Programación 2

Grado en Estadística Aplicada

Curso 2012-2013

Generación de números pseudoaleatorios. Manejo de ficheros de texto.

Jesús Correas – jcorreas@fdi.ucm.es

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

Generación de números pseudoaleatorios

- La generación de números aleatorios es muy útil en muchas aplicaciones estadísticas.
- En C++ existen funciones de librería para simular el experimento de elegir un número entero al azar.
- Veremos las **funciones básicas de la librería `cstdlib`: `srand()` y `rand()`**.
- **`srand()`** proporciona una **semilla** inicial, un valor lo más aleatorio posible. Normalmente se usa la hora que marca el reloj del ordenador (función `time()` de la librería `ctime`).
- **La semilla se debe proporcionar una sola vez al principio de la ejecución del programa.**

Generación de números pseudoaleatorios

- Una vez proporcionada la semilla, se pueden generar números pseudoaleatorios con la función **`rand()`**.
- Esta función proporciona un número entero entre 0 y `RAND_MAX` (normalmente igual a `INT_MAX`).
- Se puede utilizar para proporcionar números pseudoaleatorios en un rango menor. Por ejemplo, para generar números entre 0 y **99** :

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main() {
  srand(time(NULL));
  cout << "Generando números pseudoaleatorios:";
  for (int i = 0; i < 10; i++) {
    int num = rand() % 100;
    cout << " " << num;
  }
  cout << endl;
  return 0;
}
```

Manejo de ficheros de texto

- Escritura de datos en ficheros de texto.
- Lectura de datos de ficheros de texto.

Manejo de ficheros de texto

- Hasta ahora hemos visto que la forma que tiene un programa de **leer datos y escribir resultados** es a través del teclado y la pantalla (`cin` y `cout`).
- Sin embargo, es muy útil poder utilizar **ficheros en disco** para:
  - guardar o recuperar **información que pueda ser leída o generada por otros programas.**
  - **almacenar información de forma persistente.**
    La memoria, en cambio, mantiene información de forma **volátil**, solamente mientras el programa se está ejecutando.
- Veremos una **introducción** al manejo de **ficheros de texto**, que pueden ser leídos desde un editor de texto (por ejemplo, el bloc de notas).
- Los ficheros de texto se pueden generar fácilmente desde programas estadísticos, hojas de cálculo, bases de datos, etc.
- De esta forma, un programa en C++ puede utilizar y generar **grandes cantidades de datos** de forma sencilla.

Manejo de ficheros de texto

- Para manejar ficheros de texto debe utilizarse la librería **`fstream`**.
- Desde un programa, los ficheros de texto se pueden utilizar en tres modos fundamentales:
  - Modo **lectura**: el programa puede leer los datos del fichero para utilizarlos, **pero no puede modificar el fichero.**
  - Modo **escritura**: el programa solamente puede escribir datos en el fichero. El contenido que hubiera en el fichero antes de ejecutar el programa se pierde.
  - Modo **"append"**: el programa solamente puede escribir datos en el fichero, pero los datos escritos **se añaden al final de lo que hubiera previamente en el fichero**
- Para trabajar con un fichero se deben realizar siempre tres pasos:
  - Abrir el fichero.
  - Leer o escribir datos en el fichero.
  - Cerrar el fichero.
- **Es fundamental realizar estos tres pasos para que el programa funcione correctamente.**

Escritura de datos en ficheros de texto

- Vamos a ver cómo se escriben datos en un fichero con un ejemplo:

```
#include <fstream>
#include <cstring>
using namespace std;

const int STR = 50;
struct tLibro {
  int codigo;
  char titulo[STR];
  char autor[STR];
};
```

```
int main() {
  tLibro libro;
  libro.codigo = 17;
  strcpy(libro.titulo,"El origen de las
      especies");
  strcpy(libro.autor, "Darwin, C.");
  ofstream f;
  f.open("datos.txt");
  f << libro.codigo << endl;
  f << libro.titulo << endl;
  f << libro.autor << endl;
  f.close();
  return 0;
}
```

- Un fichero se representa en C++ mediante **una variable**.
- Para abrir un fichero en modo escritura, debe ser de tipo **`ofstream`**.
- Las operaciones sobre el fichero se realizan llamando a **métodos**, similares a llamadas a funciones. También se utiliza el **operador** $<<$.

Escritura de datos en ficheros de texto

- Los datos que se pueden escribir utilizando ficheros de texto deben ser de tipos básicos (números, caracteres) o cadenas de caracteres (arrays de caracteres o strings).
- Para escribir en un fichero datos de una estructura, hay que **escribir los campos uno a uno**.
- Hay que tener cuidado para escribir los datos de forma que después se puedan leer correctamente. Por ejemplo, el siguiente código escribe datos en un fichero de forma **incorrecta**. **¿Por qué?**

```
int main() {
  int fila, columna, valor;
  cout << "Teclea los datos de fila, columna y valor:
  cin >> fila >> columna >> valor;
  ofstream f;
  f.open("datos.txt");
  f << fila;
  f << columna;
  f << valor << endl;
  f.close();
  return 0;
}
```

Lectura de datos de ficheros de texto

- Para abrir un fichero de texto en modo lectura, es necesario crear una variable de tipo **`ifstream`**.
- Como en el caso anterior, hay que abrir el fichero, leer los datos que se quieran leer y cerrar el fichero.
- Los ficheros de texto se leen **de forma secuencial**: para poder leer un dato concreto del fichero, hay que leer primero todos los datos anteriores.
- Los ficheros de texto están estructurados en **líneas de texto** separadas por **saltos de línea**.
- Las líneas de texto tienen longitud variable.
- El contenido de las líneas de texto depende del programa que se utilice para leerlo/escribirlo.
- Existen dos formas básicas para leer datos de un fichero de texto con la librería **fstream**:
  - Lectura sin formato: método **`getline`** (y función **`getline`**).
  - Lectura con formato: operador $>>$.

Lectura sin formato: el método `getline`

- El método **`getline`** permite leer una línea entera de un fichero de texto y guardarla en un array de caracteres.
- Debe proporcionarse como parámetro el array de caracteres en el que guardar la línea de texto y el número máximo de caracteres que se quieren leer (el tamaño del array). Un ejemplo:

```
char dato[MAX];
ifstream ficEnt;
ficEnt.open("datos.txt");
if (ficEnt.is_open()) {
  while (!ficEnt.eof()) {
    ficEnt.getline(dato,MAX);
    cout << "El dato leido es: *" << dato << "*" << endl;
  }
  ficEnt.close();
}
```

- Además, se utilizan dos métodos booleanos, **`is_open`** y **`eof`**, que veremos más adelante.
- La **función** **`getline`**, definida en la librería **`string`**, es similar pero utiliza un string para almacenar la línea leída.

Lectura con formato: el operador >>

- El operador >> que ya hemos visto para leer datos del teclado se puede utilizar para leer datos de un fichero de texto.
- Este operador realiza **lectura con formato**: intenta leer un dato **del mismo tipo** que la variable que se utiliza para almacenarlo.
- En el caso de variables de tipo string o array de caracteres, lee los caracteres del fichero **hasta el primer espacio en blanco**. Para poder leer una cadena de caracteres con espacios debe utilizarse `getline`.

```
char dato[MAX];
ifstream ficEnt;
ficEnt.open("datos.txt");
if (ficEnt.is_open()) {
  while (!ficEnt.eof()) {
    ficEnt >> dato;
    cout <<"*"<<dato<<"*"<< endl;
  }
  ficEnt.close();
}
```

- Si `datos.txt` contiene la línea "Nicanor Cienfuegos", se escribe en pantalla:

```
*Nicanor*
*Cienfuegos*
```

Combinando `getline` y `>>`

- Cuando se abre un fichero en modo lectura, el programa mantiene internamente la **posición del fichero** a partir de la cual se va a realizar la siguiente operación de lectura. Esta posición se denomina **puntero de lectura**.
- Se pueden combinar las operaciones de lectura con `getline` y con `>>` teniendo en cuenta cómo se posiciona el puntero de lectura con cada operación:
  - Con `getline` se lee una línea completa y el puntero se posiciona **al principio de la siguiente línea**.
  - Con `>>` se lee un dato del tipo de la variable y el puntero se posiciona **justo después del último carácter del dato leído**.
- Si se utiliza `>>` y a continuación `getline` para leer datos en dos líneas del fichero de entrada, puede ser necesario realizar una lectura adicional para eliminar el salto de línea entre los dos datos.
  - Se puede utilizar el método `ignore`:

```
ifstream ficEnt("datos.txt");
//ignora hasta el primer salto de linea o hasta 256 caracteres.
ficEnt.ignore(256,'\n');
```

Otros métodos útiles de `ifstream`

- **`fic.is_open()`** Devuelve `true` si el fichero al que se refiere `fic` está abierto. Esta condición **es recomendable comprobarla siempre después de abrir un fichero**, tanto de lectura como de escritura.
- **`fic.eof()`** Devuelve `true` si se ha llegado al final del fichero al que se refiere `fic`.
- **`fic.good()`** Devuelve `true` si no ha habido ningún error en la última operación sobre `fic`.