
La clase cadena

- Para explicar la sobrecarga de operadores y los diversos conceptos se va a crear una clase llamada cadena que permita trabajar con cadenas de caracteres de un modo más directo e intuitivo de lo que permite C.
 - Por ejemplo, en C no se puede copiar una cadena de caracteres en otra con el operador de asignación (=), sino que es necesario utilizar la función `strcpy()`; tampoco se pueden concatenar cadenas con el operador suma (+), ni se pueden comparar con los operadores relacionales (==) y (!=), sino que hay que utilizar las funciones `strcat()` y `strcmp()`, respectivamente.
-

La clase cadena

```
// fichero cadena.h
#include <iostream.h>
#ifndef __CADENA_H
#define __CADENA_H
class cadena {
private:
char* pstr;
int nchar; // nº de caracteres (sin el '\0')
public:
cadena(); // constructor por defecto
cadena(char*); // constructor general
cadena(const cadena&); // constructor de copia
~cadena(); // destructor
void setcad(char*); // dar valor a la variable privada pstr
// sobrecarga de operadores
cadena& operator= (const cadena&);
friend cadena operator+ (const cadena&, const cadena&);
friend cadena operator+ (const cadena&, const char* );
friend cadena operator+ (const char*, const cadena&);
friend int operator== (const cadena&, const cadena&);
friend int operator!= (const cadena&, const cadena&);
friend ostream& operator<< (ostream&, const cadena&);
/* Otros operadores que se podrían sobrecargar:
friend int operator== (const cadena&, const char*);
friend int operator== (const char*, const cadena&);
friend int operator!= (const cadena&, const char*);
friend int operator!= (const char*, const cadena&);
*/
};
#endif // __CADENA_H
```

La clase cadena

- Recuérdese que la declaración de una clase es de ordinario lo único que conocen los *programadores-usuarios* de la clase: el código de las funciones y operadores sólo acceden los que programan la clase.
 - Lo importante es ver que la ***declaración de la clase tiene toda la información necesaria para poder ser utilizada.***
 - Resaltamos los siguientes aspectos:
 1. En la definición de la clase no se ha reservado memoria para la cadena, sólo para el puntero ***pstr*** : no se sabe a priori cuántos caracteres va a tener cada objeto de la clase ***cadena***. *Queremos utilizar reserva dinámica de memoria: cuando se sepa el texto que se quiere guardar en un objeto determinado, se reservará la memoria necesaria para ello.*
-

La clase cadena

- 2. Hay **3 constructores**.

El primer constructor es un **constructor por defecto** que no requiere ningún argumento, e inicializa el objeto a una cadena vacía de cero caracteres. El segundo constructor se le pasa como argumento un puntero a una cadena de caracteres cualquiera y crea un objeto que apunta a esa cadena.

El tercer constructor es un **constructor copia** que recibe como argumento una **referencia constante una cadena** por motivos de eficiencia y seguridad.

Importante :

Los **constructores de oficio** no sirven en este caso: se han definido otros, y además **un atributo miembro es un puntero** (se va a utilizar **reserva dinámica de memoria**).

La clase cadena

- 3. Se ha definido un **destructor** porque se va a utilizar **reserva dinámica de memoria**, memoria que habrá que liberar expresamente.
 - 4. La función miembro **setcad()** permite proporcionar o cambiar el valor de la cadena de caracteres que contiene un objeto. Es una función miembro típica y sólo se ha introducido aquí a modo de ejemplo. Se le pasa como argumento un puntero a **char** que contiene la dirección del primer carácter del texto a introducir. No necesita devolver ningún valor.
 - 5. Se han definido **siete operadores sobrecargados** -uno como **miembro** y seis como **friends** -, y hay **cuatro operadores relacionales** más incluidos entre comentarios que se podrían terminar de definir de modo similar.
-

La clase cadena

- 6. El primer operador sobrecargado es el **operador de asignación (=)**. Como es un operador binario que modifica el primer operando **debe necesariamente ser definido como miembro**. Este operador asigna un objeto **cadena** a otro. El miembro izquierdo de la igualdad es el primer operando y en este caso es el argumento implícito del operador. En la lista de argumentos formales que figura entre paréntesis sólo hay que incluir el segundo operando, que es un objeto cadena que se pasa **por referencia** y como **const**, pues no debe ser modificado. El **valor de retorno** de este operador requiere un comentario especial.
 - En este caso el **operador de asignación (=)** **no necesita ningún valor de retorno**: su misión en una sentencia tal como **c1 = c2**; queda suficientemente cumplida si hace que las variables miembro de **c1** sean iguales a las de **c2** (siendo **c1** y **c2** dos objetos de la clase **cadena**).
-

La clase cadena

- Sin embargo, el **operador (=) estándar de C** tiene un valor de retorno que es una referencia al resultado de la asignación. Esto es lo que permite escribir sentencias como la siguiente: **$a = b = c;$** que es equivalente a hacer **b** igual a **c** , y devolver un valor que puede ser asignado a **a** . Al final las tres variables tienen el mismo valor. Para que el operador sobrecargado se parezca todo lo posible al de C y para poder escribir sentencias de asignación múltiples con objetos de la clase ***cadena*** (**$c1 = c2 = c3;$**), es necesario que el operador de asignación sobrecargado (=) tenga valor de retorno. El **valor de retorno es una referencia al primer operando** por motivos de eficiencia, pues si no hay que crear un objeto nuevo, diferente de los dos operandos, para devolverlo como valor de retorno.
-

La clase cadena

- 7. Tres **operadores suma** (+) han sido sobrecargados **como friend** para **concatenar cadenas**. No hay argumento implícito, y los dos argumentos aparecen entre paréntesis. Los tres operadores (+) tienen un objeto de la clase **cadena** como valor de retorno. El primero de ellos **concatena dos objetos** de la clase **cadena**, y devuelve un nuevo objeto **cadena** cuyo texto es la concatenación de las cadenas de caracteres de los objetos operandos. Este resultado es diferente del de la función **strcat()**, que añade el texto del segundo argumento sobre el primer argumento (modificándolo por tanto). Los otros dos operadores (+) sobrecargados concatenan el texto de un **objeto cadena** y una **cadena de caracteres estándar**, y una **cadena de caracteres estándar** y el texto de un **objeto cadena** respectivamente. Recuérdese que el orden y el tipo de los argumentos deciden qué definición del operador sobrecargado se va a utilizar. Si se quiere poder escribir tanto **c1 + "anexo"** como **"prefacio " + c2** es necesario programar dos operadores distintos, además del que concatena dos objetos **c1 + c2**.
-

La clase cadena

- 8. La declaración de los **operadores relacionales** (`==`) y (`!=`), que permiten saber si dos objetos contienen o no el mismo texto.
 - En este caso el valor de retorno del **test de igualdad** (`==`) es un **int** que representará **true** (1) o **false** (0). Para el **operador de desigualdad** (`!=`) la situación es un poco más compleja, pues se desea que sirva para ordenar cadenas alfabéticamente, de modo similar a la función **strcmp()**. Por ello el valor de retorno de **c1!=c2** es un (-1) si **c1** es diferente y anterior alfabéticamente a **c2**, un cero (0 ó **false**) si las cadenas son idénticas, y un (1) si son diferentes y **c1** es posterior a **c2** en orden alfabético.
-

La clase cadena

- 9. Finalmente, en la declaración de la clase ***cadena*** se muestra la sobrecarga del operador de ***inserción en el flujo de salida*** (<<), que tiene como finalidad el poder utilizar ***cout*** con objetos de la clase. Este operador, al igual que el (>>), se debe sobrecargar como operador ***friend***. Recibe dos argumentos: Una referencia al flujo de salida (***ostream***, de *output stream*) y una referencia constante al objeto ***cadena*** que se desea insertar en dicho flujo. El ***valor de retorno*** es una referencia al flujo de salida ***ostream*** en el que ya se habrá introducido el objeto ***cadena***.
-

La clase cadena

```
// cadena.cpp
#include <string.h>
#include "cadena.h"
// constructor por defecto
cadena::cadena() {
    pstr = new char[1];
    strcpy(pstr, "");
    nchar = 0;
    cout << "Constructor por defecto " <<
        (long)this << endl; }
```

- Dos observaciones acerca del **constructor por defecto** (sin argumentos). La primera es que la variable miembro **pstr**, que es un puntero a *char*, se inicializa apuntando a una cadena vacía (sólo tiene el '\0' de fin de cadena). Se utiliza reserva dinámica de memoria con el operador **new**. La variable miembro **nchar**, que representa el número de caracteres, no incluye el carácter de final de cadena. Se ha incluido una sentencia de escritura que imprimirá un mensaje avisando de que se ha utilizado este constructor. Imprime también la **dirección en memoria del objeto creado**, con idea de saber cuándo se crea y se destruye cada objeto concreto del programa. Para ello se utiliza el puntero **this** y un **cast** a **long**.
-

La clase cadena

```
// constructor general
cadena::cadena(char* c)
{ nchar = strlen(c);
  pstr = new char[nchar + 1];
  strcpy(pstr, c);
  cout << "Constructor general " <<
    (long)this << endl; }
```

- El **constructor general** admite como argumento la dirección del carácter inicial de una cadena de caracteres, a partir de la cual se inicializará el nuevo objeto. Se utiliza también reserva dinámica de memoria. Se utilizan las funciones **strlen()** y **strcpy()** para determinar el número de caracteres y para copiar el argumento en la dirección a la que apunta la variable miembro **pstr**. Se imprime un mensaje que permite saber qué constructor se ha llamado y qué objeto ha sido creado.
-

La clase cadena

```
//constructor de copia
cadena::cadena(const cadena& cd) {
nchar = cd.nchar;
pstr = new char[nchar +1];
strcpy(pstr, cd.pstr);
cout << "Constructor de copia " <<
(long)this << endl;}
```

- Puede verse que el **constructor de copia** es muy similar al **constructor general**, con la única diferencia de que recibe como argumento una referencia a un objeto **cadena**, a partir del cual inicializa el nuevo objeto con reserva dinámica de memoria. Obsérvese que se tiene acceso a las variables miembro del objeto cadena pasado como argumento, pero que hay que utilizar el operador punto (.). A las variables miembro del objeto pasado como argumento implícito se tiene acceso directo.
-

La clase cadena

```
// destructor
cadena::~cadena()
{
delete [] pstr;
cout << "Destructor " << (long)this <<
endl; }
```

- En este caso no sirve el destructor de oficio, porque se está utilizando reserva dinámica de memoria. El destructor debe liberar la memoria ocupada por las cadenas de caracteres, para lo que hay que utilizar la sentencia ***delete [] pstr.*** Además se imprime un mensaje incluyendo la dirección del objeto borrado. De este modo se puede saber cuándo se crea y se destruye cada objeto, lo cual será de gran utilidad en los ejemplos que se presentarán más adelante.
-

La clase cadena

```
// función miembro setcad()
void cadena::setcad(char* c)
{ nchar = strlen(c);
  delete [] pstr;
  pstr = new char[nchar + 1];
  strcpy(pstr, c);
  cout << "Función setcad()" << endl; }
```

- La función miembro **setcad** permite sustituir el contenido de un objeto **cadena** a partir de una cadena de caracteres estándar. Esta función se diferencia del constructor general visto previamente en que actúa sobre un objeto que ya existe. Esto hace que la primera tarea a realizar sea liberar la memoria a la que la variable **pstr** apuntaba anteriormente. Después se reserva memoria para el nuevo contenido al que **pstr** apuntará. Los constructores siempre actúan sobre un objeto recién creado, y por eso no tienen que liberar memoria.
-

La clase cadena

- ```
// operador de asignación
//sobrecargado (=) cadena&
cadena::operator= (const cadena& cd) {
if(*this != cd) {
nchar = cd.nchar;
delete [] pstr;
pstr = new char[nchar + 1];
strcpy(pstr, cd.pstr);
cout << "Operador =" << endl; }
return *this; }
```
  - En la definición del **operador miembro** (=) llama la atención la sentencia **if** que aparece al comienzo de la función. Su razón de ser es evitar los efectos perjudiciales que puede tener una sentencia de **asignación de un objeto a sí mismo** (**c1=c1;**).
  - Si no se introduce ese **if** esta sentencia es todo menos inofensiva. Al asignarse un objeto a otro, lo primero que se hace es liberar la memoria ocupada por el primer operando (el que está a la izquierda), para después sacar una copia de la memoria a la que apunta el segundo operando y asignar su dirección a la variable miembro del primero. El problema es que si ambos objetos coinciden (son el mismo objeto), al liberar la memoria del primer operando, el segundo (que es el mismo) la pierde también y ya no hay nada para copiar ni para asignar, llegándose en realidad a la destrucción del objeto.
-

---

# La clase cadena

```
// operador de inserción ostream&
operator<< (ostream& co, const
cadena& cad) {
co << cad.pstr;
return co;}
```

- La definición del **operador inserción** (<<) sobrecargado es muy sencilla, pues lo único que se hace es insertar en el flujo de salida la cadena de caracteres estándar a la que apunta la variable miembro **pstr**.

```
//operadores para concatenar cadenas
```

---

---

# La clase cadena

```
cadena operator+ (const cadena& a, const cadena&
 b){
 cadena c;
 c.nchar = a.nchar + b.nchar;
 c.pstr = new char[c.nchar + 1];
 strcpy(c.pstr, a.pstr);
 strcat(c.pstr, b.pstr);
 return c;}

cadena operator+ (const cadena& a, const char*
 ch){
 cadena c;
 c.nchar = a.nchar + strlen(ch);
 c.pstr = new char[c.nchar + 1];
 strcpy(c.pstr, a.pstr);
 strcat(c.pstr, ch);
 return c;}

cadena operator+(const char* ch, const cadena&
 b){
 cadena c;
 c.nchar = strlen(ch) + b.nchar;
 c.pstr = new char[c.nchar + 1];
 strcpy(c.pstr, ch);
 strcat(c.pstr, b.pstr);
 return c;}
```

---

---

# La clase cadena

```
// sobrecarga de los operadores
//relacionales

int operator== (const cadena& c1, const
 cadena& c2){
 if(strcmp(c1.pstr, c2.pstr)==0)
 return 1;
 return 0;}

int operator!= (const cadena& c1, const
 cadena& c2){
 int dif = strcmp(c1.pstr, c2.pstr);
 if(dif<0)
 return (-1);
 if(dif==0)
 return 0;
 else
 return 1;}
// fin del fichero cadena.cpp
```

---