UNIVERSIDAD AUTONOMA
DE ENTRE RIOS

Facultad de Ciencia y Tecnología
Sede Oro Verde

Licenciatura en Sistemas
de Información

Programación Orientada a Objetos 2012
Unidad 3: cstrings

Unidad 3
cstrings

Resumen de Conceptos

Muchos de los datos que empleamos en programas deben almacenarse usando esta estructura: apellidos, nombres, direcciones, denominaciones, códigos, etc.; todos estos ejemplos corresponden a cadenas de caracteres o strings.

C++ posee una biblioteca estándar heredada de C; en ella encontraremos varias funciones para procesar cadenas de caracteres o strings al estilo de lenguaje C. También existe en C++ una biblioteca más moderna donde se almacena una clase de objetos strings con métodos para operarlos en forma más directa y sencilla. Entonces cabe plantear una serie de preguntas para aclarar la coexistencia de 2 elementos que parecen superponerse en el lenguaje de programación C++.

¿Existen dos tipos de strings en C++?
Así es. Los cstrings heredados de C que se operan como cadenas de caracteres finalizados con el caracter '/0' que denominaremos **cstrings** y los **objetos strings** que se crean a partir de la clase de igual nombre.

¿Por qué emplear cstrings del lenguaje C en C++?
Una de las razones es la de mantener la compatibilidad de código C en C+. También ciertas operaciones de C++ con cstrings no pueden realizarse a través de objetos de la clase string y deben operarse como cstrings. Es el caso de la representación de los nombres de archivos físicos, la cual debe hacerse siempre a través de un cstring.

¿Qué es un cstring?
Las cadenas de caracteres o cstrings constituyen unidades de información, formadas por secuencias de caracteres que se procesan en C/C++ como arreglos de caracteres que finalizan con el carácter nulo ('\0').

Caracteres

Hemos visto el tipo char para definir variables de tipo carácter. Una constante carácter en C++ se representa por cualquiera de los 256 símbolos del código ASCII encerrados entre apóstrofos.

```
char v='A';
```

En el ejemplo anterior se asigna el carácter 65 (la letra 'A') del código ASCII a la variable **v**. Para C++ una constante carácter es un entero (el valor ASCII de dicho carácter), por ello es posible realizar operaciones con variables de tipo char, no admitidas en otros lenguajes de programación. Obsérvense los ejemplos siguientes considerando que v tiene asignado 'A':

```
v++;  //asigna 'B' (ASCII 66) a la variable v
int x=10;
x+=v; //asigna 76 a x
```

También son caracteres muchos símbolos que no tienen una representación gráfica definida: salto de línea, tabulación, escape, etc.

```
char s=´\n´;//asigna un salto de línea (ASCII 10) a s
```

C++ dispone de algunas funciones para operar con datos de tipo char.

Constantes *cstrings*

Las constantes de tipo cstring en C++ se expresan encerrando la secuencia de caracteres entre comillas. Son ejemplos de cstrings:

```
"Alvez Julio"   "-----------"
"San Martín 3328–3000 Santa Fe" "0342-4565551"
```

Un constante cstring puede tener un único caracter. No debe confundirse esta constante como de tipo char, las cuales se representan encerradas por apóstrofos.

"A" es una constante cstring

'A' es una constante char

Las constantes cstrings pueden emplearse en los objetos de flujo de salida cout y pueden incluirse en ellas caracteres especiales para manipular la salida

```
cout<<"Programar en C++\n\nes interesante."
```

Obtendrá como salida:

```
Programar en C++

es interesante.
```

Declaración e inicialización de *cstrings*

Para declarar cstrings en C++ se debe utilizar un arreglo de caracteres o un puntero a caracter:

```
char nombre[]= "Farrell Julián";
char *direc= "San Martín 3328–3000 Santa Fe";
```

En el caso de que una función utilice como parámetros un cstring, se debe plantear en su prototipo el parámetro formal de tipo cstring como puntero a char usando el operador ***** obligatoriamente.

```
void muestra_cadena(char *s)
{
   cout<<s<<'\n';// muestra la cadena s
}
```

Estudiaremos los punteros y el operador de desreferencia * más adelante.

Al declarar una variable cstrings se debe usar la sintaxis de la declaración de arreglos de caracteres en C++:

```
char  palabra[12]={´I´,´n´,´t´,´e´,´r´,´n´,´e´,´t´};
```

aunque es posible asignarle también el valor inicial completo:

```
char  frase[20]="Ciberespacio";
```

Estas dos declaraciones e inicializaciones han asignado la información de ambos cstrings de la forma siguiente.

Palabra[0]	‘I’
Palabra[1]	‘n’
Palabra[2]	‘t’
Palabra[3]	‘e’

frase[0]	‘C’
frase[1]	‘i’
frase[2]	‘b’
frase[3]	‘e’

Palabra[0]	‘I’
Palabra[4]	‘r’
Palabra[5]	‘n’
Palabra[6]	‘e’
Palabra[7]	‘t’
Palabra[8]	‘\0’

frase[0]	‘C’
frase[4]	‘r’
frase[5]	‘e’
frase[6]	‘s’
frase[7]	‘p’
frase[8]	‘a’
frase[9]	‘c’
frase[10]	‘i’
frase[11]	‘o‘
frase[12]	‘\0‘

Las cadenas de caracteres en C++ finalizan con el carácter nulo ‘\0’ (ASCII 0). Este caracter actúa como señal o bandera para indicar el fin de la secuencia de caracteres que forman el cstring. Por lo tanto se debe especificar siempre un carácter más (como mínimo) de la cantidad que posee la cadena *visible* que se desea almacenar.

¿Qué ocurre con lo elementos declarados y no definidos en la inicialización del cstring?
Usualmente la dimensión propuesta en la declaración de un cstring supera a la cantidad de elementos asignados. Por ejemplo en la declaración:
```
char  palabra[12]=”Internet”
```
se establecen 12 caracteres para `palabra[]` y solo se emplean 8; el resto de los elementos dimensionados se asignan con el caracter nulo, o poseen valores al azar (dependiendo del compilador).

No se puede inicializar un cstring fuera de la declaración
C++ no admite ciertas operaciones con cstrings que involucren a la entidad completa, como es el caso de la asignación:
```
char  frase[20];//declara un cstring de 20 char
frase=“Ciberespacio”; //error!
```
Pero veremos que existen funciones que permiten realizar fácilmente estas tareas.

Entrada/Salida de cstrings

El ingreso de datos para variables cstrings se puede realizar a través del flujo **`cin`**, pero debemos considerar que este comando considera a ciertos caracteres como delimitadores del cstring: el carácter espacio en blanco ‘ ‘, la tabulación, el carácter de fin de línea y el de fin de archivo.
Si pretendemos leer con **cin** un cstring que incluya el carácter ‘ ‘ (espacio en blanco), el dato a asignar será truncado en el lugar donde se encuentre dicho espacio. Por ello C++ dispone de la función **cin.getline( )**. Esta función admite 3 parámetros: la variable de tipo arreglote caracteres donde alojar el cstring, la cantidad de caracteres a asignar y un parámetro de tipo char que hace de delimitador de la cadena.
```
cin.getline (char var[ ],int largo, char limite='\n');
```
por defecto el delimitador de la cadena a asignar a *var* es el carácter de avance de línea. Por ejemplo:

```
char  u[18];
cin.getline( u, 18);
```

La primer línea del código anterior declara un cstring **u**; la segunda lee desde la consola una línea de texto de hasta 18 caracteres. Si la entrada fuera de menor longitud lee hasta el carácter de avance de línea. Si en cambio empleamos:

```
char  u[18];
cin.getline(u,18,'+');
cout<<u<<endl;
```

E ingresamos por consola:

Programar en C++ es interesante

La salida debido al objeto `cout` será:

Programar en C

porque **`cin.getline(u,18,'+')`** cortará la entrada antes del caracter delimitador '+'.

Para imprimir cstrings en la pantalla se puede utilizar cout:, como se vió en el ejemplo anterior:

```
cout<<u<<endl;
```

Se debe notar que para cualquier otro tipo de datos diferente a char, al intentar mostrar una arreglo de esta forma en realidad se estaría mostrando el puntero al comienzo del mismo, por lo que en pantalla se imprimiría una dirección de memoria, y no el contenido (los elementos) del arreglo. Los arreglos de caracteres, como caso excepcional, se pueden utilizar de esta forma, y el como resultado se obtiene la cadena en pantalla; es decir, se muestran consecutivamente los caracteres del arreglo desde el comienzo (la dirección indicada por el puntero) hasta llegar al carácter de terminación '\0'.

Existen dos funciones adicionales, heredadas de C, **`gets`**( ) y **`puts`**( ) cuyos prototipos están definidos en el archivo **`cstdio`** que deberemos incluir en todo programa cliente de ambas funciones. Las funciones **`gets`**( ) y **`puts`**( ) admiten el espacio dentro del cstring:

`gets( )`: acepta un cstring desde el dispositivo estándar de entrada hasta encontrar un carácter de fin de línea y lo asigna a la variable indicada como argumento.

`puts( )`: envía al dispositivo de salida estándar la variable cstring indicada como argumento y reemplaza al carácter nulo con un carácter de nueva línea ('\n').

Recordemos que el flujo **`cout`** a diferencia de **`put`**, no agrega un carácter de próxima línea y permite enviar a salida cualquier expresión además de cstrings.

```
char  u[18],v[18];
//Se ingresa por teclado "Espacio digital" en la variable v
cin >> v;
```

Espacio Digital <ENTER>

pero solo se asigna *"Espacio"* a v

```
cout << v; // se obtiene como salida:
```

Espacio

Si se vuelve a ingresar por teclado el cstring "*Espacio digital"* en la variable u

```
gets(u);
```

Espacio Digital <ENTER>

```
// Se asigna "Espacio digital" a u
puts(u);/* se obtiene "Espacio digital" como salida y se produce un avance de línea */.
```

Espacio Digital

Se debe notar que la función gets no recibe entre sus argumentos la longitud del arreglo de caracteres donde va a colocar la cadena ingresada por el usuario, por lo que no se controla si el arreglo tiene tiene suficientes posiciones para almacenar dicha cadena (incluyendo el carácter de terminación). Es por esto que se recomienda utilizar cin.getline en lugar de gets.

Cómo definir un arreglo de cstrings

Una de las maneras de plantear en C++ un arreglo de cstrings es definir una matriz de caracteres: el primer índice nos ubica el carácter inicial de cada cstring':

```
char z[5][18]={"Santa Fe","Entre Ríos","Córdoba","Salta",
"Chaco"};
for (int i=0; i<5; i++)
     puts( z[i] ); //muestra un cstring por línea
puts z[2]; //obtiene "Córdoba" en la salida
```

Funciones de biblioteca para manejar cstrings

En el archivo de inclusión **cstring** se encuentran varios prototipos de funciones C++ predefinidas que nos permiten operar cstrings. Veamos algunas de ellas. Observará que se emplea el tipo **size_t** para declarar enteros; este tipo se halla definido en la biblioteca <cstdef> incluido en la biblioteca <cstring>, y corresponde a entero sin signo: **unsigned long**
También observará la presencia de la etiqueta **const** en algunos parámetros cstring; esto indica que la cadena de caracteres no será modificada en la función.

int strcmp(const char *c1, const char *c2)

Compara las cadenas y devuelve un entero menor que cero si **c1** está antes alfabéticamente que **c2** , devuelve cero si las cadenas son iguales; y devuelve un entero mayor que cero si **c2** está antes.

```
char a[25]; char b[25]
cout<< "Ingrese la primer cadena de caracteres:";
gets(a);
cout<< "Ingrese la segunda cadena de caracteres:";
gets(b);
if (strcmp(a,b)==0)
  cout<<"Ambas cadenas coinciden"
  else  if (strcmp(a,b)<0)
           cout<<a<<" está antes que "<<b;
        else
           cout<<b<<" está antes que "<<a;
```

int strncmp(const char *c1, const char *c2, size_t n)

Compara los primeros **n** caracteres de las cadenas empleadas como parámetros. Devuelve un entero menor que cero si los primeros n caracteres de **c1** forman una subcadena que está antes alfabéticamente que los primeros n caractrees de **c2** , devuelve cero si las subcadenas son iguales; y devuelve un entero mayor que cero si la subcadena de **c2** está antes que la de **c1**.

```
if strncmp(a,b,8)==0
    cout<<"Tienen los primeros 8 caracteres iguales\n";
  else
```

```
cout<<"Los primeros 8 caracteres no coinciden\n";
```

---

char *strcat(char *c1,const char *c2)

Concatena (agrega) el cstring indicado como segundo parámetro (**c2**) al final de la variable cstring indicado como primer parámetro (**c1**). La función devuelve el contenido de **c1**.

```
char a[20]="Internet"
char b[20]=" es la red de redes"
strcat(a,b);  //Concatena en a ambos cstrings
puts(a);  //muestra: Internet es la red de redes
```

---

char *strncat(char *c1,const char *c2, size_t n)

Concatena (agrega) los **n** primeros caracteres de **c2** a la variable **c1**. La función devuelve la dirección de **c1**.

```
char x[20]="Internet"
char y[20]=" es la red de redes"
strncat(x,y,10);
//agrega a x los primeros 10 caracteres de y
puts(x);  //muestra: Internet es la red
```

---

size_t strlen(const char *c)

Devuelve un entero con la longitud actual de la cadena argumento (sin incluir el carácter de terminación nulo).

```
char x[20]="Internet";
cout<<"La cantidad de caracteres del cstring ";
cout<<x<<" es:";
cout<<strlen(x);  //muestra el número 8
```

---

char *strstr(const char *c1, const char *c2)

Devuelve un puntero al inicio de la primer ocurrencia de c2 en c1. Si no encuentra c1 devuelve la dirección nula NULL.

```
char x[20]="Internet es la red de redes";
char y[20]="la red";
cout<<strstr(x,y)  //muestra: la red de redes
```

---

char *strchr(const char *c1,char c)

Devuelve un puntero al inicio de la primer ocurrencia del carácter c en c1. Devuelve NULL si no ocurre.

```
char x[20]="Internet es la red de redes";
puts(strchr(x, 'd'));  //muestra: d de redes
```

char tolower(char c)
char toupper(char c)
Ambas funciones convierten el carácter argumento en minúsculas y mayúsculas respectivamente. Para convertir una cadena completa se deben utilizar estas funciones sobre cada carácter:

```
char x[20]=”Internet”;
for (int i=0, n=strlen(x); i<n; i++)
  x[i] = toupper(x[i]);
cout<<x;  //muestra: INTERNET
```

Estas 2 funciones se hallan prototipadas en el archivo **cctype**.

char *strcpy(char *c1, const char *c2)
Esta función copia los caracteres de **c2** a **c1** y devuelve un puntero a c1. Podemos emplear esta función para asignar una variable cstring a otra.

```
char x[20]=”Internet es la red de redes”;
char y[20]=”la red”;
strcpy(x,y);
puts(x);  //muestra: la red
```

char *strncpy(char *c1, const char *c2, size_t n)
Esta función copia **n** caracteres de **c2** a **c1** y devuelve un puntero a **c1**. Si n es menor que la longitud de la cadena destino (**c1**), esta conserva el resto de los caracteres iniciales; si n es mayor que **c1** su longitud aumentará para dar cabida a los n caracteres, pero debe agregarse el carácter nulo **‘\0’** para completar el cstring.

```
char x[25]=”Pedro es programador”;
char y[25]=”Pablo es ingeniero”;
strncpy(x,y,5);   /*  copia  los  5  primeros
caracteres de y en x */
puts(x);  //muestra: Pablo es programador
```

Conversión de cstrings a otros tipos

Es posible emplear cstrings para capturar información que debe ser procesada en formato numérico. Por lo tanto es útil disponer de medios para convertir una cadena de tipo cstring a alguno de los tipos numéricos empleados en C++. Por ejemplo la cadena “124” no puede ser tratada numéricamente pues se trata de un cstring formado por los caracteres ‘1’, ‘2’ y ‘4’. C++ provee funciones para convertir cstrings y obtener (si es posible) el valor numérico correspondiente.

- **atoi(s):** devuelve un **int** correspondiente al cstring s
- **atol(s):** devuelve un **long** correspondiente al cstring s
- **atof(s):** devuelve un **float** correspondiente al cstring s

Si la conversión no fuera exitosa, estas funciones retornan cero.
Estas funciones se hallan en la librería **cstdlib.h** la cual debe ser incluida en el programa que las utilice.
Ejemplo:

```
cout<<"Ingrese el año:";
cin.getline(s);
int anio= atoi(s);
while (anio==0)
 {cout<<"Ud. debe ingresar un número para el año";
  cin.getline(s);
  int anio= atoi(s);
 }
```

Esta forma de ingresar información evita errores en tiempo de ejecución, pues controlamos los errores en nuestro programa.

Síntesis

1. Un dato de tipo char en c++ se almacena como un entero (el valor ASCII entre 0 y 255 del caracter) y puede ser operado algebraicamente. Pero en la entrada y salida se muestra el símbolo correspondiente.
2. Las cadenas de caracteres son un tipo de datos estructurado (similares a un arreglo de caracteres) pero en algunas ocasiones es posible utilizarlas como un dato simple o entidad completa.
3. En C++ los identificadores de cadenas o cstrings, al igual que los de los arreglos, representan una dirección de memoria al inicio de la cadena.
4. En C++ los cstrings se diferencian de los arrays porque finalizan con el carácter nulo (‘/0’).
5. Los caracteres que conforman una cadena pueden accederse de igual forma que los elementos de un arreglo unidimensional, a partir del elemento 0 y hasta la longitud de la cadena.
6. Existen varias funciones predefinidas en la biblioteca **string**.**h** para trabajar con las cadenas.
7. Es posible convertir un cstring a algunos de los tipos numéricos de C++ a través de funciones predefinidas disponibles en la biblioteca **cstdlib**. Si la cadena contiene caracteres que corresponden a un valor numérico la conversión será exitosa, en caso contrario la función de conversión devuelve cero.

Actividades

Ejercicios

Ejercicio 3.1.
Realice un programa para leer un arreglo lineal de N elementos conteniendo palabras de hasta 12 caracteres. Luego el programa deberá informar separadamente:
a) El primero de la lista de acuerdo a un orden alfabético.
b) Los elementos del arreglo que ocupan las posiciones pares.
c) Las palabras que comienzan con la sílaba 'mar' .

Ejercicio 3.2.
Escriba un programa que permita al usuario leer interactivamente un dato tipo cstring y exhiba un menú con las opciones: **1. Pasar a Mayúsculas 2. Pasar a Minúsculas. 3. Solo la inicial con mayúsculas.**
El programa debe resolver la opción seleccionada por el usuario.

Ejercicio 3.3.
Escriba un programa a partir del cual un usuario pueda ingresar una frase cualquiera y determine a través de funciones: a) La cantidad de vocales de la frase; b) La cantidad de consonantes; c) La cantidad de letras.

Ejercicio 3.4.
Escriba un programa que permita ingresar una lista de apellidos y nombres de 10 personas (apellido y nombre se asignan a una sola variable). Utilice para cada persona una variable de tipo cstring. El programa debe informar un listado con los 10 apellidos y luego un listado con los 10 nombres. Considere apellidos formados por una única palabra (la presencia del primer espacio en blanco indica el fin del apellido).

Ejercicio 3.5.
Escriba un programa que permita ingresar una lista de apellidos y nombres de N profesores de la FCyT-UADER (apellido y nombre se asignan a una sola variable). El programa debe mostrar las direcciones de correo electrónicos (e-mails) de ellos. El dominio asignado a la Facultad para el e-mail es: fcyt.uader.edu.ar, y el nombre de usuario se forma con la inicial del nombre y el apellido.
Ejemplo: Si el dato es Marelli Jorge, debe obtenerse: jmarelli@fcyt.uader.unl.edu.ar

Ejercicio 3.6.
Se ingresan las direcciones de correo electrónicos de un grupo de usuarios de Facultades de la UADER. Se debe determinar el número de usuarios de las Facultades de Ing. y Ciencias Hídricas (fich) y de Ing. Química (fiq).
Las direcciones tienen la denominación de la facultad a continuación del símbolo @.
Ejemplos: jperez@ fcyt.uader.edu.ar (Ing. Química), agarcia@ fcyt.uader.edu.ar (Cs Económicas), jprodo@ fcyt.uader.edu.ar (Ing. y Cs Hídricas), etc.

Ejercicio 3.7.

Se ingresan las direcciones de correo electrónicos de un grupo de usuarios de Facultades de la UNL. Se debe determinar la lista de dominios presentes entre las direcciones y el número de usuarios de cada dominio.

Ejercicio 3.8.
a) Escriba una función entera llamada `quita_ceros( )` que reciba un arreglo de enteros como parámetro y su longitud y elimine los ceros del mismo. Debe devolver la cantidad de elementos eliminada. b) Escriba una sobrecarga de la función anterior que reciba solo un c_string de parámetro y elimine sus espacios en blanco.

Ejercicio 3.9.
Calcule el promedio de N datos numéricos (correspondientes a calificaciones de un curso de N estudiantes. Ingrese los datos numéricos empleando cstrings y valide la entrada, solicitando de nuevo el dato si este fuera no válido (si por error no se ha ingresado un número). Ayuda: investigue el uso de las funciones atoi y atof.

Ejercicio 3.10
Escriba una función char *subcadena(char *d, const char *s, int i, int n). La función debe extraer la subcadena que se inicia en el caracter i del cstring s y tiene una longitud de n caracteres, guardarla en d y también retornala.
Ejemplo: ingresando "universidad", 1 y 3, debe devolver "niv".
Si no fuera posible extraer la subcadena debe retornar NULL. No utilice funciones de la biblioteca cstring.

Ejercicio 3.11
Escriba una función que reciba un cstring con una fecha en formato “dd/mm/aaaa” y extraiga de la misma los enteros correspondientes a día, mes y año, y los devuelva mediante un struct: struct fecha { int d, int m, int a; };

Resumen de Funciones

funciones de <cstring>

size_t **strlen** (const char *cadena); *// averiguar longitud*

char ***strcpy** (char *dest, const char *orig); *// copiar*

char ***strncpy** (char *dest, const char *orig, size_t n); *// copiar*

char ***strcat** (char *dest, const char *orig); *// concatenar*

char ***strncat** (char *dest, const char *orig, size_t n); *// concatenar*

int **strcmp** (const char *cad1, const char *cad2); *// comparar*

int **strncmp** (const char *cad1, const char *cad2, size_t n); *// comparar*

const char ***strchr** (const char *cad, const char car); *// buscar caracter*

const char ***strrchr** (const char *cad, const char car); *// buscar caracter*

const char ***strstr** (const char *cad, const char *sub); *// buscar subcadena*

size_t **strspn** (const char *cad, const char *cars); *// buscar caracteres*

char ***strtok** (char *ptr, const char *cars); *// buscar caract. y cortar cadena*

funciones de <cctype>

char **toupper** (char c); *// convertir a mayúsculas*

char **tolower** (char c); *// convertir a minúsculas*

funciones de <cstdlib>

int **atoi** (const char *s); *// convertir a entero*

long **atol** (const char *s); *// convertir a entero*

double **atof** (const char *s); *// convertir a double*

funciones de <iostream>

cin.getline (char *s, int n, char delim='\n'); *// leer por lineas*

cin.ignore (int n=1, char delim=EOF); *// ignorar parte de la entrada*