Contenido

Anexo 1. Usando cadenas y vectores con la STL

La STL (Standard Template Library) http://www.sgi.com/tech/stl/ es una de las grandes ventajas que C++ tiene con respecto a C.

1. Para usar **cadenas** (vector de caracteres) debemos poner #include <string> en nuestro programa.

   Invertir una cadena en C++ es algo muy simple; sólo hay que recorrer cada uno de los elementos del vector desde el último hasta el primero.

   El código podría ser el siguiente:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
	string cadena = " Hola mundo!";
	string aux_cadena = "";
	for (int i = cadena.size(); i >=0 ; i--)
		aux_cadena += cadena[i];
	cout<<"El texto normal es: "<<cadena<<endl;
	cout<<"El texto invertido es: "<<aux_cadena<<endl;
	return 0;
}
```

Entre otras cosas, la STL contiene muchos algoritmos ya programados que podemos usar y que nos facilitan mucho trabajo.

Como ejemplo, podríamos hacer un programa para invertir una cadena usando la STL, para ello se utilizará el algoritmo “reverse”. Para utilizar los algoritmos de la STL, debemos poner #include <algorithm> en nuestro programa.

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
int main()
{
	string cadena = "Hola mundo!";
	string aux_cadena(cadena.begin(), cadena.end());
	reverse(aux_cadena.begin(), aux_cadena.end());
	cout<<"El texto normal es: "<<cadena<<endl;
	cout<<"El texto invertido es: "<<aux_cadena<<endl;
	return 0;
}
```

2. Con la STL se pueden crear arreglos, **vectores**, sin indicar su tamaño, esto es útil porque probablemente no sepamos de antemano el número de elementos que vaya a contener el arreglo y queremos ir agregando elementos en tiempo de ejecución.

Para usar vectores debemos poner #include <vector> en nuestro programa.

El siguiente programa pide al usuario que introduzca números y los va almacenando en un vector, el programa termina cuando el usuario introduce el número cero. Al final se recorre el vector y se muestran los números que el usuario introdujo (podemos recorrerlo como si se tratase de un arreglo).

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
	int i_numero=1;
	vector<int> v_enteros;
	cout << "Teclee un número, (0 para terminar):" << endl;
	/* Mientras el usuario teclee números diferentes de cero, se almacena en el
	vector los números que el usuario va tecleando*/
	while (i_numero != 0)
	{
		cin >> i_numero;
		if (i_numero != 0)
			v_enteros.push_back(i_numero);
	}
	cout << endl << " Estos son los números introducidos:" << endl;
	// Muestra los valores de los enteros almacenados en el vector
	for (unsigned int i = 0; i < v_enteros.size(); i++)
		cout << v_enteros[i] << endl;
	return 0;
}
```

Anexo 2. Entrada y Salida con ficheros en C++.

- Para efectuar la salida de C++ en ficheros es necesario incluir **`#include <fstream>`** al principio del fichero fuente. Este header proporciona dos nuevos tipos:
  - **`ifstream`** para ficheros de entrada y
  - **`ofstream`** para ficheros de salida.
- La forma de utilizarlos es la siguiente: Supongamos que deseamos efectuar la salida del programa en el fichero **`sal.txt`**. Creamos una nueva unidad de salida análoga a **`cout`** que denominamos **`fout`** por ejemplo.

  Para ello incluimos la línea **`ofstream fout("sal.txt");`** en nuestro programa.

  Análogamente si deseamos leer desde un fichero llamado **`input.dat`**, creamos una unidad de entrada que denominamos **fin** por ejemplo, análoga a **`cin`**

  `ifstream fin("input.dat")`

  Con estas unidades de entrada y salida podemos leer y escribir desde fichero, de forma análoga a como lo hacemos con *cin* y *cout* en la entrada y salida standard.

- Por ejemplo, si tenemos una variable ***r*** y queremos calcular una variable `V=4*pi*r*r*r`, incluiremos las sentencias

  ```
  cout<<"entrar radio"<<endl;

  fin>>r;

  V=4*pi*r*r*r;

  fout<<" volumen= "<<V<<endl;
  ```

- A veces es necesario dar el camino completo donde se encuentran los ficheros. Por ejemplo si `input.dat` se encuentra en `C:\Archivos de programas\datos`, habrá que darle la ruta completa. Para ello deberemos tener en cuenta que \ es un carácter de escape y para introducir \ hay que hacerlo como \\. Además los blancos se interpretan como fin de sentencia, por lo cual deberemos introducir

  ```
  ifstream fin("C:\\""Archivos de Programas""\\datos")
  ```

  Las segundas comillas son necesarias para que las interprete como la apertura de unas segundas comillas y no como el fin de las primeras.

- Si tenemos un fichero con datos importantes, deberemos asegurarnos de que no escribimos sobre él por error, destruyendo los datos existentes. Esto se consigue definiendo el fichero como de sólo lectura. Esto se consigue con la clase **`ios_base`**. Por ejemplo si tenemos un fichero `input.dat` y queremos declararlo como de sólo lectura incluiremos la sentencia

  ```
  ifstream fin("input.dat",ios_base::in);
  ```

  Tambien se puede declarar un fichero de solo escritura con `ios_base::out`. Puede ser interesante en determinadas ocasiones el añadir las salidas de sucesivas ejecuciones de un programa a un fichero. Esto se consigue abriéndolo en modo añadir,

  ofstream fout("salida.dat", ios_base::app);

- Las diferentes opciones se pueden combinar por medio del operador **|**. Por ejemplo,

  ```
  ofstream fout("salida.txt",ios_base::app|ios_base::out);
  ```

  declara fout como de sólo escritura y en modo añadir.

- Un fichero puede declararse simultáneamente de lectura y escritura mediante el tipo **`fstream`**. Por ejemplo,

  ```
  fstream iofi("temp.dat",ios_base::in|ios_base::out);
  ```

  declara al fichero `temp.dat` como de escritura y lectura simultáneamente. Por supuesto, esto rara vez es necesario, salvo en programas que generan una gran cantidad de datos que son utilizados por la siguiente ejecución del programa.

- Los ficheros utilizados deben ser cerrados antes de acabar el programa. Esto se consigue mediante las sentencias `fout.close(); fin.close(); iofi.close();`

Anexo 3. Representación gráfica. Gráficas con Gnuplot.

Los lenguajes C y C++ carecen de posibilidades de representación gráfica en su definición estándar. Sin embargo existe un gran número de librerías gráficas tanto públicas como comerciales que permiten la salida gráfica tanto en C como C++. En esta práctica utilizaremos el programa público *gnuplot* como utilidad de representación gráfica, realizando el proceso en tres etapas: nuestro programa imprime los resultados en ficheros de datos y posteriormente graba un fichero de comandos para gnuplot (script) para que lea esos ficheros de datos para representarlos gráficamente y, para finalizar, ejecuta el script desde el sistema operativo (o desde el programa en C++ a través de una llamada al sistema) guardando los resultados en un fichero.

gnuplot puede exportar las gráficas en un gran número de formatos gráficos, que podemos utilizar para confeccionar posteriormente la memoria de la práctica.

Gnuplot : nociones elementales.

Para ejecutar *gnuplot* escribir *gnuplot* en la línea de comandos, o hacer click en el icono de gnuplot, dependiendo del sistema operativo.

- El comando

  plot sin(x)

  dibuja la función sin(x) entre -10 y 10.

- El comando

  plot [0:10] sin(x)

  dibuja la función *sin(x)* entre 0 y 10.

- Si existe un fichero de datos llamado *datos.dat,* el comando

  plot "datos.dat"

  dibuja los puntos. El fichero *datos.dat* debe de tener en la primera columna la variable *x* y en la segunda la variable *y.*

- El comando

  replot cos(x)

  dibuja la función cos(x) en la misma ventana que está abierta, superponiendo la gráfica sobre las gráficas que ya están dibujadas. El comando *plot* dibuja la gráfica borrando lo que previamente se había dibujado.

- Si se tiene una función calculada numéricamente en un conjunto de puntos contenidos en un fichero *funcion.dat* y se quiere unir con una línea continua habrá que escribir

  plot "funcion.dat" **with lines**

- Un conjunto de comandos de *gnuplot* se puede escribir en un fichero (*cmdfich.gpl* por ejemplo), que se puede cargar en *gnuplot* mediante el comando

  load "cmdfich.gpl"

- Para utilizar las gráficas realizadas con *gnuplot* en documentos hace falta guardarlas en un formato adecuado. Se utiliza el comando ***set output*** para redirigir la salida de *gnuplot* a un fichero. Si se escribe

  set terminal pdf

  set output "Insercion.pdf"

  plot "datos.dat"

  se dibuja el contenido de *datos.dat* en el fichero "Insercion.pdf".

- También se pueden dibujar varias series de datos en la misma gráfica, separando las instrucciones de cada dibujo por comas

  plot "f" using 1:2 title 'Insercion_tiempos'**,** "f" using 1:3 title 'Burbuja_tiempos'

- Para activar/desactivar la escala logarítmica en el eje y(lo mismo para el eje x):

set logscale y

unset logscale y

- Se pueden dibujar gráficas de funciones sencillas que servirá para comparar los datos obtenidos contra su coste teórico. Por ejemplo, si sabemos que el coste teórico del algoritmo es lineal se podrá dibujar la comparativa como:

  plot "f" using 1:2 title 'Binaria_real', x title 'Binaria_teorico'

  Si el coste fuera cúbico se podría poner:

  plot "f" using 1:2 title 'Binaria_real', x**3 title 'Binaria_teorico'

❖ **Como ajustar curvas.**

1. Se define la función de la que se desea hacer el ajuste.

   En el caso de un polinomio de segundo grado del que se desean ajustar los parámetros a,b,c, la función será

   y(x)=a+b*x+c*x*x

2. Se ajusta la curva de puntos empíricos por mínimos cuadrados mediante la función 'fit' de *gnuplot*.

   fit y(x) "f" using 1:2 via a,b,c

   así se ajusta la curva definida por las columnas 1 y 2 del fichero "f" calculando el valor de los parámetros a,b,c**.** Aparecerán en pantalla las iteraciones del algoritmo de ajuste y el resultado final del mismo.

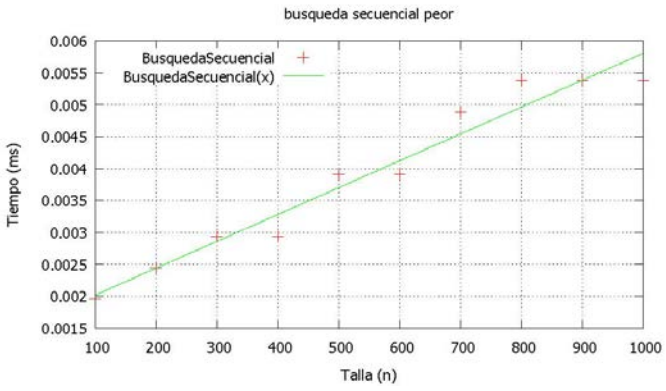
3. Se representa la curva empírica y la curva teórica ajustada.

   plot "f" using 1:2 title 'Insercion_real', y(x) title 'Inserción_teórico'

Ejemplos de ficheros de comandos.

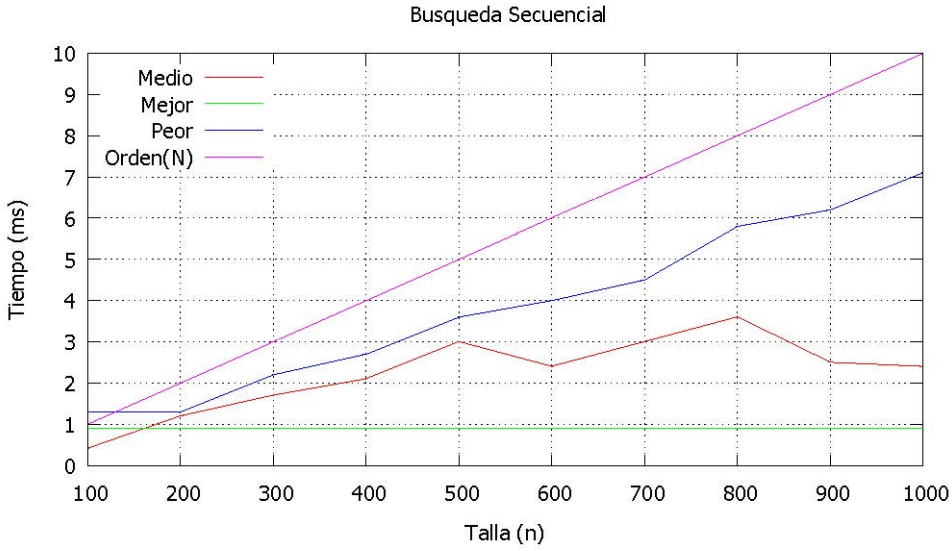
1. Fichero por ejemplo "`CmdCaso.gpl`"

```
 # ESTE ES UN SCRIPT DE GNUPLOT PARA ESTUDIO DE UN CASO
set title "busqueda secuencial peor"
set key top left vertical inside
set grid
set xlabel "Talla (n)"
set ylabel "Tiempo (ms)"
plot "secuencialPeor.dat" using 1:2
BusquedaSecuencial(x) = a*x + b
fit BusquedaSecuencial(x) "secuencialPeor.dat" using 1:2 via a,b
plot "secuencialPeor.dat" using 1:2 title 'BusquedaSecuencial',
BusquedaSecuencial(x)
set terminal pdf
set output "secuencialPeor.pdf"
replot
pause 5"Pulsa Enter para continuar..."
```



2. Fichero por ejemplo "`CmdComparar.gpl`"

```
#ESTE ES UN SCRIPT DE GNUPLOT PARA COMPARACION DE LOS CASOS DE UN ALGORTIMO
set title " Busqueda Secuencial"
set key top left vertical inside
set grid
set xlabel "Talla (n)"
set ylabel "Tiempo (ms)"
plot "secuencialMedio.dat" using 1:2 with lines title "Medio",
"secuencialMejor.dat" using 1:2 with lines title "Mejor","secuencialPeor.dat"
using 1:2 with lines title "Peor","secuencialOrden.dat" using 1:2 with lines title
"Orden(N)"
set terminal pdf
set output "BusquedaSecuencial.pdf"
replot
pause 5 "Pulsa Enter para continuar..."
```



❖ **Enlaces web**

http://www.gnuplot.info (página oficial de *gnuplot*)

http://www.duke.edu/~hpgavin/gnuplot.html