Strings

C++ proporciona una alternativa simple y segura de utilizar cadenas de texto. La clase `string` es parte del `namespace std`.

> Más info en: http://www.cplusplus.com/reference/string/string/

Declaración de strings

```
using namespace std;
// ...
string cadena;
```

También se puede definir un valor inicial utilizando el constructor de la clase:

```
using namespace std;
// ...
string cadena("Cadena de ejemplo");
// o
string cadena2 = "Esta es otra cadena de ejemplo";
```

Las operaciones de E/S (entrada/salida) también son soportadas:

```
cin >> cadena;
cout << "La cadena ingresada es: " << cadena << endl;
```

Tomar en cuenta que si se necesita leer una cadena compuesta **por múltiples palabras** o se desea leer **una línea a la vez**, se debe utilizar la función `getline`:

```
getline(cin, cadena, '\n');
```

Concatenación o unión de strings

Los strings también pueden ser asignados unos con otros o ser combinados utilizando el operador `+`

```
string cadena1 = "Hola";
string cadena2 = " Mundo!";
string cadena3 = cadena1 + cadena2;

// Imprime Hola Mundo!
cout << cadena3 << endl;
```

Naturalmente, el operador `+=` también está definido. La unión de string funcionará siempre y cuando al menos uno de las cadenas de texto sea un `string` de C++. Las otras cadena pueden ser `string` o `char*`.

Comparación de strings

Una de las partes más confusas de usar `char*s` como cadenas de texto es que las comparaciones son complejas, ya que requieren el uso de funciones de comparación especiales. El uso de los operadores de relacionales (`==` , `!=` , etc.) no sirven en esos casos. Afortunadamente, con los strings de C++, todos los operadores relacionales funcionan

debidamente.

```
string password;

getline(cin, password, '\n');
if(password == "foobar")
{
    cout << "Acceso permitido";
}
```

Largo de strings y acceso a elementos individuales

Para obtener el largo de una cadena, *en términos de bytes*, de texto se pueden utilizar las funciones `length()` o `size()`, ambas miembros de la clase `string`.

```
size_t size() const;
size_t length() const;
```

```
string cadena = "Hola mundo"; // Cadena de largo 10
int largo = cadena.length(); // También se puede utilizar size();
```

El valor retornado por las funciones `length()` o `size()` podría ser diferente al actual número de caracteres **codificados** de la cadena de texto. Esto aplica para las secuencias de caracteres multi-byte, como UTF-8.

```
string cadena = "áéíóú";
cout << cadena.size() << endl; // Imprime 10

cadena = "aeiou";
cout << cadena.size() << endl; // Imprime 5
```

Los *strings*, al igual que los *chars**, pueden ser indexados de forma numérica. Por ejemplo, se podría iterar sobre todos los caracteres de un `string` indexándolo a través de números, como si se tratara de un array.

Al iterar sobre strings es muy importante utilizar alguna de las funciones `length()` o `size()` para comprobar el final de la cadena de texto, ya que un string de C++ no se garantiza que termine con el caracter nulo (`'\0'`).

```
for(int i = 0; i < cadena.length(); i++)
{
    cout << cadena[i];
}
```

También es posible obtener la **referencia** de un caracter con la función `at()`, miembro de la clase `string`. El valor devuelto por esta función puede ser `char*` o `const char*`.

```
char& at (size_t pos);
const char& at (size_t pos) const;
```

```
for (int i = 0; i < cadena.length(); ++i)
{
  cout << str.at(i);
}
```

Por otro lado, los strings son secuencias, al igual que cualquier otro contenedor STL, por lo tanto se pueden usar iteradores para recorrer el contenido del string.

```
string::iterator iterador;
for(iterador = cadena.begin(); iterador != cadena.end(); iterador++)
{
    cout << * iterador;
}
```

Nótese que `cadena.end()` está fuera del término del string, por lo tanto no debemos imprimirlo. `cadena.begin()` representa el primer caracter del string.

```
string::reverse_iterator rit
for (rit = str.rbegin(); rit != str.rend(); rit++)
{
    cout << * rit;
}
```

> Es muy importante saber que usar iteradores sobre strings que son **modificados** podría generar resultados inesperados, por ende se sugiere que sólo se usen cuando estemos seguros que el string no sufrirá modificaciones a lo largo de la ejecución.

Búsqueda y sub-cadenas

La clase string soporta búsqueda simple y extracción de cadenas a través de las funciones `find()`, `rfind()`, and `substr()`.

Función `find()` y `rfind()`

Esta función recibe como parámetros el patrón o cadena de búsqueda y la posición desde donde se desea comenzar a buscar. Retorna la posición de la primera ocurrencia de la cadena buscada ó, el valor especial `string::npos`, si la cadena no fue encontrada.

```
int find(string& str, site_t pos);
```

```
string cadena = "este es un ejemplo...";
int i = 0;
int apariciones = 0;
for(i = cadena.find("es", 0); i != string::npos; i = cadena.find("es", i))
{
    apariciones++;
    i++;  // Se mueve a la siguiente posición para evitar encontar la misma palabra
}
cout << "Apariciones: " << apariciones << endl;
```

De forma similar, se puede utilizar la función `rfind()` exactamente de la misma manera, pero en vez de comenzar buscando desde el inicio, se haría desde el final del string. Tomar en cuenta que la coincidencia de palabras se sigue haciendo de izquierda a derecha.

¿Cómo quedaría el ejemplo anterior utilizando la `rfind()`?

Función `substr()`

Esta función permite obtener una parte de una cadena de texto como un nuevo string, comenzando desde una `posición` dada hasta un `largo` de caracteres.

```
string substr(size_t pos = 0, size_t len = npos);
```

```
string cadena = "abcdefghijklmnop";
string primeras_letras = cadena.substr(0, 10);
cout << "Las primeras 10 letras del alfabeto son: " << primeras_letras << endl;
```

Modificando strings a través de separaciones o eliminaciones

Es posible modificar strings ya se apara remover parte de él o agregar nuevo contenido.

```
string& erase(size_t pos = 0, size_t len = npos);
```

La función `erase()` es similar a la función `substr()` en cuanto a su prototipo; toma la posición inicial y la cantidad de caracteres a remover. La posición, como siempre, comienza desde 0 en adelante.

```
string cadena = "elimina aaa";
cadena.erase(8, 3); // elimina aaa
```

Para eliminar el contenido completo de un string utilizamos:

```
cadena.erase(0, cadena.length());
cout << cadena.size() << endl; // Imprime 0
```

También es posible utilizar la función `clear()`:

```
void clear();
```

```
cadena.clear();
cout << cadena.size() << endl; // Imprime 0
```

Por otro lado, es posible dividir un string dentro de si mismo. La función `insert` acepta la posición y un string e inserta esa cadena de texto comenzando en la posición dada:

```
string cadena = "ade";
cadena.insert(1, "bc");
// cadena ahora es "abcde"
cout << cadena << endl;
```

Verificando cadenas vacías

```
bool empty() const;
```

La función `empty()` verifica si la cadena de texto tiene largo 0.

```
string cadena = "";
if(cadena.empty()) {
  cout << "La cadena está vacía" << endl;
}
```

Recuperando la cadena estilo C

En algunas ocasiones puede ser de utilidad obtener la representación `char*` de un `string`. Este puede ser necesario en los casos que se necesite utililzar alguna función de la librería estándar de C que acepte como parámetro una variable del tipo `char*`. La función `c_str()`, miembro de la clase `string`, retorna la cadena de texto con el tipo de dato `char*`.

```
const char* c_str();
```

Es importante señalar que el valor retornado es un valor constante, es decir, es de sólo lectura y no puede ser modificado. De ser necesario modificarlo se deberá copiar su contenido a otra variable utilizando la función `strcpy`.

```
string cadena = "Hola Mundo!";
const char* cadena_c = cadena.c_str();

cout << cadena_c << endl;
cout << strlen(cadena_c));
```