1. Flujos de datos (*streams*)

Desde el principio se han estado utilizando flujos de caracteres, de entrada (`cin`) y salida (`cout`) estándar. Aquí se hará un pequeño análisis para clarificar un poco su utilización.

En principio, existen tres tipos de Entrada/Salida **estándar** (`std`), estos son: la *salida estándar* (pantalla), *entrada estándar* (teclado) y *errores estándar* (también por la pantalla), estos suelen ser nombradas como `stdout`, `stdin` y `stderr` respectivamente. Como ya se ha mencionado, es posible acceder a ellos a través de variables del tipo *flujo* (`stream`) llamadas `cout`, `cin` y `cerr`. Sin embargo, no son los únicos flujos con los que se puede trabajar.

Las variables del tipo flujo utilizan el **operador de inserción** (<<) y el **operador de extracción** (>>) los cuales son operadores tales como la suma (+), la resta (-) o cualquier otra, pero tienen un comportamiento específico, que es el de leer o escribir datos hacia o desde un tipo de "fuente de datos". Los flujos o *streams*, pueden ser de entrada (`istream`), salida (`ostream`) o ambos (`iostream`) dependiendo del tipo de dispositivo al cual se accede a través de éste. Así, por ejemplo, si se tratara de un archivo de texto en el disco, este podría ser bidireccional, es decir, que podemos leer y escribir en el mismo.

1.1. Operador de extracción

El operador de extracción (>>) está diseñado de una forma tal que, según el tipo de dato, será tomado del *stream* y almacenado en una variable, terminando su lectura cuando encuentra un carácter de separación (espacio, tabulación, nueva linea, etc). Analice la siguiente situación, suponga que el *stream* contiene los siguientes datos, donde la ↑ indica la posición de la próxima extracción.

```
1 2↲
↑
3 4 5 6
```

Si se ejecuta un `cin >> i;` donde `i` es una variable del tipo entero, entonces el `1` quedará almacenado en `i` y ↑ apuntando al espacio vacío.

```
 2↲
↑
3 4 5 6
```

En una próxima extracción (`cin >> i;`), el espacio será ignorado y el `2` se almacenará en `i`, quedando ↑ apuntando a la nueva linea (`'\n'`).

```
↲
↑
3 4 5 6
```

Si se hacen lecturas consecutivas (`cin >> a >> b >> c;`), entonces, cada una de estas lecturas almacenarán los valores 3, 4 y 5 respectivamente, ignorando el carácter de nueva

linea y todos los espacios encontrados:

```
 6
↑
```

Es por ello que al utilizar `getline()` luego del operador de extracción, podría dar como resultado un error de lectura. Ya que esta función lee todos los caracteres hasta encontrar una *nueva línea* (`'\n'`) el cual pudo haber quedado inmediatamente después de una lectura con el operador <<.

```
16↲
↑
Matias Avalos↲
```

```
1 int a;
2 cin >> a;
3 string nombre;
4 getline(cin, nombre);
5 cout << a << " - " << nombre;
```

Teniendo en cuenta el *stream* y el fragmento de código, el resultado por pantalla tras ejecutarse sería el siguiente:

```
 16 -
```

Se procederá a hacer el análisis de lo ocurrido. Luego de ejecutarse la *línea 2*. El *stream* de entrada quedará de la siguiente manera habiendo almacenado en `'a'` el valor `16`:

```
↲
↑
Matias Avalos↲
```

Ahora al ejecutarse la *línea 4*, se encuentra automáticamente con una *nueva línea*. Es decir, que se almacenará en `'nombre'` la cadena vacía. Generando así el resultado expuesto anteriormente.

Como se ha expuesto en el *Apunte N°07*, es aconsejable utilizar siempre `getline()` y luego las funciones `atoi()` y `atof()` para convertir y validar la entrada de datos por parte de `cin`.

1.2. Función `get()`

La función `get()` es utilizada, habitualmente, para obtener un solo carácter del *stream*. En el siguiente ejemplo se utiliza para "simular" un comportamiento similar al de `getline()`:

```
1 string leer_linea() {
2   string out = "";
3   char c;
```

```
4     cin.get(c);          // Se lee el primer carácter
5     while(c != '\n') { // Mientras el carácter no sea un ENTER
6       out += c;          // Se acumula el carácter en el string
7       cin.get(c);        // Se lee el siguiente carácter...
8     }
9     return out;
10  }
```

A través de `get()` en la *línea 4*, se obtiene el primer carácter a extraer del *stream*, luego es comparado con el carácter de salto de línea (*en 5*), dado el caso el carácter es anexado al *string* originalmente vacío y se pasa a leer el próximo carácter (*línea 7*). Así sucesivamente hasta encontrar el `'\n'`.

2. Manipulación de archivos

Hasta el momento, todos los programas que se han expuesto y desarrollado almacenan sus datos en RAM. Es decir, al finalizar la ejecución del software todo es *desalocado* de la memoria. Es deseable, en la mayoría de los sistemas, poder contar con un mecanismo de **persistencia** de datos. Esto puede ser logrado de muchas formas y en muchos formatos distintos, popularmente –aunque no necesariamente– en una *Base de Datos*.

En esta oportunidad se expondrá cómo almacenar y leer datos en **archivos de texto plano**. Es decir, que podremos guardar información en documentos que podrían ser modificados externamente a través de un simple editor. Y a su vez, poder recuperar estos datos desde el mismo programa.

2.1. Apertura y cierre

En C++ los archivos son abiertos como *streams* y tienen las mismas herramientas anteriormente analizadas. Pero antes de poder acceder a estos, debemos abrir o crear un archivo ya sea para lectura y/o escritura declarando una variable del tipo `ifstream` (*input-file stream*), `ofstream` (*output-file stream*) o simplemente `fstream` (*file stream* para entrada-salida) todas ellas dentro de la biblioteca `'fstream'`. Para poder luego utilizar la función `open()`.

Es posible verificar si no hubo problemas al intentar abrir el archivo utilizando la función `good()`. Y, una vez terminada su utilización, debemos siempre cerrar el archivo con la función `close()`.

Programa 1: Abrir y cerrar un archivo de texto.

```
1  #include <iostream>
2  #include <fstream>
3
4  using namespace std;
5
6  int main() {
7    fstream archivo;
8    archivo.open("Hola.txt");
```

```
9     if(archivo.good()) {
10      cout << "'Hola.txt' fue abierto correctamente.\n";
11    } else {
12      cout << "'Hola.txt' no pudo ser abierto correctamente.\n";
13    }
14    entrada.close();
15  }
```

Sería siempre conveniente, dado el caso en que el archivo cumpla una función primordial en un software, terminar abruptamente la ejecución del mismo en caso de no poder abrirlo. Para ello podemos usar la función `exit()` proporcionada por la biblioteca `cstdlib`:

```
if(not entrada.good()) {
  cerr << "No se pudo abrir correctamente el archivo X...\n";
  exit(1); // valor distinto de 0 para indicar una falla...
}
// Resto del programa...
```

2.2. Escritura

Para únicamente escribir en un archivo se utilizará `ofstream` y luego simplemente podemos emplear el operador de inserción (<<):

Programa 2: Escribir en un archivo de texto.

```
1  #include <iostream>
2  #include <fstream>
3  #include <cstdlib>
4
5  using namespace std;
6
7  int main() {
8    string ingresoUsuario;
9    cout << "Escriba un texto: ";
10   getline(cin, ingresoUsuario);
11   ofstream aSalida;  // output file stream
12   // Se abre el archivo
13   aSalida.open("Hola.txt");
14   if(not aSalida.good()) {
15     cerr << "Error al intentar crear 'Hola.txt'.\n";
16     exit(1); // termina el programa...
17   } // sino...
18   // Se escribe en el archivo 'Hola.txt' el ingreso del usuario:
19   aSalida << ingresoUsuario << endl;
20   aSalida.close();
21 }
```

Cuando un archivo del tipo `ofstream` se abre (con la función `open()`) este es creado, de existir, pisará el anterior borrando todo su contenido y reemplazándolo por el nuevo.

A esta última operación se la conoce como **truncar**. Si se quisiera agregar información al final del archivo, es decir, **anexar** nuevo contenido, puede pasar un segundo argumento a `open()` con el valor `ofstream::app` (*append*) quedando:

```
aSalida.open("Hola.txt", ofstream::app);
```

Existen alternativas al operador de inserción, como por ejemplo la función `put()`. Ésta permite escribir un solo carácter al *stream*.

```
char a = 'A';
aSalida.put(a); // escribe una A (mayúscula) en el archivo...
```

2.3. Lectura

En el caso de que utilicemos un archivo de solo lectura, podemos hacer uso de `ifstream` que soporta el operador de extracción (<<), `get()` y `getline()` al igual que `cin`, por lo tanto es como leer del teclado.

Contenido en 'Hola.txt'

```
10↲
¡Hola, mundo!↲
```

Programa 3: Lectura de un arhivo de texto.

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
int main() {
  ifstream aEntrada;
  aEntrada.open("Hola.txt");
  if(not aEntrada.good()) {
    cerr << "No se pudo abrir el archivo 'Hola.txt'\n";
    exit(1); // termina el programa con error...
  } // sino...
  string lineaTexto;
  // Se lee el "10" (1ra línea):
  getline(aEntrada, lineaTexto);
  int n = atoi(lineaTexto.c_str()); // pasar a int

  // Se lee "¡Hola, mundo!" (2da línea):
  getline(aEntrada, lineaTexto);
  // Se muestra 'n' veces el texto de la 2da línea:
  for(int i=0; i < n ; i++) {
    cout << lineaTexto << '\n';
  }
  aEntrada.close();
}
```

En el Programa 3, se lee de un archivo de 2 líneas donde *se repite tantas veces la segunda línea como indica en la primera*. Esto se logra leyendo primeramente del archivo `'Hola.txt'` la primer línea utilizando `getline()` (*en 14*) y luego la segunda con el mismo método (*en 18*) y guardando lo leído temporalmente en la variable `lineaTexto`.

2.3.1. Fin del archivo (*End-of-File*)

Los archivos de texto contienen un carácter especial que indica cuando se ha alcanzado el final del mismo. Este es útil cuando no se sabe qué tan largo es el archivo. Es posible comprobar si se ha alcanzado a través de la función miembro `eof()`.

Programa 4: Copiar texto de un archivo a otro todo en mayúsculas.

```
1  #include <iostream> // cerr, cout
2  #include <fstream>  // ifstream, ofstream
3  #include <cctype>   // toupper()
4  #include <cstdlib>  // exit()
5
6  using namespace std;
7
8  int main() {
9    // Se abre el archivo de entrada (debe existir)
10   ifstream archEntrada;
11   archEntrada.open("entrada.txt");
12   if(not archEntrada.good()) {
13   cerr << "No se pudo abrir achivo de entrada.\n";
14   exit(1); // Termina el programa...
15   }
16   // Se crea el archivo de salida
17   ofstream archSalida;
18   archSalida.open("salida.txt");
19   if(not archSalida.good()) {
20     cerr << "No se pudo crear el archivo de salida.\n";
21     exit(1); // Termina el programa...
22   }
23
24   cout << "Copiando archivo en mayúsculas...\n";
25   char c;
26   archEntrada.get(c); // Extracción del primer carácter.
27
28   // Mientras no se alcance el final del archivo:
29   while(not archEntrada.eof()) {
30     archSalida.put(toupper(c)); // Se escribe la mayúscula.
31     archEntrada.get(c);         // Próximo carácter.
32   }
33   cout << "Copia completa.\n";
34
35   archEntrada.close();
36   archSalida.close();
37 }
```

En este último ejemplo se ha optado por escribir utilizando la función `put()`, pero se obtendría el mismo resultado utilizando el operador de inserción con un `char` (`archSalida << c;`).

2.4. *Streams* genéricos (`ios`)

Todos los *streams*, tanto los `istream`, `ostream` e `iostream` (utilizados por `cin, cout, cerr`) como los de archivos `ifstream`, `ofstream` y `fstream` son a su vez `ios` (*input-output streams*). Es por ello que sin importar cual de estos tipos de datos sea, siempre podremos tratarlos como si se trataran de este tipo de dato. Si se quisiera encapsular el chequeo al abrir un archivo (sin importar si es de entrada o salida), podemos hacer lo siguiente:

```
void comprobar_apertura_archivo(ios &archivo) {
  if(not archivo.good()) {
    cerr << "Error de Entrada/Salida.\n";
    exit(1); // Termina el programa...
  }
}
```

2.5. Borrado, copiado y renombrado de archivos

En la biblioteca estándar `filesystem` contamos con muchas funciones que podrían llegar a ser de utilidad en el manejo de archivos y carpetas, entre ellas están `copy()`, `exists()`, `remove()` y `rename()`.

Podemos hacer uso de estas funciones a través del espacio `filesystem`:

```
#include <filesystem>
// ...

// Se borra el archivo 'copia.txt'
filesystem::remove("copia.txt");

// 'copia.txt' no debe existir o causará un error
filesystem::copy("original.txt", "copia.txt");

// renombra el archivo 'nombre_original.txt' a 'nuevo_nombre.txt'
filesystem::rename("nombre_original.txt", "nuevo_nombre.txt");

if(filesystem::exists("archivo.txt")) {
  cout << "El archivo 'archivo.txt' existe.";
} else {
  cout << "No existe el archivo 'archivo.txt'.";
}
```

Al disponer de estas herramientas es posible hacer software más sofisticado y robusto, que esté preparado para diversos escenarios:

Programa 5: Copia a mayúsculas con validaciones y copias.

```cpp
#include <iostream>   // cerr, cout
#include <fstream>    // ifstream, ofstream
#include <cctype>     // toupper()
#include <cstdlib>    // exit()
#include <filesystem> // exists(), copy(), rename(), remove()
using namespace std;

string pedir_texto(string msj);
void comprobar_apertura_archivo(ios &archivo);
void comprobar_y_copiar(string nom_archivo);
bool obtener_si_o_no(string msj);

int main() {
  string nom_entrada=pedir_texto("Archivo de entrada: ");
  // Se abre el archivo de entrada:
  ifstream archEntrada;
  archEntrada.open(nom_entrada);
  // Si no existe termina el programa...
  comprobar_apertura_archivo(archEntrada);

  string nom_salida = pedir_texto("Archivo de salida: ");
  // Si un archivo con el mismo nombre existe, se ofrece copiarlo
  comprobar_y_copiar(nom_salida);

  // Se crea/pisa el archivo de salida con el nombre elegido
  ofstream archSalida;
  archSalida.open(nom_salida);
  comprobar_apertura_archivo(archSalida);

  cout << "Copiando del archivo '" + nom_entrada + "'...\n";
  char c;
  archEntrada.get(c); // Primer carácter del archivo.

  // Mientras no se alcance el final del archivo:
  while(not archEntrada.eof()) {
    archSalida.put(toupper(c)); // Se escribe la mayúscula.
    archEntrada.get(c);         // Próximo carácter.
  }
  cout << "Copia completa en '" + nom_salida + "'.\n";

  archEntrada.close();
  archSalida.close();
}

void comprobar_apertura_archivo(ios &archivo) {
  if(not archivo.good()) {
    cerr << "Error de Entrada/Salida.\n";
  exit(1); // Termina el programa...
  }
}
```

```
51
52 string pedir_texto(string msj) {
53   string texto;
54   do {
55     cout << msj;
56     getline(cin,texto);
57   } while (texto == "" or isspace(texto[0]));
58   return texto;
59 }
60
61 bool obtener_si_o_no(string msj) {
62   string si_no = pedir_texto(msj);
63   while(toupper(si_no[0]) != 'S' and toupper(si_no[0]) != 'N') {
64     cout << "Error: Complete Sí o No\n\n";
65     si_no = pedir_texto(msj);
66   }
67   return toupper(si_no[0]) == 'S';
68 }
69 void comprobar_y_copiar(string nom_orig) {
70   if(filesystem::exists(nom_orig)) {
71     bool pisar =
72       obtener_si_o_no(nom_orig+" ya existe,¿crear copia?(S/N) ");
73     if(pisar) {
74       bool copiado = false;
75       while(not copiado) {
76         string nom_copia =
77                     pedir_texto("Ingrese nombre para la copia: ");
78         if(filesystem::exists(nom_copia)) {
79           cout << "El archivo ya existe, elija otro nombre...\n";
80         } else {
81          filesystem::copy(nom_orig, nom_copia);
82          cout << "\nSe creo la copia de " + nom_orig + " en "
83                   + nom_copia + "\n\n";
84           copiado = true;
85         }
86       }
87     }
88   }
89 }
```

Una posible salida de este programa podría ser como se muestra a continuación, teniendo en cuenta el siguiente archivo de entrada:

Contenido en 'entrada.txt'

```
Esto es un archivo de prueba↲
con multiples lineas↲
el cual sera copiado↲
en un nuevo archivo↲
con todas las letras en↲
mayuscula.↲
```

```
Archivo de entrada: entrada.txt
Archivo de salida: salida.txt
salida.txt ya existe,¿crear copia?(S/N) d
Error: Complete Sí o No

salida.txt ya existe,¿crear copia?(S/N) s
Ingrese nombre para la copia: salida_anterior.txt

Se creo la copia de salida.txt en salida_anterior.txt

Copiando del archivo 'entrada.txt'...
Copia completa en 'salida.txt'.
```

Se obtendría el resultado:

Contenido en `'salida.txt'`

```
ESTO ES UN ARCHIVO DE PRUEBA↲
CON MULTIPLES LINEAS↲
EL CUAL SERA COPIADO↲
EN UN NUEVO ARCHIVO↲
CON TODAS LAS LETRAS EN↲
MAYUSCULA.↲
```

3. Ejercitación

Codifique en C++ los siguiente programas utilizando arreglos, registros y archivos.

1. Desarrolle un software que pida un nombre para un archivo de texto y permita ingresar y almacenar en dicho archivo todo lo que se escriba a continuación hasta que el usuario ingrese la cadena `'FinalizarIngreso'` (con `'F'` e `'I'` en mayúscula).

2. Desarrolle un software que sea capaz de leer datos de hasta 20 equipos de futbol para poder generar una tabla de posiciones con la siguiente información:

   - Nombre del equipo
   - Total de partidos jugados
   - Cantidad de partidos ganados
   - Cantidad de partidos empatados
   - Cantidad de partidos perdidos
   - Cantidad de goles a favor
   - Cantidad de goles a en contra
   - Diferencia de goles entre los a favor y en contra
   - Puntos totales

El software debe mostrar los equipos en orden descendente según la cantidad de puntos que tengan, sabiendo que los puntos son: 3 por partido ganado, 1 por empate y 0 por perdidos. Los datos necesarios para generar la tabla están almacenados en un archivo de texto plano `'liga.dat'` con el siguiente formato:

```
Cangallo FC
6
2
2
8
4

Cebollitas
4
1
5
4
7
```

Donde **línea 1:** nombre del equipo; **línea 2:** partidos ganados; **línea 3:** partidos empatados; **línea 4:** partidos perdidos; **línea 5:** goles a favor; **línea 6:** goles en contra. Una línea vacía indica una separación entre equipos.

3. Programe una agenda donde pueda ingresar nombres, apellidos, fecha de cumpleaños, e-mail y teléfono de hasta 100 personas. Guarde sus datos en un archivo de texto `'agenda.dat'` del que pueda recuperar los datos en cada ejecución y agregar más personas.

4. Al punto anterior agregue la posibilidad de modificar los *registros* ya existentes.