

---

---

# Entrada/Salida

---

---

---

# Entrada Salida desde Consola

---

# Entrada Salida desde Consola

**cout:** console out: Imprime por pantalla un dato.

El operador de inserción ( << ) inserta el flujo de datos en la salida estándar (la pantalla)

```
1 #include <iostream>
2
3 int main() {
4     std::cout << "Hola Mundo" << std::endl;
5     return 0;
6 }
```

**cin:** console in: Lee un dato del teclado

El operador de extracción ( >> ) extrae el flujo de datos de la entrada estándar (teclado)

```
1 #include <iostream>
2
3 int main() {
4     char letra;
5     std::cin >> letra;
6     return 0;
7 }
```

---

Entrada Salida desde Archivos

---

---

# Entrada Salida desde archivos

Escribir y leer texto desde un archivo de texto plano en C++ es similar a escribir y leer texto por consola. Usamos:

- operador de inserción (<<) para guardar texto en un archivo.
- operador de extracción (>>) para extraer texto de un archivo.

---

# Stream

## **cin y cout son streams (flujos)**

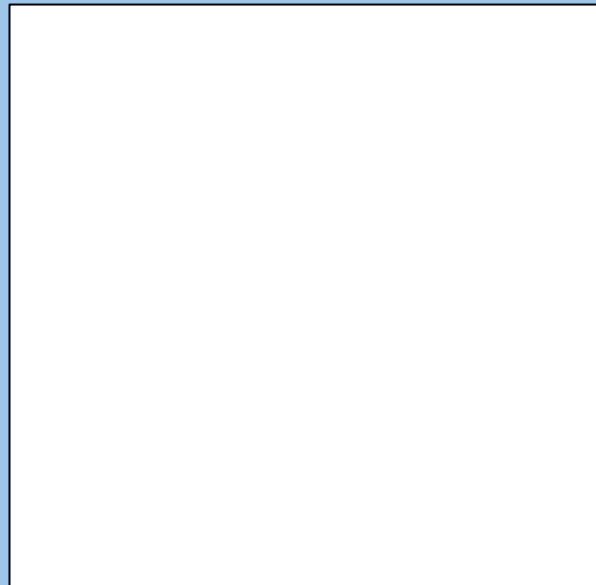
- Hay dos tipos principales de streams:
  - input stream: flujo de datos que representa una fuente de entrada (Ej: cin, ifstream).
  - output stream: flujo de datos que representa un destino de salida (Ej: cout, ofstream)

---

# Output file stream (ofstream)

```
1 #include <iostream>
2 #include <fstream>
3
4 int main() {
5
6
7
8
10
11
12
13}
```

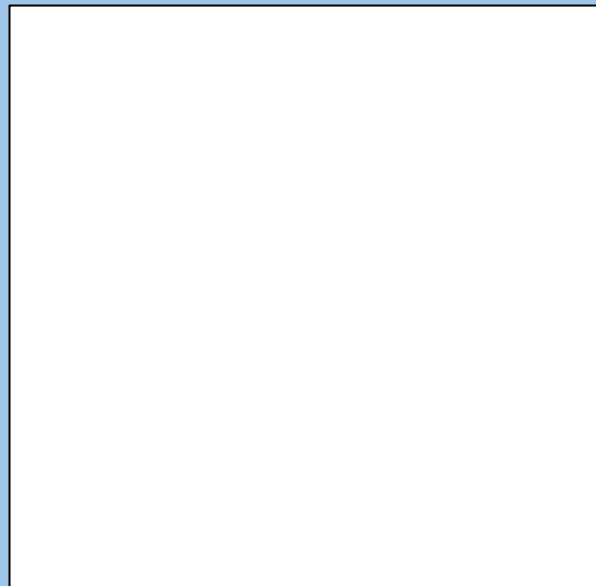
mi\_archivo.txt



# Output file stream (ofstream)

```
1 #include <iostream>
2 #include <fstream>
3
4 int main() {
5     std::ofstream fout;
6
7
8
9
10
11
12
13 }
```

mi\_archivo.txt

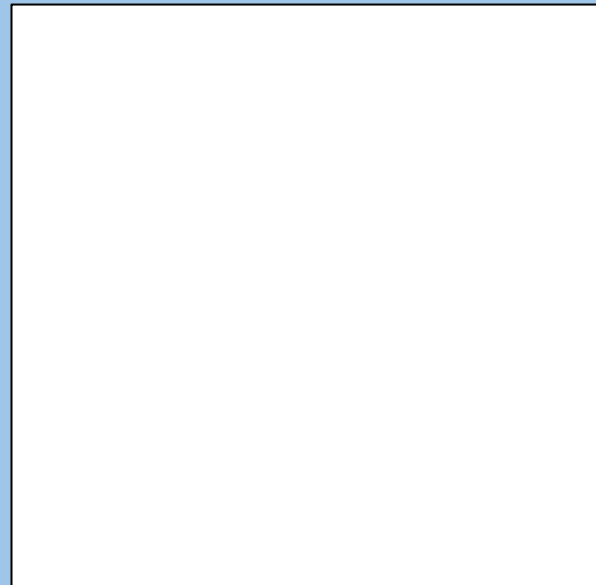




# Output file stream (ofstream)

```
1 #include <iostream>
2 #include <fstream>
3
4 int main() {
5     std::ofstream fout;
6     fout.open("mi_archivo.txt");
7
8
9
10
11
12
13 }
```

mi\_archivo.txt



# Output file stream (ofstream)

```
1 #include <iostream>
2 #include <fstream>
3
4 int main() {
5     std::ofstream fout;
6     fout.open("mi_archivo.txt");
7     fout << 89 << std::endl;
8
9
10
11
12
13 }
```

mi\_archivo.txt

89



# Output file stream (ofstream)

```
1 #include <iostream>
2 #include <fstream>
3
4 int main() {
5     std::ofstream fout;
6     fout.open("mi_archivo.txt");
7     fout << 89 << std::endl;
8     fout << 6 << std::endl;
9
10
11
12
13}
```

mi\_archivo.txt

89  
6




# Output file stream (ofstream)

```
1 #include <iostream>
2 #include <fstream>
3
4 int main() {
5     std::ofstream fout;
6     fout.open("mi_archivo.txt");
7     fout << 89 << std::endl;
8     fout << 6 << std::endl;
10    fout << 25 << std::endl;
11
12
13}
```

mi\_archivo.txt

89  
6  
25



# Output file stream (ofstream)

```
1 #include <iostream>
2 #include <fstream>
3
4 int main() {
5     std::ofstream fout;
6     fout.open("mi_archivo.txt");
7     fout << 89 << std::endl;
8     fout << 6 << std::endl;
10    fout << 25 << std::endl;
11    fout.close();
12    return 0;
13}
```

mi\_archivo.txt

```
89
6
25
```

---

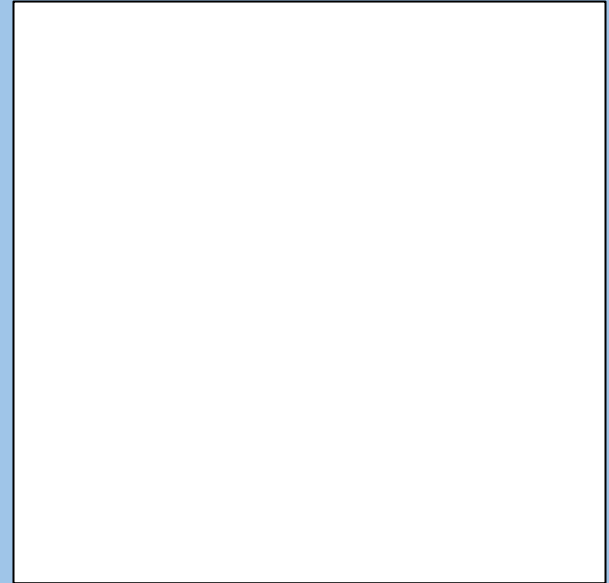
# Escribir valores de distintos Tipos de datos

- Hasta ahora escribimos únicamente enteros (int)
- También podemos escribir valores bool, float, char, etc.

# Escribir valores de distintos Tipos de datos

```
1 include <iostream>
2 #include <fstream>
3
4 int main() {
5     char c = 'x';
6     float f = 1.5;
7     bool b = true;
8
9
10
11
12
13
14
15
16}
```

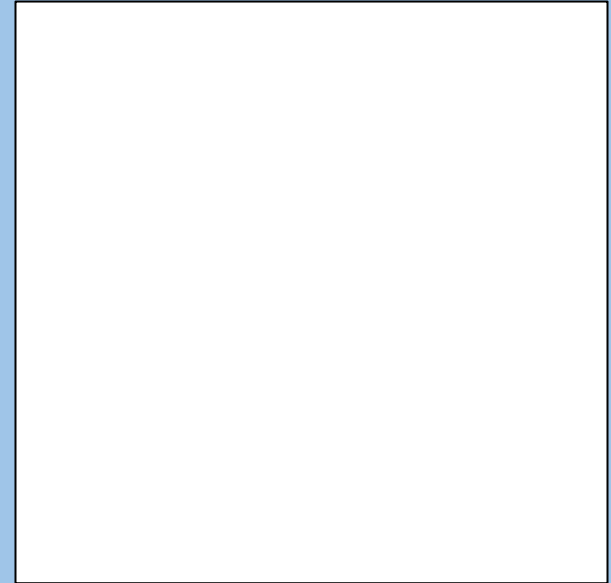
datos.txt



# Escribir valores de distintos Tipos de datos

```
1 include <iostream>
2 #include <fstream>
3
4 int main() {
5     char c = 'x';
6     float f = 1.5;
7     bool b = true;
8     std::ofstream fout;
9     fout.open("datos.txt");
10
11
12
13
14
15
16}
```

datos.txt





# Escribir valores de distintos Tipos de datos

```
1 include <iostream>
2 #include <fstream>
3
4 int main() {
5     char c = 'x';
6     float f = 1.5;
7     bool b = true;
8     std::ofstream fout;
9     fout.open("datos.txt");
10    fout << c << " ";
11
12
13
14
15
16}
```

datos.txt

X



# Escribir valores de distintos Tipos de datos

```
1 include <iostream>
2 #include <fstream>
3
4 int main() {
5     char c = 'x';
6     float f = 1.5;
7     bool b = true;
8     std::ofstream fout;
9     fout.open("datos.txt");
10    fout << c << " ";
11    fout << f << " ";
12
13
14
15
16}
```

datos.txt

x 1.5

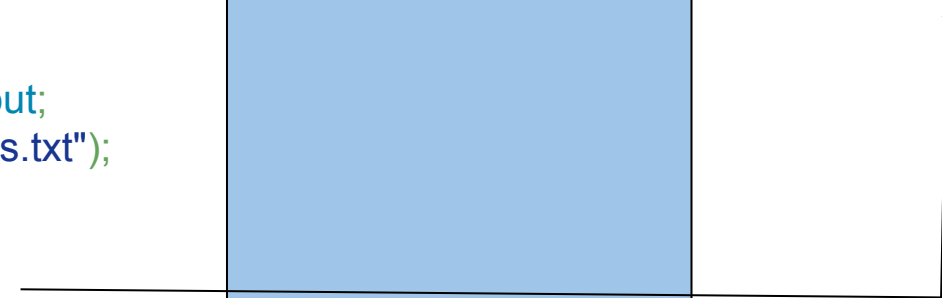


# Escribir valores de distintos Tipos de datos

```
1 include <iostream>
2 #include <fstream>
3
4 int main() {
5     char c = 'x';
6     float f = 1.5;
7     bool b = true;
8     std::ofstream fout;
9     fout.open("datos.txt");
10    fout << c << " ";
11    fout << f << " ";
12    fout << b << " ";
13
14
15
16}
```

datos.txt

x 1.5 1



# Escribir valores de distintos Tipos de datos

```
1 include <iostream>
2 #include <fstream>
3
4 int main() {
5     char c = 'x';
6     float f = 1.5;
7     bool b = true;
8     std::ofstream fout;
9     fout.open("datos.txt");
10    fout << c << " ";
11    fout << f << " ";
12    fout << b << " ";
13    fout.close();
14    return 0;
15
16}
```

datos.txt

x 1.5 1

# Escribir al final de un archivo existente

- Qué hace la operación `ofstream.open("archivo.txt")` si `archivo.txt` ya Existe?
  - Si existe, sobrescribe todo su contenido (borra lo que había antes)
  - Si no existe, crea el archivo
- ¿Cómo podemos hacer para que el contenido anterior sea respetado?
  - Para escribir al final del archivo hay que abrirlo en modo append
  - Para abrir un archivo en modo append, hay que usar `ofstream.open("archivo.txt", ios_base::app)`

# input file stream (ifstream)

mi\_archivo1.txt

```
1 #include <iostream>
2 #include <fstream>
3
4 int main() {
5     int a;
6     float b;
7
8
9
10
11
12
13
14
15
16 }
```

34 4.6

# input file stream (ifstream)

mi\_archivo1.txt

```
1 #include <iostream>
2 #include <fstream>
3
4 int main() {
5     int a;
6     float b;
7     std::ifstream fin;
8
9
10
11
12
13
14
15
16 }
```

34 4.6

# input file stream (ifstream)

mi\_archivo1.txt

```
1 #include <iostream>
2 #include <fstream>
3
4 int main() {
5     int a;
6     float b;
7     std::ifstream fin;
8     fin.open("mi_archivo1.txt");
9
10
11
12
13
14
15
16 }
```


34 4.6



# input file stream (ifstream)

mi\_archivo1.txt

```
1 #include <iostream>
2 #include <fstream>
3
4 int main() {
5     int a;
6     float b;
7     std::ifstream fin;
8     fin.open("mi_archivo1.txt");
10    fin >> a;
11
12
13
14
15
16}
```



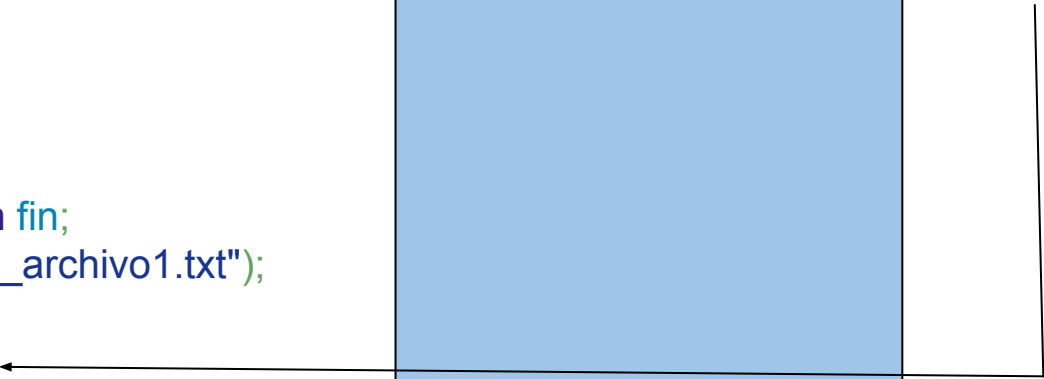
34 4.6

# input file stream (ifstream)

mi\_archivo1.txt

```
1 #include <iostream>
2 #include <fstream>
3
4 int main() {
5     int a;
6     float b;
7     std::ifstream fin;
8     fin.open("mi_archivo1.txt");
10    fin >> a;
11    fin >> b;
12
13
14
15
16}
```

34 4.6



# input file stream (ifstream)

```
1 #include <iostream>
2 #include <fstream>
3
4 int main() {
5     int a;
6     float b;
7     std::ifstream fin;
8     fin.open("mi_archivo1.txt");
9
10    fin >> a;
11    fin >> b;
12    fin.close();
13
14    return 0;
15 }
```

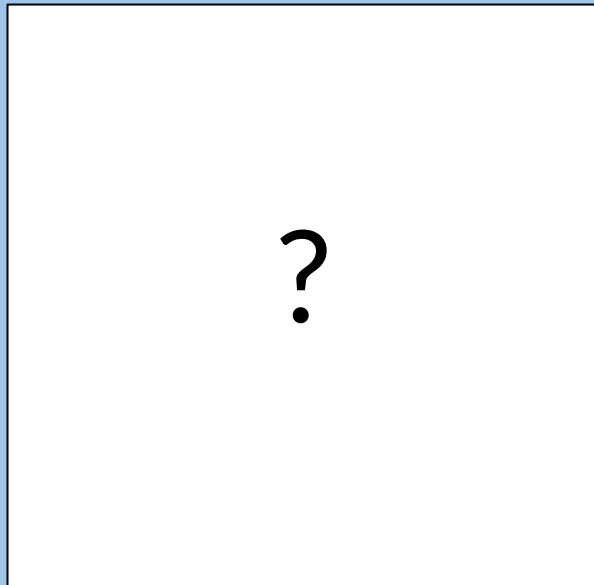
mi\_archivo1.txt

34 4.6

---

# ¿Qué pasa cuando no conocemos la cantidad de datos que hay en un archivo que tenemos que leer?

datos.txt



---

# Función end-of-file (eof)

- Además de open y close tenemos la función eof().
- La función eof() retorna true si ya no hay más contenido del archivo para leer.
- Usaremos eof() sólo cuando abrimos un archivo para lectura.
- Ejemplo:
  - Queremos leer de un archivo una lista de enteros y calcular la suma de sus elementos.

# End-of-file (eof)

```
1 include <iostream>
2 #include <fstream>
3
4 int main() {
5
6
7
8
9
10
11
12
13
14
15
16}
```

datos.txt

45 63 9

# End-of-file (eof)

```
1 include <iostream>
2 #include <fstream>
3
4 int main() {
5     std::ifstream fin;
6     fin.open("datos.txt");
7
8
9
10
11
12
13
14
15
16}
```

datos.txt

45 63 9

# End-of-file (eof)

```
1 include <iostream>
2 #include <fstream>
3
4 int main() {
5     std::ifstream fin;
6     fin.open("datos.txt");
7
8     while( !fin.eof() ){
9         int a;
10        fin >> a; ←
11    }
12
13
14
15
16}
```

datos.txt

45 63 9

A horizontal arrow points from the file stream object 'fin' in the C++ code to the file 'datos.txt'. A vertical line descends from the file name to the first line of the file's content, '45 63 9', indicating the current position of the file pointer.



# End-of-file (eof)

```
1 include <iostream>
2 #include <fstream>
3
4 int main() {
5     std::ifstream fin;
6     fin.open("datos.txt");
7
8     while( !fin.eof() ){
9         int a;
10        fin >> a; ←
11    }
12
13
14
15
16}
```

datos.txt

45 63 9



# End-of-file (eof)

```
1 include <iostream>
2 #include <fstream>
3
4 int main() {
5     std::ifstream fin;
6     fin.open("datos.txt");
7
8     while( !fin.eof() ){
9         int a;
10        fin >> a; ←
11    }
12
13
14
15
16}
```

datos.txt

45 63 9

A horizontal line with an arrowhead pointing left connects the file stream object 'fin' in the C++ code to the file 'datos.txt'. A vertical line descends from the file name to the file's content box, indicating the data source for the stream.

# End-of-file (eof)

```
1 include <iostream>
2 #include <fstream>
3
4 int main() {
5     std::ifstream fin;
6     fin.open("datos.txt");
7
8     while( !fin.eof() ){
9         int a;
10        fin >> a;
11    }
12
13    fin.close();
14
15    return 0;
16}
```

datos.txt

45 63 9

---

# Manejo de Errores

- Hasta ahora tenemos las funciones open, close y eof para operar con archivos.
  - ¿Qué pasa cuando queremos abrir un archivo para lectura que no existe?
  - ¿Qué pasa cuando no tenemos permisos para leer un archivo?
  - ¿Qué pasa cuando no tenemos permisos para sobrescribir un archivo?
  - Para todos esos casos, se puede consultar a la función fail()
  - La función **fail()** retorna true si hubo una falla al intentar ejecutar una operación (por ejemplo: open, close)
-

# Ejemplo de uso de fail()

```
1 include <iostream>
2 #include <fstream>
3
4 int main() {
5     std::ifstream fin;
6     fin.open("un_archivo.txt");
7
8     if (fin.fail()) {
9         std::cout << "Error" << std::endl;
10    }else{
11        std::cout << "Abierto" << std::endl;
12    }
13
14    fin.close();
15    return 0;
16}
```

# Resumen: E/S con archivos en C++

- **ifstream**: stream de lectura de archivos
- **ofstream**: stream para escritura de archivos
- **open()**: abre un archivo para escritura o lectura dependiendo del tipo de stream
- **close()**: cierra un archivo
- **Operador de inserción (<<)**: escribe un valor en el stream
- **Operador de extracción (>>)**: lee un valor del stream
- **eof()**: retorna true si la lectura del archivo llegó al final
- **fail()**: retorna true si la última operación falló