Universidad Autónoma de Entre Ríos
Facultad de Ciencia y Tecnología
Sede: Oro Verde

FUNDAMENTOS DE PROGRAMACIÓN

RESUMEN DE CONTENIDOS N° 13

ARCHIVOS

RESUMEN DE CONTENIDOS N° 13

Archivos

INTRODUCCIÓN

Los datos tratados por un programa, como hemos visto hasta ahora, tienen dos limitaciones importantes: por un lado, la cantidad de datos que puede almacenar la memoria principal del procesador (RAM) ya que ésta es limitada; y por otro, la existencia de los datos esta condicionada al tiempo que dure la ejecución del programa, es decir, cuando éste termina todos sus datos desaparecen de la memoria central del procesador y se pierden.
Se utilizan entonces las estructuras de datos externas denominadas **FICHEROS** o **ARCHIVOS** para manipulación y almacenamiento de datos para usos futuros.

Los conceptos que introduciremos, son válidos y adaptables a cualquier lenguaje de programación. Pero más adelante, en esta unidad, se hará hincapié en su forma de uso específicamente en C++.

ARCHIVOS

Los archivos no están contenidos en la memoria central del procesador, sino que residen en las denominadas **"memorias auxiliares"** o **"secundarias"**, tales como dispositivos magnéticos (discos duros, cintas), discos ópticos (CDROM, DVD), memorias permanentes de estado sólido (memorias flash USB), etc.

Estas "memorias" permiten guardar datos o disponer de ellos en el momento que sea necesario mediante una escritura o lectura respectivamente.

A modo de ejemplo, se supone que una empresa tiene los siguientes datos por cada empleado:

- ⇨ Nombre y apellido
- ⇨ Documento de identidad
- ⇨ Sueldo

Con ellos, se forma un registro al que se llamará **Empleado**.

Empleado		
Nom	**DNI**	**Suel**

Los campos corresponden a la información del nombre, documento de identidad y sueldo de la entidad empleado.
Si ahora se quisiera operar o manipular los datos de todos los empleados de la empresa, se debería disponer de los registros de todos ellos.
Ese conjunto de registros, es posible reunirlos en una estructura de almacenamiento permanente, denominando a ese conjunto **ARCHIVO**. Cada **archivo se individualizará por su nombre**. Dicho nombre seguirá las mismas reglas que se usa para nombres de variables. En el ejemplo propuesto, podría llamarse a ese archivo **EMPLE**. Por lo tanto, el archivo **EMPLE** estará formado por un conjunto de registros cuya estructura corresponde a la variable registro **Empleado**.

Definimos a un archivo de la siguiente manera:

Un archivo es un conjunto de registros con una estructura común y organizados para un propósito particular.

Un conjunto de archivos relacionados a su vez, constituye una **BASE DE DATOS**.

Organización de los datos de un archivo

Los archivos se organizan para su almacenamiento y acceso, según las necesidades de los procesos que los van a usar.
Existen en general 3 tipos de organizaciones:

1. *Secuencial*
2. *Directa*
3. *Secuencial con índice*

1. SECUENCIAL: cada registro, salvo el primero, tiene otro que le antecede y todos, excepto el último, uno que le sigue.
El orden en que aparecen, es el orden en que han sido escritos.
En este tipo de organización, por lo tanto, el orden físico coincide con el orden lógico.
Para acceder al registro **n** es necesario leer los **n-1** registros anteriores.

2. SECUENCIAL CON ÍNDICE: habrá una tabla con índices, que permitirá indicar la localización de un grupo de registros que se encuentran almacenados en forma secuencial. Por ejemplo, si la ubicación de los registros de aquellos empleados que empiezan con A se encuentra en la posición X, aquellos que empiezan con B en la posición Y, etc.; para acceder a ALVAREZ primero nos ubicamos en X (posición inicial) y de allí al apellido buscado.

3. DIRECTO: no necesariamente el orden físico de los registros se corresponden con un determinado orden lógico que nos propongamos.
Podemos, en este tipo de organización, acceder directamente a un registro, sin necesidad de leer los precedentes.
La información se coloca y se accede aleatoriamente mediante su posición, es decir indicando el lugar relativo que ocupa dentro del conjunto de posiciones posibles.

Incluyendo archivos en un programa.

Existen acciones que, independientemente del lenguaje de programación escogido, deben realizarse para el óptimo funcionamiento de los archivos.

Dichas acciones a realizar son:

a- *Crear una variable lógica que maneje el archivo físico en el disco:* Es decir, se tendrá almacenado (o se creará por primera vez) en la memoria secundaria, el archivo, el cual tendrá un nombre y una extensión, y estará almacenado en algún lugar, con lo cual tendrá además una ruta de acceso (root). Este archivo, será la ***variable física***, y se deberá conocer su nombre y ruta exactos para que cualquier programa pueda, sobre todo, usar la información almacenada en el mismo.
Pero, en el programa, cuando se procede a codificar y se desea usar un archivo, se utiliza lo que se denomina, una ***variable lógica***. Es decir, se da un nombre (siguiendo las pautas para cualquier nombre de variable) a una variable que hará referencia a ese archivo.
Dependiendo de cada lenguaje, variará la forma de vincular ambas variables, pero el concepto no varía.
Con esto, cada persona que programe, podrá utilizar el nombre de variable lógica que desee para hacer referencia al mismo archivo físico.

Ejemplo:

ARCHIVO FÍSICO	ARCHIVO LÓGICO
C:/PROGRAMAS/proyecto1/empleados.txt	Programador uno, lo llamará: ARCHIVO
	Programador dos, lo llamará: ARCHI
	Programador tres, lo llamará: fichero_a1

b- *Abrir el archivo:* El término *"Abrir",* se utiliza para dos conceptos: CREAR y OBTENER. Es decir, la acción ABRIR, permite decirle al complilador, qué es lo que se quiere hacer, si crear por primera vez el archivo, o abrirlo para obtener sus datos.
En resumen, se deberá especificar cómo será utilizado el archivo:
- Entrada de datos; o sea, recuperar desde memoria externa la información, mediante la lectura.
- Salida de información; o sea escribir en memoria externa los datos que deseamos almacenar.

Para ello, básicamente se debe hacer: **ABRIR <VAR_LOGICA> PARA E** o **ABRIR <VAR_LOGICA> PARA S.**

La indicación *para **E***, indica al procesador que se usará el archivo para la entrada de datos; *para **S***, le indica al procesador la salida de información.

La acción ABRIR debe figurar en el programa antes de realizar cualquier operación relacionada con ese archivo.

c- *Leer y Escribir en el archivo:* El término *"Leer",* se utiliza para ingresar los datos del archivo al programa. El término *"Escribir",* se utiliza para grabar los datos procesados en el archivo.
La sintaxis de estas acciones, varía mucho entre los diversos lenguajes de programación, con lo cual, sólo se dejará plasmada la idea central de dichas acciones.

d- *Cierre del archivo:* Luego de haber concluido con el procesamiento de un determinado archivo, se debe proceder a cerrarlo, o sea, la acción contraria a cuando se necesitaba comenzar a usarlo.
Básicamente la acción es: **Cerrar <VAR_LOGICA>**

ARCHIVOS (FICHEROS) EN C++

Los archivos se pueden clasificar atendiendo a diferentes criterios. En función de la codificación o formato en el que almacenan la información, los archivos se pueden clasificar en: archivos de Texto y Archivos Binarios. **En esta unidad trabajaremos con archivos de texto**.

En los archivos de texto, la información se almacena como una secuencia de caracteres y cada carácter se almacena utilizando una codificación estándar (usualmente basada en la codificación ASCII, UTF-8, etc). Al tratarse de un formato estandarizado, otros programas diferentes de aquel que creó el archivo podrán entender y procesar su contenido. Por ejemplo, un programa podría generar un archivo de texto con los datos de las personas de una agenda y posteriormente dicho archivo podría ser entendido y procesado por otros programas. Por ejemplo, podría ser visualizado y editado mediante programas de edición de textos de propósito general, tales como gedit, kate, gvim, emacs, etc. en Linux, textedit en MacOS-X y notepad en Windows, entre otros.

Flujos de Entrada y Salida Asociados a Ficheros

Un programa codificado en C++ realiza la entrada y salida de información a través de flujos (stream en inglés) de entrada y salida respectivamente. En unidades anteriores, se vió cómo realizar la entrada y salida de datos a través de los flujos estándares de entrada y salida (cin y cout, respectivamente), o flujos de E/S o I/O, usualmente conectados con el teclado y la pantalla de la consola. El mismo concepto explicado para E/S de los flujos estándares o la E/S de cadenas de caracteres también es aplicable a los flujos de E/S vinculados a archivos.

En el caso de entrada y salida a y desde archvios, C++ posee mecanismos para asociar y vincular estos flujos con ficheros almacenados en memoria secundaria en el sistema de archivos. De esta manera, toda la entrada y salida de información se realiza a través de estos flujos vinculados a archivos, denominados manejadores de archivos. De este modo, una vez que un programa vincula un archivo con un determinado flujo de entrada o salida, las operaciones de lectura o escritura funcionan como los flujos estándar cin y cout.

Cuando un programa quiere realizar una entrada o salida de datos con un determinado archivo, debe realizar las siguientes acciones:

1. Incluir la biblioteca **`<fstream>`**, que contiene los elementos necesarios para procesar el archivo.

2. Usar el espacio de nombres std. (**`using namespace std;`**)

3. Declarar las variables que actuarán como manejadores de archivos (Variables lógicas).

4. Abrir el flujo de datos, vinculando la variable correspondiente con el archivo especificado. Esta operación establece un vínculo entre la variable (manejador de archivo) definida en el programa y el archivo gestionado por el sistema operativo. De esta forma, toda transferencia de información entre el programa y un archivo se realizará a través de la variable manejador que ha sido vinculada con dicho archivo (vinculación de variable física con variable lógica).

5. Comprobar que la apertura del archivo del paso previo se realizó correctamente. Si la vinculación con el archivo especificado no pudo realizarse por algún motivo (por ejemplo, si se quiso hacer entrada de datos de un archivo que no existe, o si es imposible crear un archivo en el que escribir datos), entonces la operación de apertura fallaría.

6. Realizar la transferencia de información (de entrada o de salida) con el archivo a través de la variable de flujo vinculada al mismo. En el caso de salida, los datos deberán escribirse siguiendo un formato adecuado que permita su posterior lectura. Por ejemplo, escribiendo separadores adecuados entre los diferentes valores almacenados.
Normalmente, tanto la entrada como la salida de datos se realizan mediante un proceso iterativo. En el caso de entrada dicho proceso suele requerir la lectura de todo el contenido del archivo.

7. Comprobar que el procesamiento del archivo del paso previo se realizó correctamente. En el caso de procesamiento para entrada ello suele consistir en comprobar si el estado de la variable manejador indica que se ha alcanzado el final del fichero (EOF). En el procesamiento para salida suele consistir en comprobar si el estado de la variable manejador indica que se ha producido un error de escritura.

8. Finalmente, cerrar el flujo para liberar la variable manejador de su vinculación con el fichero. En caso de no cerrar el flujo, éste será cerrado automáticamente cuando termine el ámbito de vida de la variable manejador del fichero.

Nota: es importante tener en cuenta que cuando un flujo pasa al estado erróneo (fail()), entonces cualquier operación de entrada o salida que se realice sobre él también fallará.

Las variables de tipo flujo pueden ser usadas como parámetros de subprogramas. En este caso hay que tener en cuenta que es necesario que, tanto si el archivo se quiere pasar como un parámetro de entrada, salida o entrada/salida, dicho paso de parámetro debe hacerse por referencia (no constante).

Modos de apertura de archivo

Básicamente, lo primero que debe hacerse al desear abrir un archivo en un programa C++, es especificar qué tipo de apertura se realizará, si para E o para S.

Esto se efectúa de la siguiente manera:
a) Abrir archivo para SALIDA:

ofstream archi; //donde *archi*, es el nombre de la variable lógica

b) Abrir archivo para ENTRADA:

ifstream archi; //donde *archi*, es el nombre de la variable lógica

Dependiendo del propósito del programa que se quiera realizar, es posible que sea necesario agregar datos a los ya existentes en el archivo, o quizá sea necesario que las operaciones del programa no se lleven a cabo en caso de que el archivo especificado exista en el disco, etc.

Para éstos casos podemos especificar el modo de apertura del archivo incluyendo un parámetro adicional en la operación de apertura, cualquiera de los siguientes:

- **ios::app** Operaciones de añadidura.
- **ios::ate** Coloca el apuntador del archivo al final del mismo.
- **ios::in** Operaciones de lectura. Esta es la opción por defecto para objetos de la clase ifstream.
- **ios::out** Operaciones de escritura. Esta es la opción por defecto para objetos de la clase ofstream.
- **ios::nocreate** Si el archivo no existe se suspende la operación.
- **ios::noreplace** Crea un archivo, si existe uno con el mismo nombre la operación se suspende.
- **ios::trunc** Crea un archivo, si existe uno con el mismo nombre lo borra.
- **ios::binary** Operaciones binarias.

Utilizando los especificadores de modo de apertura se puede conseguir un mayor control en las operaciones de E/S en archivos.

Algunos ejemplos básicos de programas

```
// Ejemplo 1.
// Este programa escribe en un archivo la cadena "Hola mundo.".
#include <fstream>
#include <iostream>
#include <cstdlib>

using namespace std;
```

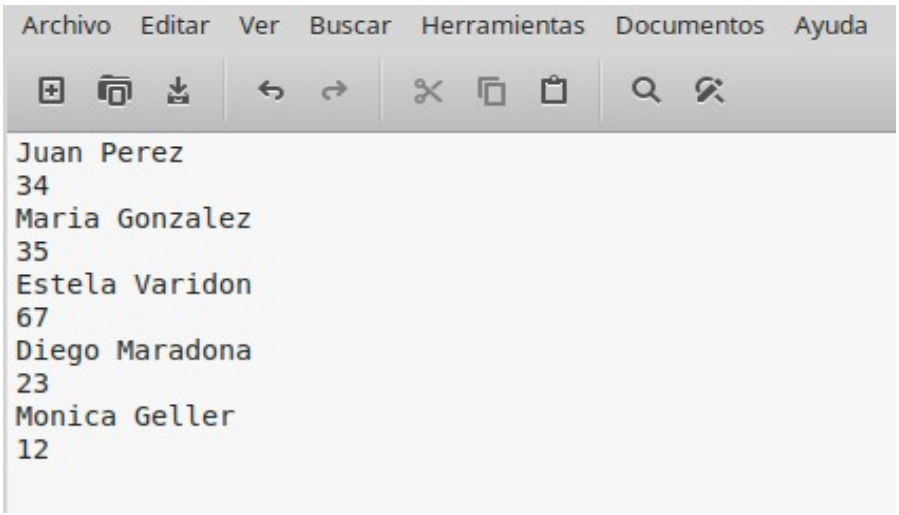
```
int main(int argc, char *argv[]) {
     ofstream archivo;   //archivo, es el nombre de la varible logica
                         //ofstream ABRE ARCHIVO PARA SALIDA
     archivo.open ("./prueba.txt");  //vinculacion con archivo físico
     if ( archivo.fail() ){
          cout << "Error en la apertura del archivo.";
     }
     else {
          archivo << "Hola mundo.";     //envía al archivo la cadena
     }
     archivo.close ();
     return 0;
}
```

```
// Ejemplo 2.
// Este programa lee de un archivo una cadena de texto.
// El archivo debe existir previamente y llamarse prueba.txt
#include <fstream>
#include <iostream>
#include <cstdlib>

using namespace std;
int main(int argc, char *argv[]) {
     ifstream archivo;  //archivo, es el nombre de la varible logica
                        //ifstream ABRE ARCHIVO PARA ENTRADA
     archivo.open ("./prueba.txt"); //vinculacion con archivo físico
     if ( archivo.fail() ){
          cout << "Error en la apertura del archivo.";
     }
     else {
          string linea;
          getline(archivo, linea);   //LEE DESDE EL ARCHIVO
          cout << "La cadena leida es:" << endl;
          cout << linea;
     }
     archivo.close ();
     return 0;
}
```

```
// Ejemplo 3.
// Este programa escribe en un archivo un listado de structs.
#include <fstream>
#include <iostream>
#include <cstdlib>
using namespace std;

struct alumno {          //CON ESTA ESTRUCUTRA SE GUARDARA
          string nombre;  // EL CONTENIDO EN EL ARCHIVO
          int edad;
};

int main(int argc, char *argv[]) {
     ofstream archivo;
     alumno lista[5];
     // El usuario carga el listado.
     for (int i=0; i<5; i++) {
          cout << "Nombre: ";
          getline(cin,lista[i].nombre);
          cout << "Edad: ";
```

```
            cin >> lista[i].edad;
            cin.ignore();

      }
      // Lo guardamos en el archivo.
      archivo.open ("./prueba_lista.txt");
      if ( archivo.fail() ) {
            cout << "Error en la apertura del archivo.";
      }
      else {
            for (int i=0; i<5; i++) {
              archivo << lista[i].nombre <<endl<< lista[i].edad <<endl;
            }
      }
      archivo.close ();
      return 0;
}
```

Si se abre el archivo en disco "prueba_lista.txt", se verá, por ejemplo, la siguiente información:




```
// Ejemplo 4.
// Este programa lee de un archivo un listado de structs.
// A priori no sabemos la cantidad de lineas que tiene el archivo.
// Pero si es necesario saber como estÃ¡n formadas las lineas para
poder realizar la lectura de forma consistente.
#include <fstream>
#include <iostream>
#include <cstdlib>
using namespace std;


struct alumno {          //CON ESTA ESTRUCUTRA SE GUARDARA
      string nombre;   // EL CONTENIDO EN EL ARCHIVO
      int edad;
};


int main(int argc, char *argv[]) {
      ifstream archivo;
```

```
        alumno lista[10];
        int cuantos = 0;
        archivo.open ("prueba_lista.txt");
        if ( archivo.fail() ) {
              cout << "Error en la apertura del archivo.";
        }
        else {
              getline(archivo,lista[cuantos].nombre);
              // Mientras NO se encuentre el final del archivo...
              while ( !archivo.eof() ) {
                    archivo >> lista[cuantos].edad;
                    archivo.ignore();
                    cuantos++;
                    getline(archivo,lista[cuantos].nombre);
              }
        }
        for (int i=0; i<cuantos; i++) {
              cout << "Nombre: ";
              cout << lista[i].nombre << "   ";
              cout << "Edad: ";
              cout << lista[i].edad << endl;
        }
        archivo.close ();
        return 0;
}
```

Algunas consideraciones a tener en cuenta a la hora de trabajar con archivos…

- C++ trabaja con “flujos” de entrada y de salida. El flujo de entrada por defecto es el teclado y el flujo de salida por defecto es la pantalla. Los archivos se manejan en la misma forma de flujo. Al abrir un archivo para lectura se crea un flujo de entrada para recibir la información del archivo. Al abrir un archivo para escritura, se crea un flujo de salida para almacenar la información en el archivo.
- Así como se trabaja con **cin** y **getline** para el ingreso de datos por teclado, de la misma forma se trabajará con un archivo abierto para lectura. Se podrán combinar cin y getline para la correcta lectura de datos int, char, float (leídos con cin) y datos string (leídos con getline).
- Para la correcta lectura de datos de diferente tipo, el uso de cin.ignore() es útil tanto en las lecturas de teclado como en las lecturas del archivo. (*ver ejemplos 3 y 4*)
- El éxito de la correcta lectura de un archivo depende del conocimiento que se tenga de como fue escrito el archivo, es decir: de que tipo es la información que se va a leer y cuales son los delimitadores utilizados.
- Además del uso de delimitadores por defecto (espacios y retornos de carro para los datos numéricos y retornos de carro para las cadenas) puede guardarse información utilizando otros limitadores (como ; en los archivos .csv o “|” llamado pipe).