Manejo de Archivos

Prof. Andrés Arcia

Entrada/Salida de Archivos

- El sistema de streams en C++ tiene establecido un rango de 0 a longitud-1 para las posiciones de un archivo.
- Un archivo tiene dos operaciones básicas: abrir (open) y cerrar (close).

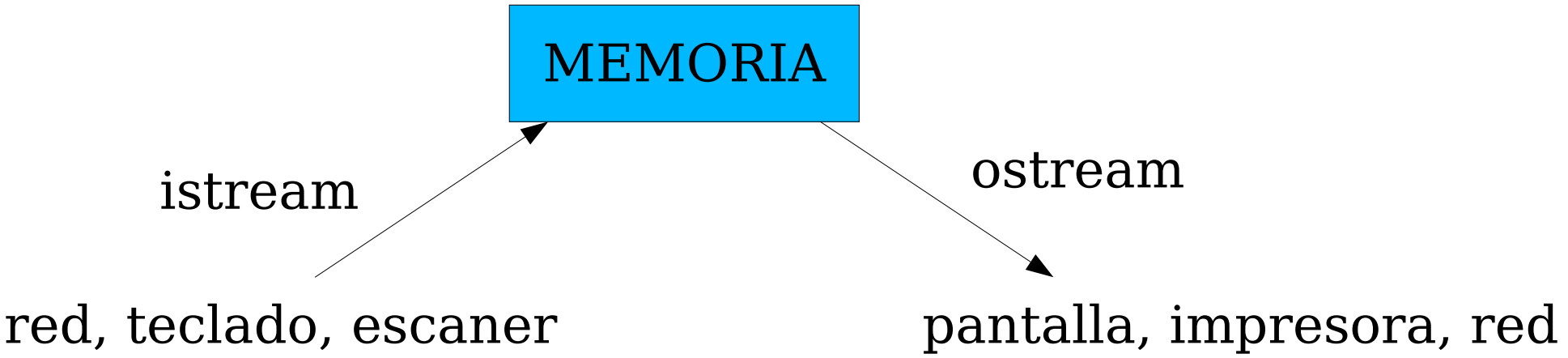
void open(const char *, openmode)

openmode:

in (entrada), out (salida), ate (posición inicial al final del archivo), app (toda salida al final), trunc (borrar lo que existe) y binary (modo binario).

Maneras de Abrir un Archivo

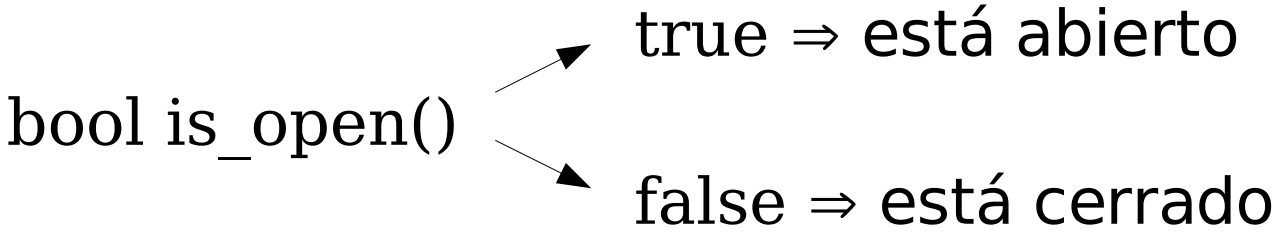
```
ifstream miArchivo(“data”, ios::in | ios::binary);
ofstream miOtroArchivo;
miOtroArchivo.open(“data2”, ios::out | ios::binary);
fstream foo;
foo.open(“data3”, ios::in | ios::out | ios::binary);
```



Modos por Omisión

ofstream	ios::out | ios::trunc
ifstream	ios::in
fstream	ios::in | ios::out

¿Cómo saber si un archivo está abierto?



Cerrar un Archivo

- Para vaciar el buffer y cerrar un archivo:
  - void close()
- Un objeto del tipo stream puede ser utilizado para abrir un archivo diferente, pero solo si el anterior ha sido cerrado:

```
ifstream bar(“arch1”);
bar.close();
bar.open(“arch2”);
... // trabajar con “arch2” ...
```

Medidas de Seguridad

- Una buena práctica es siempre preguntar si el archivo está abierto; al estilo de programación defensiva:

```
if (archivo.is_open())
        { // procedimiento normal }
else
        { // arrojar una excepción
          // mensaje de error a cerr ó cout
          // salida con valor entero (exit) }
```

Banderas que Indican Estado

bad(): true si ocurre un error de lectura ó escritura.

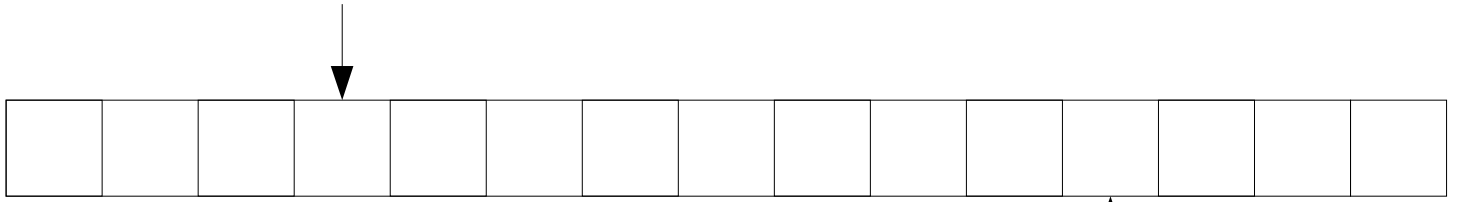
fail(): true en los mismos casos de bad pero detecta errores de formato. Por ejemplo, se espera un float y se recibe un char.

eof(): true cuando se alcanza el fin de archivo.

Si algunas de estas funciones resulta verdad, luego las banderas se pueden devolver al estado original utilizando: clear().

Flujo de Datos

puntero get
(próximo a obtener)



puntero put
(próximo a sobrescribir)

fstream = get + put

Los apuntadores put y get son independientes, pueden apuntar a cualquier parte del archivo.

Lectura de Lineas en Modo Texto

- Normalmente, cuando se obtienen datos de un archivo, se hace a través de cada tipo de dato.
- Para obtener una línea completa de texto, se hace con la funcion ifstream::getline(char *, size_t)
- size_t es el tipo del tamaño máximo del buffer de memoria, que queda como responsabilidad del usuario.

Posiciones de los Cursores

- Para observar la posición de los cursores virtuales de un archivo:
  - tellg(): dice donde está el apuntador get.
  - tellp(): dice donde está el apuntador put.
- Para mover los cursores:
  - seekg(pos_type pos): mueve una posición absoluta desde el principio del archivo al puntero get.
  - seekp(pos_type pos): mueve una posición absoluta desde el principio del archivo al puntero put.

Posiciones de los Cursores

- Los posicionamientos con seekg y seekp de un solo parámetro son recomendados para archivos texto.
- Para archivos binarios se recomienda:
  - seekp(off_type pos, seekdir dir)
  - seekg(off_type pos, seekdir dir)

Posiciones de los Cursores

- Según el seekp y el seekg de dos parámetros, los punteros se mueven, según las siguientes direcciones:
  - ios::beg desde el principio
  - ios::cur desde la posición actual
  - ios::end desde el final

Archivos Binarios

- Las funciones: getline, <<, >>, etc. no tienen mucho sentido en los archivos binarios.
- Los operadores que tienen sentido:
  - ostream & write(char *, streamsize)
  - istream & read(char *, streamsize)
- Hay que recordar que la responsabilidad de la memoria (char *), la maneja el usuario.

Sincronía entre el Buffer y el Disco

streamBuf:

Cuando se invoca una función put, hay que tomar en cuenta que no todas las operaciones las hace en memoria.

Hay sincronía cuando:

- Cuando se cierra el archivo
- Cuando el buffer se llena
- Explicitamente: flush, endl, sync.

Sincronía entre el Buffer y el Disco

- Al leer con read podemos pasarnos del fin de archivo?
- R: Si es posible.
- Para resolver este problema, utilizamos la función gcount(), que nos dice cuantos bytes fueron leídos durante la última lectura.