 7) < 0\n";
```

```
        //      Comparaison de z1 avec s2.
        //      Version surchargée de la fonction compare pour comparer
        //      une portion de z1 avec s2.
        //      Les paramètres de la fonction compare sont respectivement :
        //      indice de départ de z1, longueur de la portion de z1, s2.

        f = z1.compare(0, s2.size(), s2);
        if ( f == 0) cout << "z1.compare(0, s2.size(), s2) == 0\n";
              else     if (f > 0) cout << "z1.compare(0, s2.size(), s2) > 0\n";
                             else cout << "z1.compare(0, s2.size(), s2) < 0\n";

}
```

SORTIE :

```
s1 > z1
s1.compare(s2) > 0
s1.compare(22, 7, s3, 0, 7) == 0
s1.compare(0, s2.size(), s2) == 0
```

Extraction d'une sous-chaîne d'objets de la classe string

Il s'agit de la fonction membre substr où l'on retrouve 2 paramètres : l'indice de départ de la sous-chaîne et le nombre de caractères à extraire.

Ex. :
```
string s("L'avion a pris son envol. ");
cout << s.substr(2, 5) << endl;
```

SORTIE :

```
avion
```

Recherche de caractères dans une chaîne de caractères

```
#include <iostream>
#include <string>

using namespace std;

void main()
{
        //      Les 5 littéraux de chaînes sont concaténés en un seul.

        string s("Les valeurs des sous-arbres de gauche"
                        "\nsont inferieures a la valeur du"
                        "\nnoeud parent et les valeurs des"
                        "\nsous-arbres de droite sont superieures"
                        "\na la valeur du noeud parent");

        //      Trouver "arbres" aux emplacements 21 et 107.
        //
        //      Grâce à la fonction find, si la chaîne est repérée,
        //      son indice d'emplacement de départ est retourné;
        //      autrement, la valeur string::npos (constante public
        //      static définie dans la classe string) est renvoyée.
        //      Cette valeur est retournée par les fonctions de
        //      recherche pour indiquer qu'une sous-chaîne ou qu'un
        //      caractère est introuvé. La fonction rfind se distingue
        //      de la fonction find par le fait qu'elle effectue une
        //      recherche en commençant par la fin.

        cout << "String d'origine:\n"         << s
             << "\n\n(find) \"arbres\" se retrouve a : "
             << s.find( "arbres")
             << "\n(rfind) \"arbres\" se retrouve a : "
             << s.rfind( "arbres");

        //      Trouver 'p' de parent aux emplacements 76 et 162.
        //
        //      La fonction find_first_of recherche la 1e occurrence d'un
        //      caractère parmi "qpxz" à partir du début de s.
        //      La fonction find_last_of recherche la dernière occurrence
        //      d'un caractère parmi "qpxz" à partir de la fin de s.

        cout << "\n(find_first_of) premier caractere dans \"qpxz\":"
             << s.find_first_of("qpxz")
             << "\n(find_last_of) premier caractere dans \"qpxz\":"
             << s.find_last_of("qpxz");
```

```
        //      Trouver 'g' à l'emplacement 31.
        //
        //      La fonction find_first_not_of cherche le 1er caractère à
        //      partir du début de s qui n'est pas contenu dans "eLs vdaoubr".

        cout  << "\n(find_first_not_of) premier caractere\n"
              << "non contenu dans \"eLs vdaoubr\": "
              << s.find_first_not_of("eLs vdaoubr");

        //      Trouver 'n' à l'emplacement 166.
        //      La fonction find_last_not_of trouve le 1e caractère non
        //      contenu dans "eLs vdaoubr" à partir de la fin de s.

        cout  << "\n(find_last_not_of) premier caractere\n"
              << "non contenu dans \"eLs vdaoubr\": "
              << s.find_last_not_of("eLs vdaoubr")          << endl;


}
```

SORTIE :

```
String d’origine :
Les valeurs des sous-arbres de gauche
sont inferieures a la valeur du
nœud parent et les valeurs des
sous-arbres de droite sont superieures
a la valeur du nœud parent

(find)"arbres" se retrouve a : 21
(rfind)"arbres" se retrouve a : 107
(find_first_of) premier caractere dans "qpxz":76
(find_last_of) premier caractere dans "qpxz":162
(find_first_not_of) premier caractere
non contenu dans "eLs vdaoubr": 6
(find_last_not_of) premier caractere
non contenu dans "eLs vdaoubr": 167
```

Remplacement de caractères dans une chaîne de caractères

La fonction membre erase enlève tous les caractères à partir d'un emplacement jusqu'à la fin de la chaîne.

```
string s("coude") ;
cout << s.erase(3)   << endl ;
```

SORTIE :

cou

La fonction membre replace permet de remplacer une sous-chaîne par une autre.

Ex. 1 : Remplacer tous les espaces par un "_"

```
string s("coude a coude");
int x = s.find(" ");
while (x < string::npos)              // longueur maximale de la chaîne
{
      s.replace(x, 1, "_");           // 3 paramètres :      - indice de départ
                                                            - le # de caractères à remplacer
                                                            - la chaîne de remplacement
      x = s.find(" ", x + 1) ;        // renvoie string::npos lorsque la fin de s est atteinte.
}
cout << s << endl ;
```

SORTIE :

coude_a_coude

Ex. 2 : Remplacer chaque "_" et le caractère suivant par deux points

```
x = s.find("_");
while (x < string::npos)              // longueur maximale de la chaîne
{
      s.replace(x, 2, "xxxxx..yyy", 5, 2);// 5 paramètres : - indice de départ
                                                            - le # de caractères à remplacer
                                                            - la chaîne de remplacement
                                                            - la sous-chaîne ".."
      x = s.find("_", x + 1) ;        // renvoie string::npos lorsque la fin de s est atteinte.
}
cout << s << endl ;
```

SORTIE :

coude….oude

Insertion de caractères dans un objet de la classe string

Il s'agit de la fonction membre insert.

```
string      s1("debut fin"),
            s2("milieu "),
            s3("12345678"),
            s4("xx");

//      Insertion de "milieu " à l'emplacement 6

s1.insert(6, s2) ;
cout << s1 << endl ;

//      Insertion de "xx" à l'emplacement 3 dans s3.
//      3e paramètre : l'élément de départ de s4.
//      4e paramètre : le nombre de caractères de s4 à insérer.

s3.insert(3, s4, 0, string ::npos) ;
```

SORTIE :

```
debut milieu fin
123xx45678
```

Conversion en chaînes char * de C

Ex. :

```
#include <iostream>
#include <string>

using namespace std;

void main()
{
	string s("chaines");

	const char * ptr1 = 0;

	int longueur = s.length();

	char * ptr2 = new char[longueur + 1];		// Renferme le caractère NULL.

	ptr1 = s.data();			// Affecte la chaîne de s à ptr1 sans pointeur NULL.

	s.copy(ptr2, longueur, 0);	// Copie les caractères de s en mémoire.
	ptr2[longueur] = 0;		// Ajoute le caractère NULL.

	//	c_str() retourne un élément const char * terminé par un caractère NULL.

	cout	<< "s est " << s
		 << "\nconversion de s: " << s.c_str()
		 << "\nptr1 est ";

	for (int k = 0; k < longueur; ++k)
		cout << *(ptr1 + k);	// Utilisation de l'arithmétique de pointeurs

	cout << "\nptr2 est " << ptr2 << endl;
	delete [] ptr2;
}
```

SORTIE :

```
s est chaines
conversion de s: chaines
ptr1 est chaines
ptr2 est chaines
```

Gestion de flux de chaînes

Nous devons inclure les fichiers d'en-tête <sstream> et <iostream>. La classe istringstream supporte l'entrée d'un objet de la classe string alors que la classe ostringstream supporte la sortie vers un objet de la classe string. La fonction membre str de ostringstream renvoie une référence vers un objet de la classe string.

Exemple 1 - La classe ostringstream

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

void main()
{
        ostringstream Sortie;

        string s1("Sortie : ");
        int i = 35;

        Sortie << s1 << i;

        cout    << "Cela contient : "
                << Sortie.str() << endl;

        Sortie << "\nVitesse : 50 Km.";        // Ajout d'une nouvelle chaîne.
        cout    << "Cela contient maintenant : "
                << Sortie.str() << endl;
}
```

SORTIE :

```
Cela contient : Sortie : 35
Cela contient maintenant : Sortie : 35
Vitesse : 50 Km.
```

Exemple 2 - La classe istringstream

```
#include <iostream>
#include <string>
#include <sstream>

using namespace std;

void main()
{
	string s("Test d'entree 123 4.7 A");

	istringstream Entree(s);

	string s1, s2;
	int i;
	double d;
	char c;

	Entree >> s1 >> s2 >> i >> d >> c;

	cout	<< "s1: "	<< s1
		<< "\ns2: "	<< s2
		<< "\ni: "	<< i
		<< "\nd: "	<< d
		<< "\nc: "	<< c	<< endl;
}
```

SORTIE :

```
s1 : Test
s2 : d'entree
i: 123
d: 4.7
c: A
```

la classe string

(Tiré du livre de N. M. Josuttis, «The C++ Standard Library A Tutorial and Reference», chap.11)

Fonction membre	**Description**
Définitions de types	
string::size_type	Type de donnée de l'indice d'un caractère de la chaîne ou de la longueur de la chaîne.
string::iterator string::const_iterator string::reverse_iterator string::const_reverse_iterator static const string::size_type string::npos	La définition d'un pointeur généralisé.
static const string::size_type string::npos	Constante qui signifie « not found » ou « all remaining characters ». Voir plus loin.
Constructeurs	
string::string()	Constructeur par défaut. Crée une chaîne vide. **Ex . :** string s;
string::string(const string & str)	Crée une chaîne à l'aide d'un constructeur de copie. **Ex . :** string w = "abcd"; string s(w);
string::string(const char * str)	Crée une chaîne renfermant les caractères de str. **Ex . :** char * s = "abcd"; string w(s);
string::string(const char * str, string::size_type i)	Crée une chaîne en l'initialisant à la sous-chaîne de str renfermant les i premiers caractères de str. **Ex. :** char * s = "abcd"; string w(s, 2);

string::string(const char * str,
string::size_type i,
string::size_type j)
	Crée une chaîne en l'initialisant à la sous-chaîne de str renfermant jusqu'à j caractères de str à partir du caractère en position i.
	Ex. : char * s = "abcd";
	string::size_type i;
	string::size_type j;
	i = 2; j = 3;
	string w(s, i, j); // w égale "cd".

string::string(string::size_type i, char c)	Crée une chaîne renfermant i occurrences du caractère c.
	string::size_type i = 3;
	string s(i, 'h');

Destructeurs	
string::~string()	Libère l'espace mémoire.

Longueur et capacité de stockage	
string::size_type string::length() const	Le nombre de caractères de la chaîne
string::size_type string::size() const	Idem à length.
bool string::empty() const	Équivalent à size() == 0.
string::size_type string::max_size() const	Retourne le nombre maximum de caractères qu'un objet peut contenir.
string::size_type string::capacity() const	Retourne le nombre de caractères qu'un objet peut contenir sans réallocation de la mémoire.
void string::reserve() void string::reserve(string::size_type n)	Réserve de l'espace mémoire pour au moins n caractères. Si n < capacity() alors une demande de réduction de la capacité est faite. Si n < length() alors alors une demande de réduction de la capacité pour se rapprocher du nombre de caractères présents est faite. En l'absence de paramètre, une demande de réduction de la capacité est faite. La capacité n'est jamais réduite en bas du nombre de caractères présents. Finalement, une demande de réduction de la capacité n'est pas toujours réalisée.
void string::resize(string::size_type n) void string::resize(string::size_type n, char c)	Modifie le nombre de caractères de *this à n. Si le nombre de caractères != size() alors on ajoute ou on enlève des caractères à la fin de la chaîne *this. Si le nombre de caractères augmente, on ajoute la valeur de c le nombre de fois nécessaire; si c est absent, on ajoute '\0'.

Comparaisons	

bool operator comparaison
(const string & str1, const string & str2)
bool operator comparaison
(const string & str1, const char * str2)
bool operator comparaison
(const char * str1, const string & str2)
où comparaison désigne ==, !=, <, >, <= ou >=.
	Ex. : char * C = "abcdef";
	string str1 = "abcdef";
	string str2 = "ghi";
	if (C == str1)
	if (str1 < str2) cout << "OK";

int string::compare(const string & str) const
 	
int string::compare(const char * str) const
	Retourne 0 si (*this) == str.
	une valeur négative si (*this) < str,
	une valeur positive si (*this) > str.
	Ex. : char * C = "abcdef";
	string str1 = "abcdef";
	string str2 = "ghi";
	if (str1.compare(str2) < 0)
	if (str2.compare(C) > 0)
	cout << "OK";

int string::compare(string::size_type i,
string::size_type n,
const string & str) const
 	
int string::compare(string::size_type i,
string::size_type n,
const char * str) const
	Compare au plus n caractères de la chaîne *this à partir de la position i, avec la chaîne str.
	Ex. : char * C = "abcdef";
	string str1 = "abcdef";
	string str2 = "ghi";
	if (str1.compare(2, 3, str2) < 0)
	if (str2.compare(1, 2, C) > 0)
	cout << "OK";

int string::compare(string::size_type i,
string::size_type n,
const string & str,
string::size_type j,
string::size_type m) const
 	
int string::compare(string::size_type i,
string::size_type n,
const char * str,
string::size_type j,
string::size_type m) const
	Compare au plus n caractères de la chaîne *this à partir de la position i, avec au plus m caractères de la chaîne str à partir de la position j.
	Ex. :
	char * C = "abhief";
	string str1 = "abghi";
	string str2 = "ghijk";
	if (str1.compare(2, 3, str2, 0, 3) == 0)
	if (str2.compare(1, 2, C, 2, 2) == 0)
	cout << "OK";

Accès à un caractère	
char & string::operator [] (string::size_type i) const char & string::operator [] (string::size_type i) const	Retourne le caractère en position i (le premier caractère est en position 0). **Ex. :** string str = "abg"; str[str.length() - 1] = 'c'; cout << str << str[0] << str[1];
char & string::at (string::size_type i) const char& string::at(string::size_type i) const	Retourne le caractère en position i (le premier caractère est en position 0). **Ex. :** string str = "abg"; str.at(str.length() - 1) = 'c'; cout << str << str.at(0) << str.at(1);
Conversion en un tableau de caractères	
const string::char * c_str() const	Retourne l'objet *this comme un tableau de caractères auquel on a ajouté à la fin le caractère null '\0'. La valeur de retour appartient à l'objet *this; elle ne peut être modifiée, détruite ou l'espace mémoire libéré; elle existe tant et aussi longtemps que *this existe. **Ex. :** string str = "abg"; const char * c = str.c_str(); cout << str << " " << c;
const char * string::data() const	Retourne l'objet *this comme un tableau de caractères sans ajout du caractère null '\0'. La valeur de retour appartient à l'objet *this; elle ne peut être modifiée, détruite ou l'espace mémoire libéré; elle existe tant et aussi longtemps que *this existe. **Ex. :** string str = "abg"; const char * c = str.data(); cout << str << " " << c;

string::size_type string::copy
(char * buffer, string::size_type n) const
 	
string::size_type string::copy
(char * buffer, string::size_type n,
string::size_type i) const
	Copie dans buffer au plus n caractères de l'objet *this (à partir de la position i).
	Aucun caractère null n'est ajouté. Il faut s'assurer que l'espace mémoire nécessaire pour la variable buffer a été alloué.
	Ex. : string str = "abcdef";
	char * c = new char[4];
	str.copy(c, 3, 2);
	c[3] = '\0';
	cout << c;

Affectation	

string & string::operator = (const string & str)
 	
string & string::assign(const string & str)
	Affecte la valeur de str à l'objet courant. Retourne *this.
	Ex. : string str1 = "ghi";
	string str2 = str1;
	string str3;
	str3.assign(str2);

string & string::operator = (const char * str) string & string::assign(const char * str)	Affecte la valeur de str à l'objet courant. Cela n'inclut pas le caractère '\0'. Retourne *this. str ne doit pas être un pointeur NULL. **Ex. :** string str1 = "ghi"; string str2; str2.assign("abcdef");
string & string::operator = (char c)	Affecte la valeur de c à l'objet courant. Retourne *this. (*this).length() == 1. **Ex. :** string str2; str2 = 'b';
string & string::assign(string::size_type n, char c)	Affecte n occurrences de c à l'objet courant. Retourne *this. **Ex. :** string str; str.assign(5, 'a');

string & string::assign(const string & str,
string::size_type i,
string::size_type n)
	Affecte à l'objet courant au plus n caractères de str à partir de la position i. Retourne *this.
	Ex. : string str = "abcdefghi";
	string str1;
	str1.assign(str, 3, 3);
	cout << str1 << endl; // Affiche "def".

string & string::assign(const char * str,
string::size_type n)
	Affecte à l'objet courant les n premiers caractères de str. Retourne *this.
	Ex. : string str;
	str.assign("abcdefgh", 3);
	cout << str << endl; // Affiche "abc".

void string::swap(string & str) void swap(string & str1, string & str2)	Échange les valeurs de *this et str ou encore les valeurs de str1 et str2.

Concaténation de caractères	

string & string::operator += (const string & str)
 	
string & string::append(const string & str)
	Concatène les caractères de str à la fin de la chaîne courante *this. Retourne *this.
	Ex. : string str1 = "ab";
	string str2 = "cde";
	str1 += str2;
	str2.append(str1);

string & string::operator += (const char * str) string & string::append(const char * str)	Concatène les caractères de str à la fin de la chaîne courante *this. Retourne *this. str ne doit pas être un pointeur NULL. **Ex. :** string str1 = "ab"; string str2 = "cde"; str1 += "cde"; str2.append("abcde");

string & string::append(const char * str,
string::size_type n)
	Concatène les n premiers caractères de str à la fin de la chaîne courante *this. Retourne *this. str ne doit pas être un pointeur NULL.
	Ex. : const char * str = "abcdef";
	string str1 = "def";
	str1.append(str, 3);
	cout << str1; // Affiche "defabc".

string & string::append(const string & str,
string::size_type i,
string::size_type n)
	Concatène à la fin de la chaîne courante *this au plus n caractères de str à partir de la position i. Retourne *this.
	Ex. : string str = "abcdefgh";
	string str1;
	str1.append(str, 2, 3);
	cout << str1 << endl; // Affiche "cde".

string & string::append(string::size_type n,
char c)
	Concatène à la fin de la chaîne courante *this n occurrences de la valeur de c. Retourne *this.
	Ex. : string str = "abc";
	string str1 = "***";
	str.append(3, '*');
	cout << str1 + str << endl;
	// Affiche "***abc***".

string & string::operator += (char c) void string::push_back(char c)	Concatène à la fin de la chaîne courante *this la valeur de c. Retourne *this dans le premier cas.

string operator + (const string & str1, const string & str2) string operator + (const string & str1, const char * str2) string operator + (const char * str1, const string & str2) string operator + (const string & str, char c) string operator + (char c, const string & str)	Retourne la concaténation des 2 chaînes de caractères.

Recherche de caractères	

string::size_type string::find(char c) const
 	
string::size_type string::find
(char c,
string::size_type i) const
 	
string::size_type string::rfind(char c) const
 	
string::size_type string::rfind
(char c,
string::size_type i) const
	Retourne l'indice de la première occurrence de c rencontrée dans *this à partir du début.
	Retourne l'indice de la première occurrence de c rencontrée dans *this en se déplaçant de la position i vers la droite.
	
	Retourne l'indice de la première occurrence de c rencontrée dans *this en se déplaçant vers la gauche à partir de la fin.
	Retourne l'indice de la première occurrence de c rencontrée dans *this en se déplaçant vers la gauche à partir de la position i.
	
	S'il n'existe aucune occurrence de c dans *this, alors string::npos est retourné.
	Ex. :
	string s("abcdef");
	if (s.find('g') == string::npos) cout << "OK";

string::size_type string::find (const string & str) const string::size_type string::find (const string & str, string::size_type i) const string::size_type string::rfind (const string & str) const string::size_type string::rfind (const string & str, string::size_type i) const	Idem à la situation précédente à l'exception qu'une sous-chaîne str est recherchée à la place d'un caractère c. Ces fonctions retournent la position du premier caractère de str dans *this à l'endroit où la sous-chaîne str est trouvée. S'il n'existe aucune occurrence de str dans *this, alors string::npos est retourné.

string::size_type string::find (const char * str) const string::size_type string::find (const char * str, string::size_type i) const string::size_type string::rfind (const char * str) const string::size_type string::rfind (const char * str, string::size_type i) const	Idem à la situation précédente à l'exception qu'une sous-chaîne str de type char * est recherchée à la place d'un objet de la classe string.
string::size_type string::find_first_of (const string & str) const string::size_type string:: find_first_of (const string & str, string::size_type i) const string::size_type string::find_first_not_of (const string & str) const string::size_type string:: find_first_not_of (const string & str, string::size_type i) const	Retourne la position du premier caractère rencontré dans *this faisant partie de la sous-chaîne de str débutant à la position i ou au début de str. Retourne la position du premier caractère rencontré dans *this ne faisant pas partie de la sous-chaîne de str débutant à la position i ou au début de str. **Note :** S'il n'existe aucun caractère dans *this répondant à ces exigences, alors string::npos est retourné.
string::size_type string::find_first_of (const char * str) const string::size_type string:: find_first_of (const char * str, string::size_type i) const string::size_type string::find_first_not_of (const char * str) const string::size_type string:: find_first_not_of (const char * str, string::size_type i) const	Idem au cas précédent à l'exception que le 1^{e} paramètre est de type const char *.

string::size_type string::find_first_of (char c) const string::size_type string:: find_first_of (char c, string::size_type i) const string::size_type string::find_first_not_of (char c) const string::size_type string:: find_first_not_of (char c, string::size_type i) const	Retourne la position du premier caractère c rencontré dans *this à partir de la position i.
string::size_type string::find_last_of (const string & str) const string::size_type string:: find_last_of (const string & str, string::size_type i) const string::size_type string::find_last_not_of (const string & str) const string::size_type string:: find_last_not_of (const string & str, string::size_type i) const	Retourne la position du dernier caractère rencontré dans *this faisant partie de la sous-chaîne de str débutant à la position i ou au début de str. Retourne la position du dernier caractère rencontré dans *this ne faisant pas partie de la sous-chaîne de str débutant à la position i ou au début de str. **Note :** S'il n'existe aucun caractère dans *this répondant à ces exigences, alors string::npos est retourné.
string::size_type string::find_last_of (const char * str) const string::size_type string:: find_last_of (const char * str, string::size_type i) const string::size_type string::find_last_not_of (const char * str) const string::size_type string:: find_last_not_of (const char * str, string::size_type i) const	Idem au cas précédent à l'exception que le 1^{e} paramètre est de type const char *.

string::size_type string::find_last_of (char c) const string::size_type string:: find_last_of (char c, string::size_type i) const string::size_type string::find_last_not_of (char c) const string::size_type string:: find_last_not_of (char c, string::size_type i) const	Retourne la position du dernier caractère c rencontré dans *this à partir de la position i.
etc.	

Insertion de caractères	
string & string::insert(string::size_type i, const string & str)	Insère les caractères de str dans *this à partir de la position i. Retourne *this.
string & string::insert(string::size_type i, const string & str, string::size_type j, string::size_type n)	Insère au plus n caractères de str à partir de la position j dans *this à partir de la position i. Retourne *this.
string & string::insert(string::size_type i, const char * str)	Insère les caractères de str dans *this à partir de la position i. Retourne *this.
string & string::insert(string::size_type i, const char * str, string::size_type n)	Insère les n premiers caractères de str dans *this à partir de la position i. Retourne *this.

string & string::insert(string::size_type i,
string::size_type n,
char c)
 	
void string::insert(iterator pos,
string::size_type n,
char c)
	On insère n occurrences du caractère c dans *this soit, à partir de la position i, soit, avant le caractère en position pos. Retourne *this dans le premier cas.
	
	Ambiguïté possible : 0 comme 1e argument
	
	std ::string s;
	. . .
	s.insert(0, 1, ‘ ‘); // erreur.
	s.insert((std::string::size_type) 0, 1, ‘ ‘); // OK

iterator string::insert(iterator pos, char c)	Insère une copie du caractère c dans *this avant le caractère en position pos. Retourne la position du caractère inséré.
void string::insert(iterator pos, InputIterator debut, InputIterator fin)	Insère dans *this les caractères dans les positions [debut, fin) avant le caractère en position pos.

Remplacement de caractères	
string & string ::replace(string::size_type i, string::size_type n, const string & str) string & string::replace(iterator debut, iterator fin, const string & str)	Remplace au plus n caractères de *this à partir de la position i par les caractères de la chaîne str. Retourne *this. Remplace les caractères dans les positions [debut, fin) de *this par les caractères de la chaîne str. Retourne *this.
string & string::replace(string::size_type i, string::size_type n, const string & str, string::size_type j, string::size_type m)	Remplace au plus n caractères de *this à partir de la position i par au plus m caractères de la chaîne str à partir de la position j. Retourne *this.
string & string::replace(string::size_type i, string::size_type n, const char * str) string & string::replace(iterator debut, iterator fin, const char * str)	Remplace au plus n caractères de *this à partir de la position i par les caractères de la chaîne str. Retourne *this. str ne doit pas être un pointeur NULL. Remplace les caractères dans les positions [debut, fin) de *this par les caractères de la chaîne str. Retourne *this. str ne doit pas être un pointeur NULL.

string & string::replace(string::size_type i,
string::size_type n,
const char * str,
string::size_type m)
 	
string & string::replace(iterator debut,
iterator fin,
const char * str,
string::size_type m)
	Remplace au plus n caractères de *this à partir de la position i par les m premiers caractères de la chaîne str. Retourne *this. str doit avoir au moins m caractères.
	
	Remplace les caractères dans les positions [debut, fin) de *this par les m premiers caractères de la chaîne str. Retourne *this. str doit avoir au moins m caractères.
	string str = "abcdeghi";
	str.replace(2, 3, "-*-*", 3);
	cout << str << endl;
	// Affiche "ab-*-ghi".

string & string::replace(string::size_type i, string::size_type n, string::size_type m, char c) string & string::replace(iterator debut, iterator fin, string::size_type m, char c)	Remplace au plus n caractères de *this à partir de la position i par m occurrences de la valeur de c. Retourne *this. Remplace les caractères dans les positions [debut, fin) de *this par m occurrences de la valeur de c. Retourne *this.

string & string::replace(iterator debut, iterator fin, InputIterator pos_deb, InputIterator pos_fin)	Remplace les caractères dans les positions [debut, fin) de *this par ceux dans les positions [pos_deb, pos_fin). Retourne *this.

Effacer des caractères	
void string ::clear() string & string ::erase()	Enlève tous les caractères de *this. *this est vide après l'appel. Retourne *this dans le deuxième cas.
string &string ::erase(string::size_type i) string & string :: erase(string::size_type i, string::size_type n)	Enlève au plus n caractères de *this à partir de la position i. Si n est absent, tous les caractères de la position i jusqu'à la fin sont enlevés. Retourne *this.

iterator string ::erase(iterator i)
 	
iterator string ::erase(iterator debut,
iterator fin)
	Enlève de *this le caractère en position i ou encore les caractères dans les positions [debut, fin). Retourne la position du caractère qui suit le dernier caractère enlevé.
	Ex. : string str = "abcdeghi";
	str.erase(3, 2);
	cout << str << endl;
	// Affiche "abcghi".

Fonctions d'entrées / sorties	
ostream & operator << (ostream & strm, const string & str)	Écrire les caractères de str dans le flux strm.
istream & operator >> (istream & strm, string & str)	Lit les caractères du mot suivant dans str.
istream & getline(istream & strm, string & str) istream & getline(istream & strm, string & str, char c)	Lit les caractères de la ligne suivante de strm dans str. La lecture prend fin lorsque le caractère dans c ou le caractère de fin de ligne est rencontré.

Sous-chaînes de caractères	
string string::substr() const string string::substr(string::size_type i) const string string::substr(string::size_type i, string::size_type n) const	Retourne une sous-chaîne de *this d'au plus n caractères débutant à la position i. Si n est absent, tous les caractères de la position i jusqu'à la fin sont retournés. Si i et n sont absents, une copie de *this est retournée.
Positionnement d'un itérateur	
iterator string::begin() const_iterator string::begin() const	Retourne la position de début de chaîne. Si la chaîne est vide, cet appel est équivalent à end().
iterator string::end() const_iterator string::end() const	Retourne la position de fin de chaîne i.e. la position après le dernier caractère. (*s).end() n'est pas défini. Si la chaîne est vide, cet appel est équivalent à begin().
reverse_iterator string::rbegin() const_reverse_iterator string::rbegin() const	Retourne la position du dernier caractère de la chaîne. Si la chaîne est vide, cet appel est équivalent à rend().
reverse_iterator string::rend() const_reverse_iterator string::rend() const	Retourne la position avant le premier caractère de la chaîne. (*s).rend() n'est pas défini. Si la chaîne est vide, cet appel est équivalent à rbegin().

ooooooooooooooooooooooooooooooooooooooooooooooooo