Programación II

Guía 1- Primeros ejercicios C++

2011

Strings

Objetivo de la Guía

Uso de clases del lenguaje - Strings

Bibliografía

http://cppreference.com y *http://www.cplusplus.com/*

La clase `String`

```
1.#include <string>
2.#include <iostream>
3.using namespace std;
4.int main(){
5.    string s1,s2;
6.    string s3("patotero");
7.    cout <<" Ingrese dos string's ..\n";
8.    cin>>s1>>s2;
9.    cout <<"Ingreso "<<s1<<" y "<< s2<<endl;
10.   int len = s3.length();
11.   cout <<"s3 vale "<<s3<<endl;
12.   s2=s3.substr(0,len-4);
13.   cout <<"Ahora s2 (s3.substr(0,len-4)) es "<<s2<<endl;
14.   }
```

En el ejemplo se ve el uso de *`strings`* en C++ . En C un *`string`* es un arreglo de caracteres terminado por el carácter especial `00` (que se escribe como `\0`). Muchos programas utilizan aún este formato. Para acceder al formato "viejo" de *strings* y a las funciones asociadas se debe incluir *`cstring`*. Sin embargo, las ventajas de la clase *`string`* son muchas.

En la línea 5 se declaran 2 *strings* sin inicializarlas, en la 6 se declara un *string* con un valor inicial. Estas declaraciones parecen pero no son iguales a las declaraciones de tipos básicos (¿Porque?).

Sucede que esta clase de datos es muy usada, por lo tanto "alguien" (en este caso los diseñadores de C++) decidió incluirla entre las clases básicas del sistema. Además dejó abierta la posibilidad de que el usuario (o sea Ud.) agregue sus propias clases de datos, junto con las rutinas que operan sobre ella. Estas clases de datos se conocen como **Tipos Abstractos de Datos** (**Abstract Data Types** o ADT**)**.

Abstract Data Type: Una colección de datos bien definida y las operaciones que pueden hacerse sobre esta colección.

La noción de ADT es central a la materia y a la Programación Orientada a Objetos. En C++

estas construcciones se llaman `class` (clases) y a las variables cuyo tipo es una `class`, se las llama "instancias de la clase" u *objetos*. (Para la Programación Orientada a Objetos faltan otros ingredientes: herencia y polimorfismo)

En el caso de `strings`, en la línea 10 puede verse el uso de una de las operaciones previamente definidas `s3.length()` que es una función `int` que devuelve la longitud del `string`. No es una función normal, ya que no tiene parámetro, pero sin embargo actúa sobre `s3` (devuelve su longitud). A este tipo de funciones, que deben ligarse a un objeto, se le llaman *métodos*.

Los métodos ligados a `string` se encuentran definidos en el directorio `...include/.../string` (el nombre no se da en forma exacta ya que varía de UNIX a UNIX). El formato que tiene es complejo ya que usa la facilidad de *templates*, aún no desarrollada en clase.

Métodos de la clase `String`

Constructores (indique que hacen)

`string s`	Crea un *string* vacío
`string s (string s1)`	
`string (char c,int l)`	

Operadores

==, <, <=, !=, > y >=	
Asignación =	
Concatenación +	

Métodos mas usuales:

Método	Prámetros	Descripción
`string& append ()`	`string s`	`Coloca a s al final del string actual. Devuelve un puntero al string actual.`
`string& append ()`	`string s` `int l`	
`int find () (1) (2)`	`string s`	
`int find () (1) (2)`	`char c`	
`int rfind () (1) (2)`	`string s`	
`int find_first_of() (1) (2)`	`string str`	
`int find_first_not_of() (1) (2)`	`string str`	
`string& insert() (2)`	`int ind,` `string str`	

Método	Prámetros	Descripción
string& replace ((1) (2)	int ind, int l, string s)	
int length()		
int size()		
string substr()	int len	
string substr()	int ind, int len	
void swap ()	string s	

Algunos de estos métodos tienen variantes admitiendo: un *índice* (1) para indicar desde donde operar y una *longitud* (2) que indica cuantos caracteres tomar.

Como ejemplo, modifique el siguiente programa hasta obtener los resultados para completar la tabla anterior:

```
1. // Uso del find
2.
3. #include <string>
4. #include <iostream>
5. using namespace std;
6.
7. int main(){
8.   string a ("un string y otro string unidos");
9.      cout<<"string a=<"<<a<<">"<<endl;
10.
   cout<<"a.find(\"unamonos\",0,2)="<<a.find("unamonos",0,2)<<endl
   ;
11.
   cout<<"a.find(\"unamonos\",5,2)="<<a.find("unamonos",5,2)<<endl
   ;
12.
   cout<<"a.find(\"unamonos\",5,3)="<<a.find("unamonos",5,3)<<endl
   ;
13.     // en realidad deberiamos chequear siempre si el find tuvo
   exito
14.     // por ejemplo:
15.     cout<<"a.find(\"unamonos\",5,3)=";
16.     if (a.find("unamonos",5,3)!=string::npos)
   cout<<a.find("unamonos",5,3)<<endl;
17.          else cout <<"no se encuentra"<<endl;
18.}
```

En la última instrucción del programa anterior se usó una constante definida en la clase *string.* Se trapa de la constante `npos` . Esta constante (*null position)* se usa para indicar que no se encontró lo buscado en el *string.* Para accederla se debe indicar que se trata de `npos` que pertenece a la clase *string* usando la siguiente nomenclatura: `string::npos` . El operador `::` se llama operador de ambiente (*scope).*

Ejercicios:

1. En Jerigonzo, una palabra se forma repitiendo la vocal antecedida por una "p" en cada sílaba. Por ejemplo, *tomate* => *to**po**ma**pa**te**pe* **,** *ruidoso* => *ru**pu**i**pi**do**po**so**po.* . Escriba una función que convierta a Jerigonzo y un programa que la pruebe.
2. Escriba la función para pasar de Jerigonzo a español.
3. La gente de habla inglesa tiene su propia versión de jerigonzo, se llama *piglatin.* En *piglatin,* la palabra *piglatin* se escribe *iglatinpay.* La Internet le va a ayudar. (Hay un Google en *piglatin* http://www.google.com/intl/xx-piglatin/ .
4. También puede ayudar a proponer alguna *Cifra Cesar* (Código de criptografía), y a hacer las rutinas necesarias.
5. Pruebe lo anterior con *Cifras Vignere.*
6. Haga una versión de *translate* (tr) de UNIX.
7. Haga un programa que cuente la cantidad de caracteres, palabras y líneas de un archivo (parecido al *wc* de UNIX)
8. Dada una grilla de *m* x *n (m<=n<=50)* y una palabra, su programa deberá indicar a partir de qué coordenada se encuentra esa palabra (si no está devuelve un (-1,-1)) y en que sentido debe leerse (cuatro sentidos posibles: si u=arriba, d=abajo, l=izquierda, r=derecha; los sentidos son u-d, d-u, l-r, r-l).
9. Dados dos strings *a* y *b* su programa deberá encontrar el *string* mas largo *x* tal que una permutación de *x* sea una subsecuencia de *a* y una permutación de *x* sea una subsecuencia de *b.* Permita leer mas de un caso de prueba.
   Ej: (oficina ,medicinal) x=icina
   (colegio, gelatina) x=elg.
10. Una busca en *google* acerca de *stringology* le acercará mas información acerca de este fascinante tema. ¿Aplicaciones? *Pattern matching* en *Computational Biology.*
11. Escriba un programa que pase un número entre 1 y 3000 ingresado en formato decimal a romano y otro que pase de romano a decimal. (ambos números son strings).