



FILE MANAGEMENT IN C++

Prepared By: Mayuri Popat,
Assistant Professor |CE Dept, CSPIT,
CHARUSAT.
Email id: mayuripopat.ce@charusat.ac.in

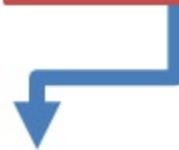
STREAM CLASSES FOR FILE OPERATION

STREAM CLASSES FOR FILE OPERATION

ios

STREAM CLASSES FOR FILE OPERATION

ios

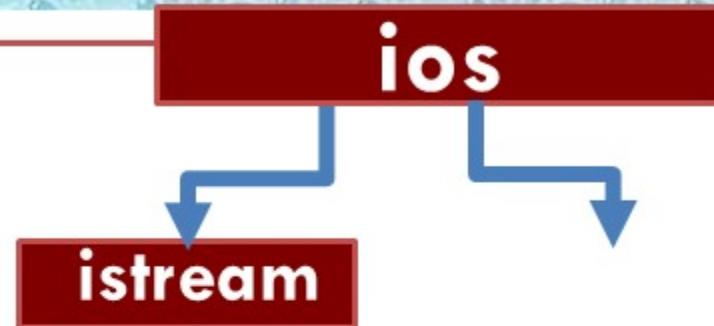


STREAM CLASSES FOR FILE OPERATION

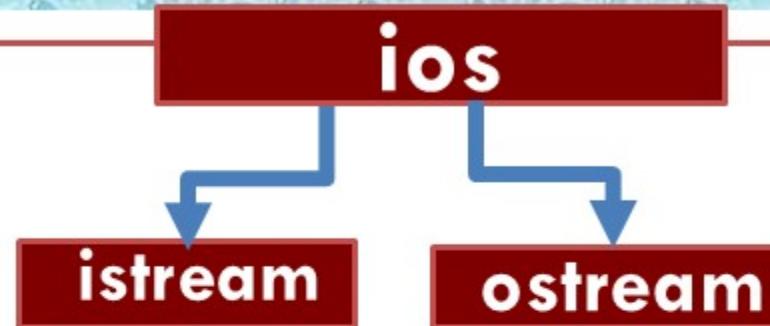
ios



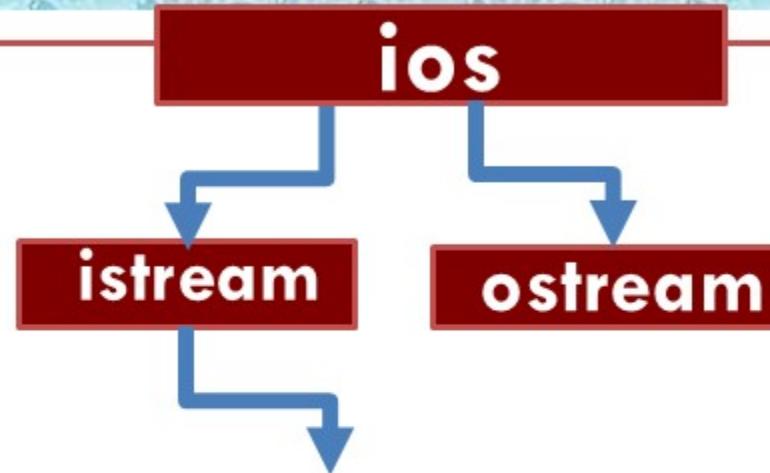
STREAM CLASSES FOR FILE OPERATION



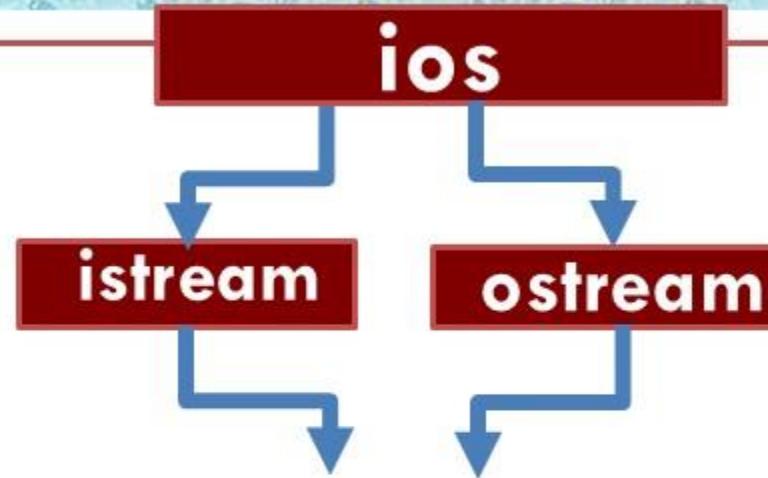
STREAM CLASSES FOR FILE OPERATION



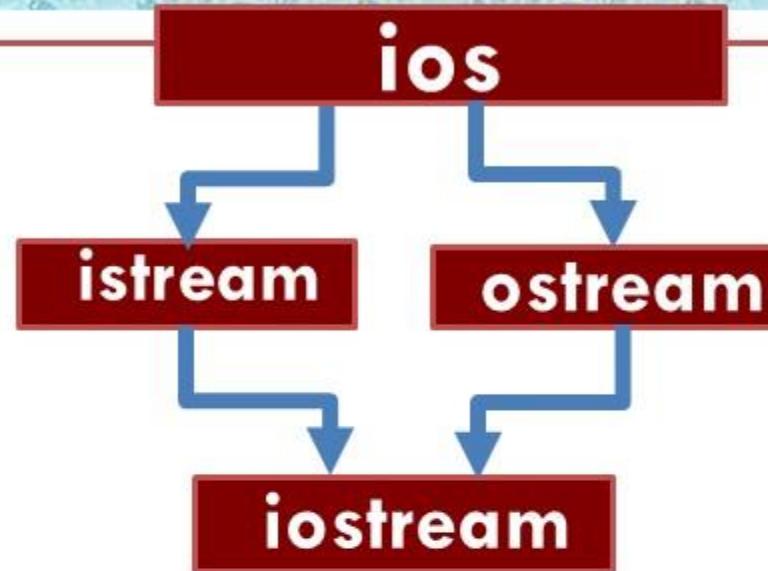
STREAM CLASSES FOR FILE OPERATION



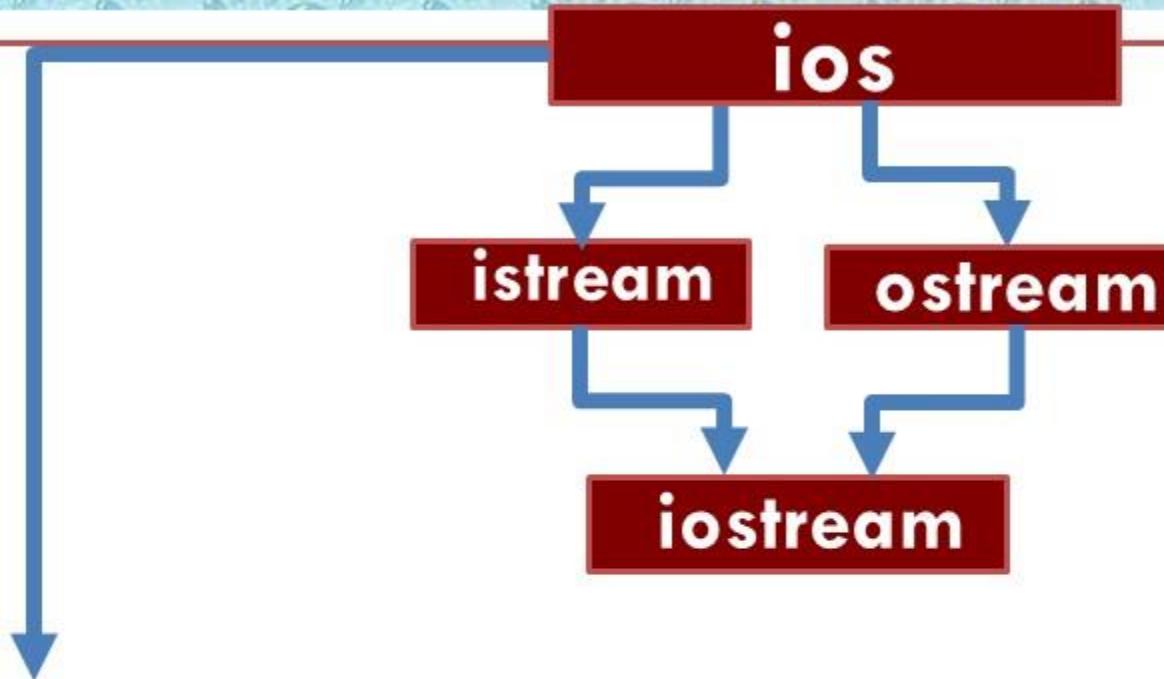
STREAM CLASSES FOR FILE OPERATION



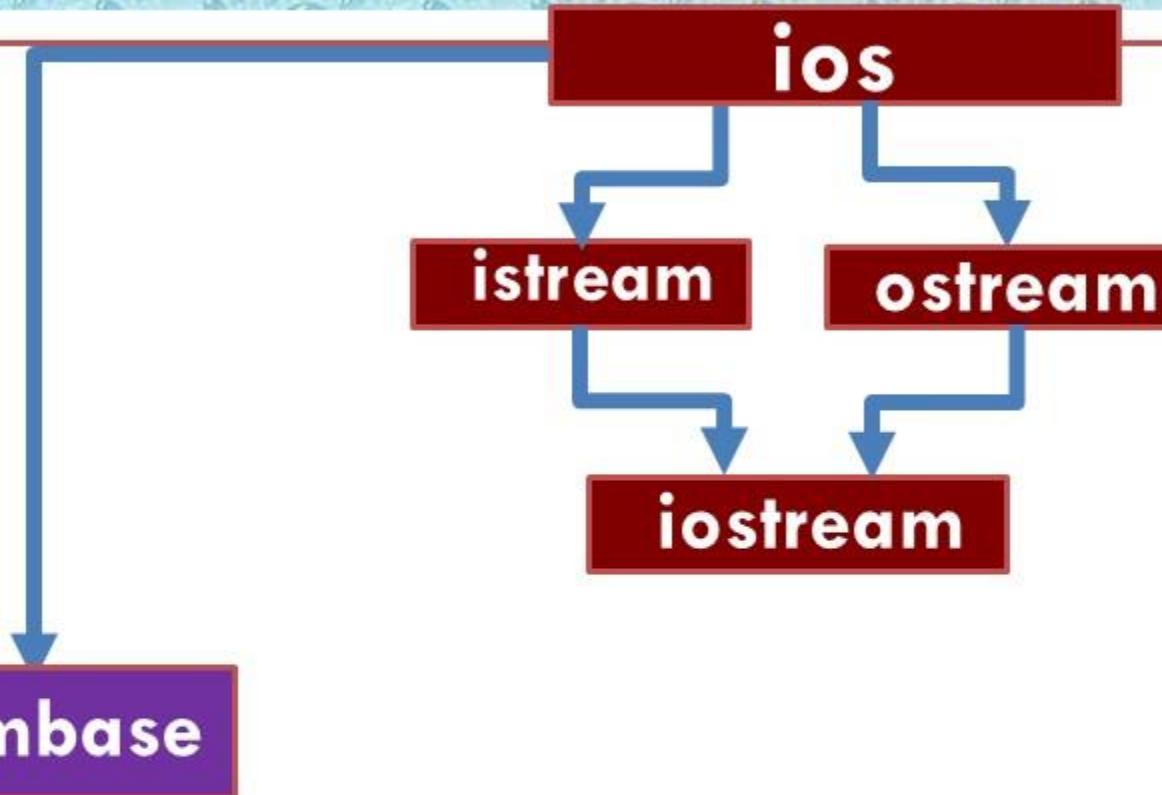
STREAM CLASSES FOR FILE OPERATION



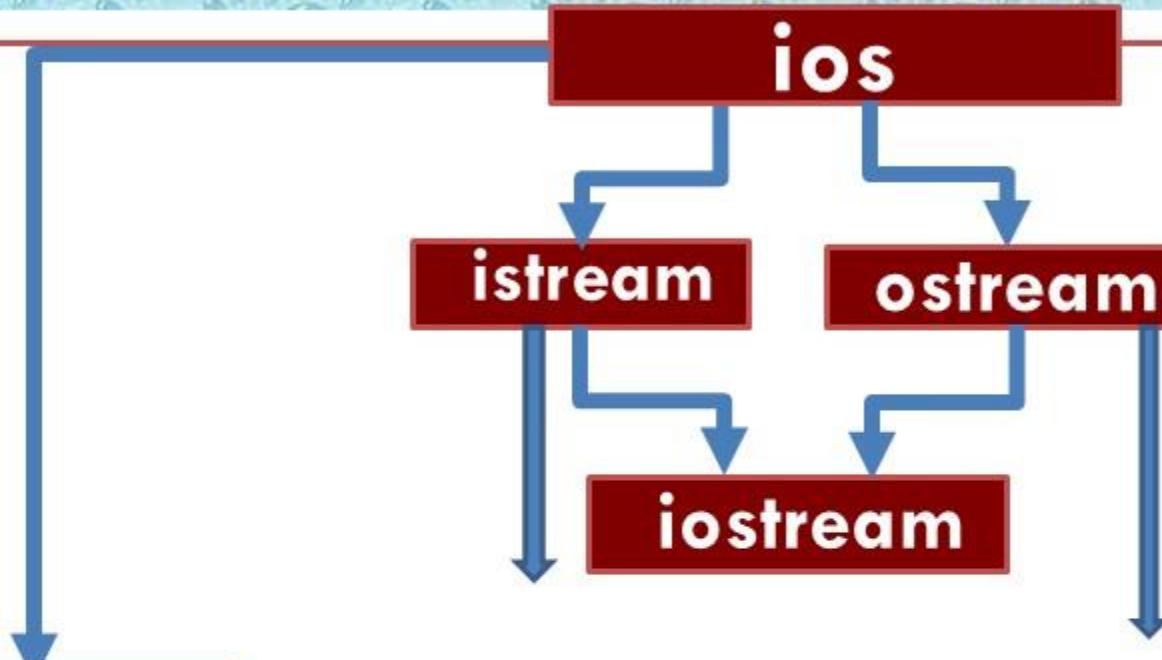
STREAM CLASSES FOR FILE OPERATION



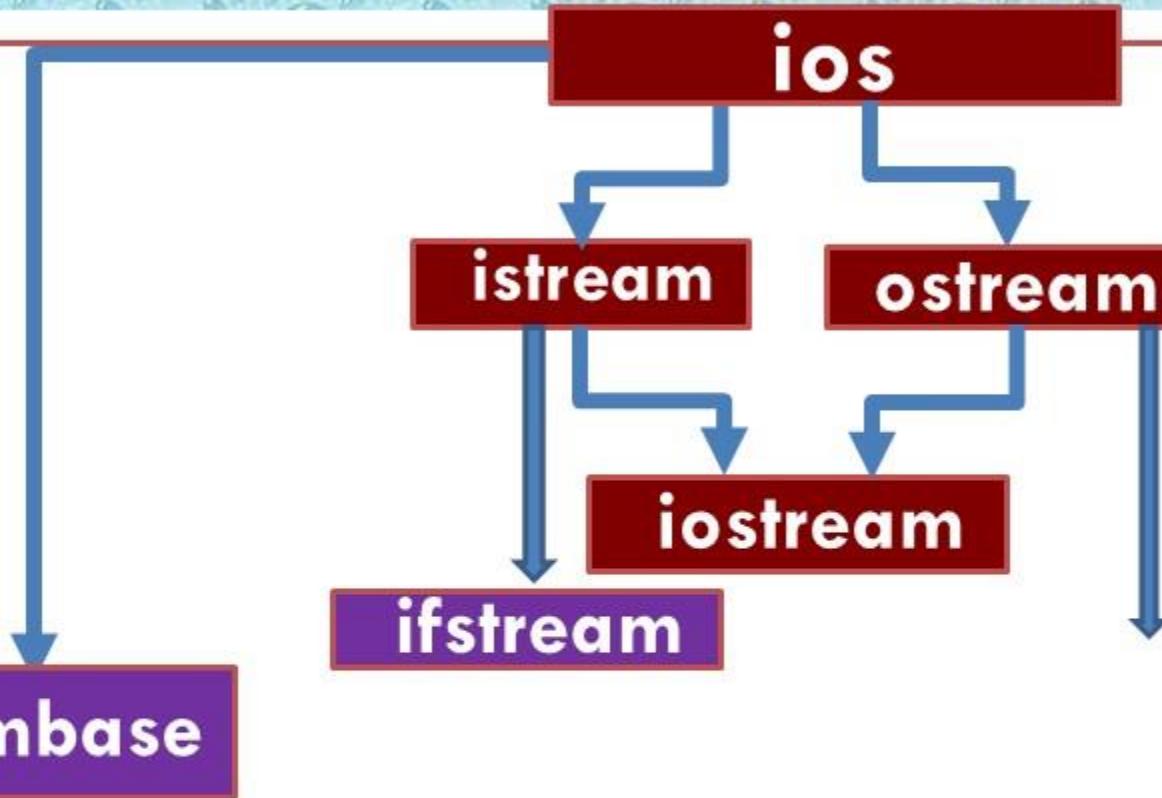
STREAM CLASSES FOR FILE OPERATION



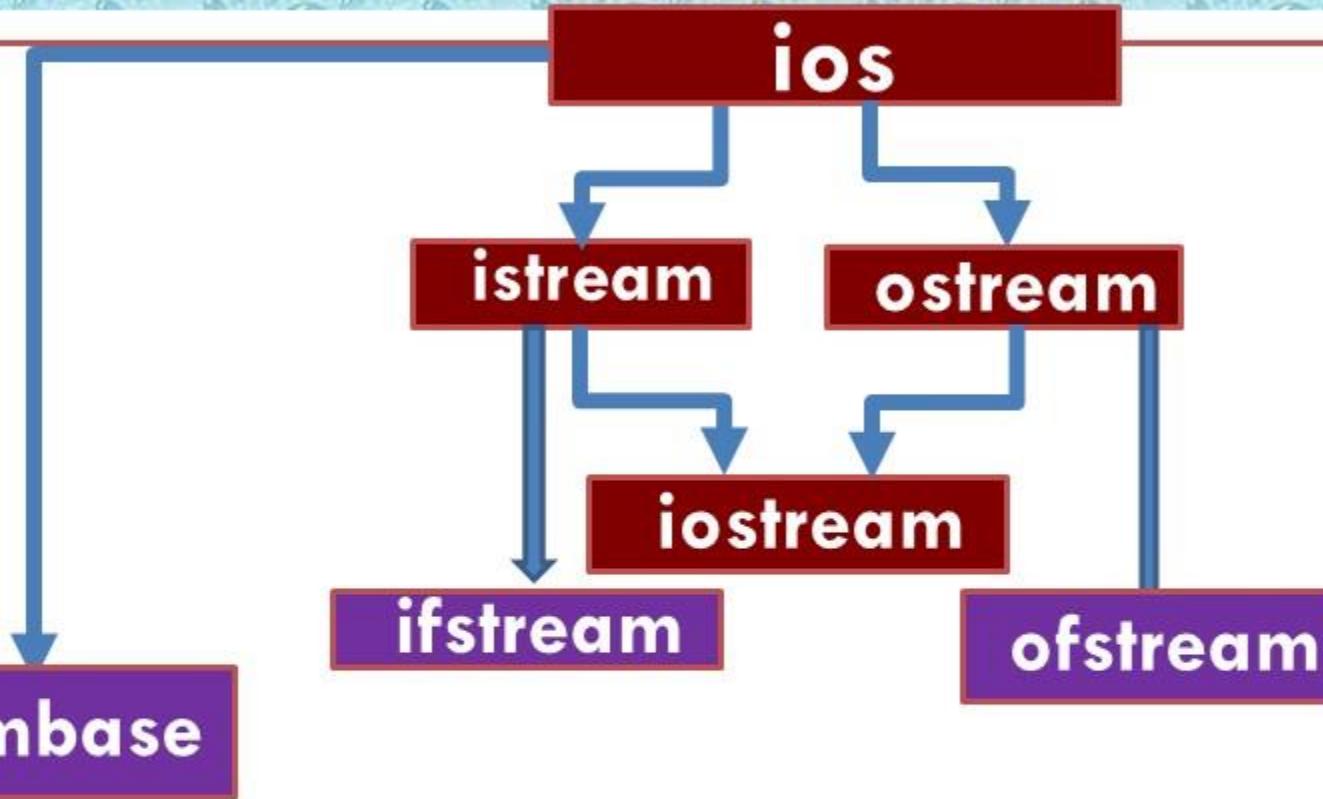
STREAM CLASSES FOR FILE OPERATION



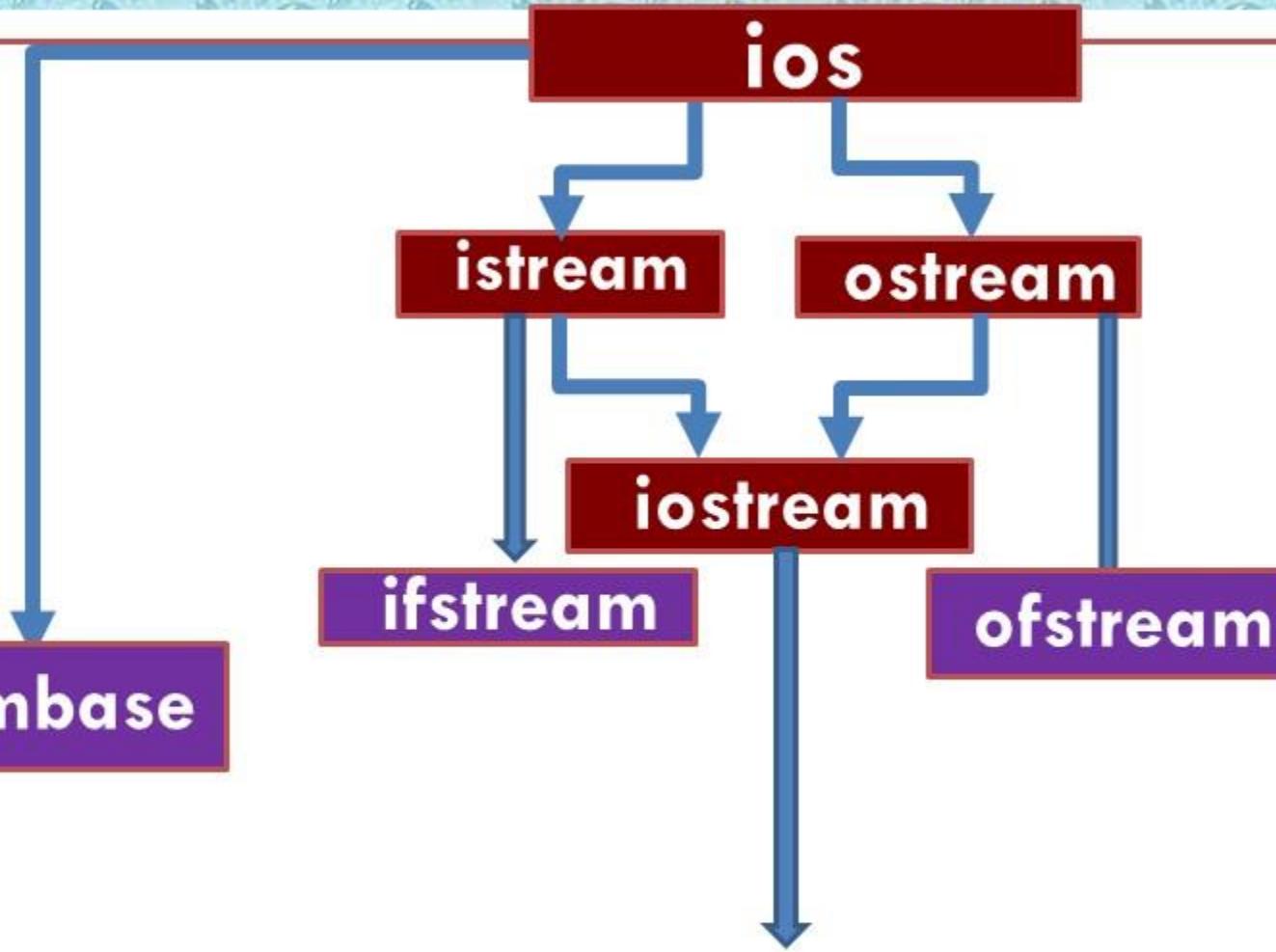
STREAM CLASSES FOR FILE OPERATION



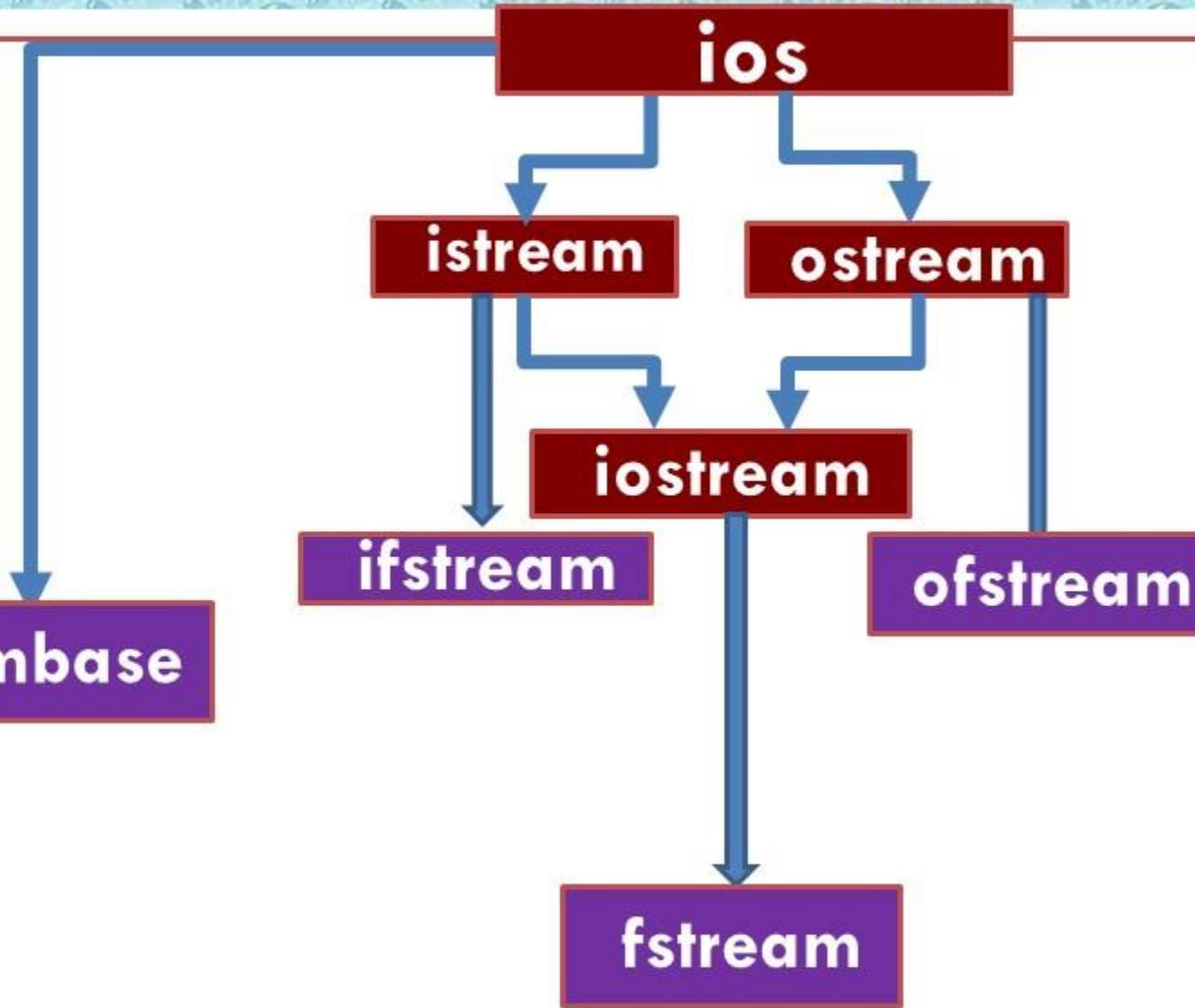
STREAM CLASSES FOR FILE OPERATION



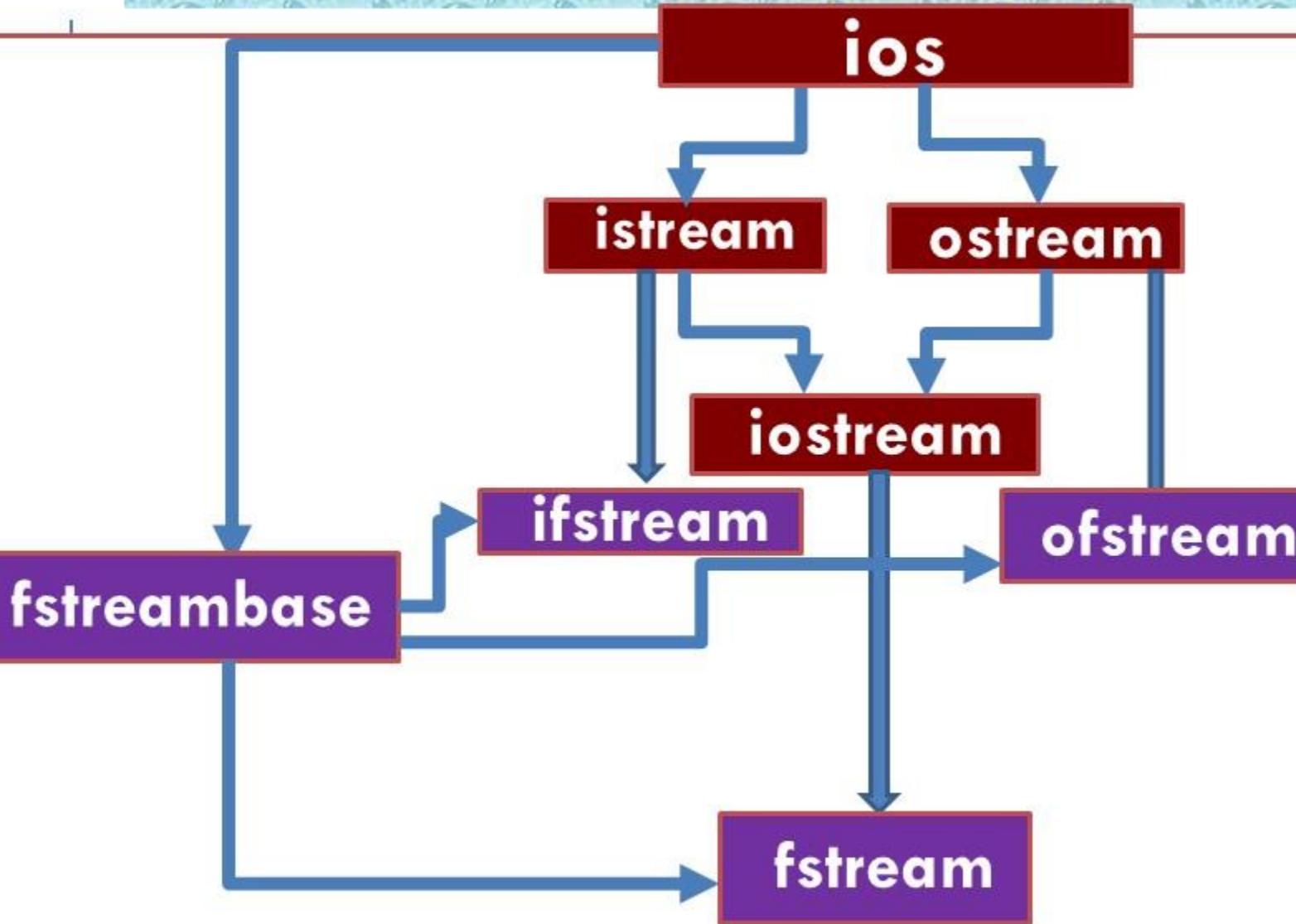
STREAM CLASSES FOR FILE OPERATION



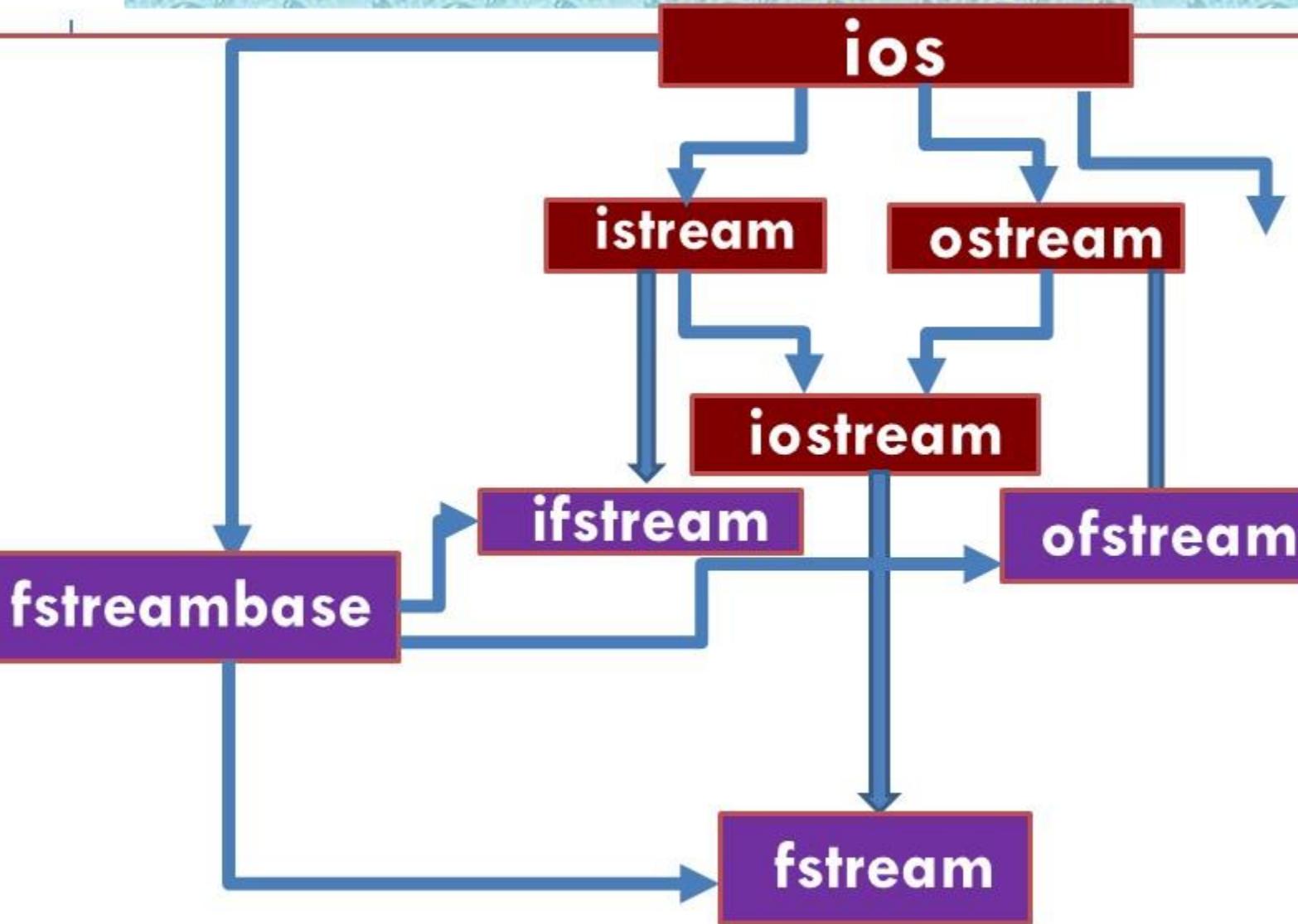
STREAM CLASSES FOR FILE OPERATION



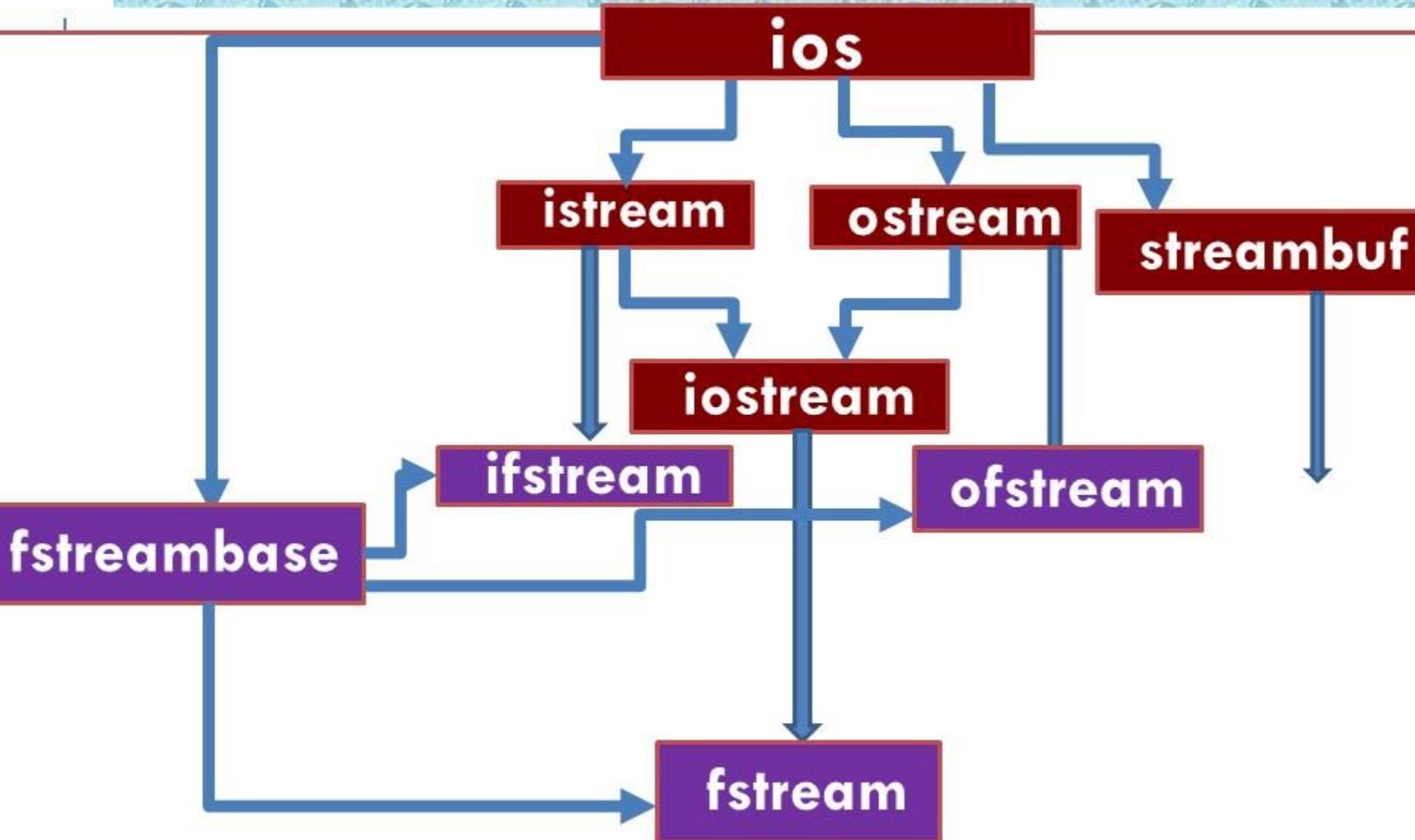
STREAM CLASSES FOR FILE OPERATION



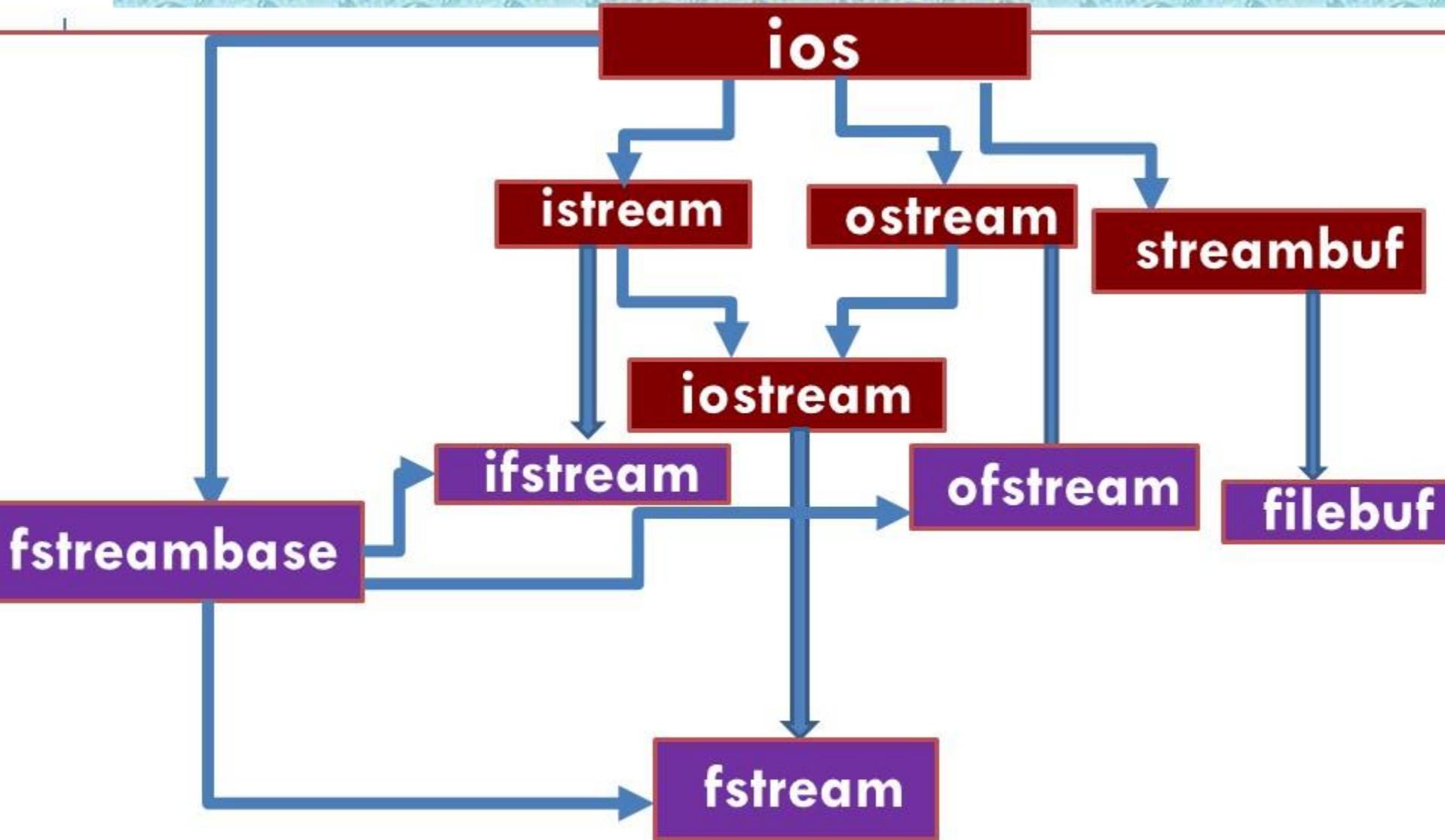
STREAM CLASSES FOR FILE OPERATION



STREAM CLASSES FOR FILE OPERATION

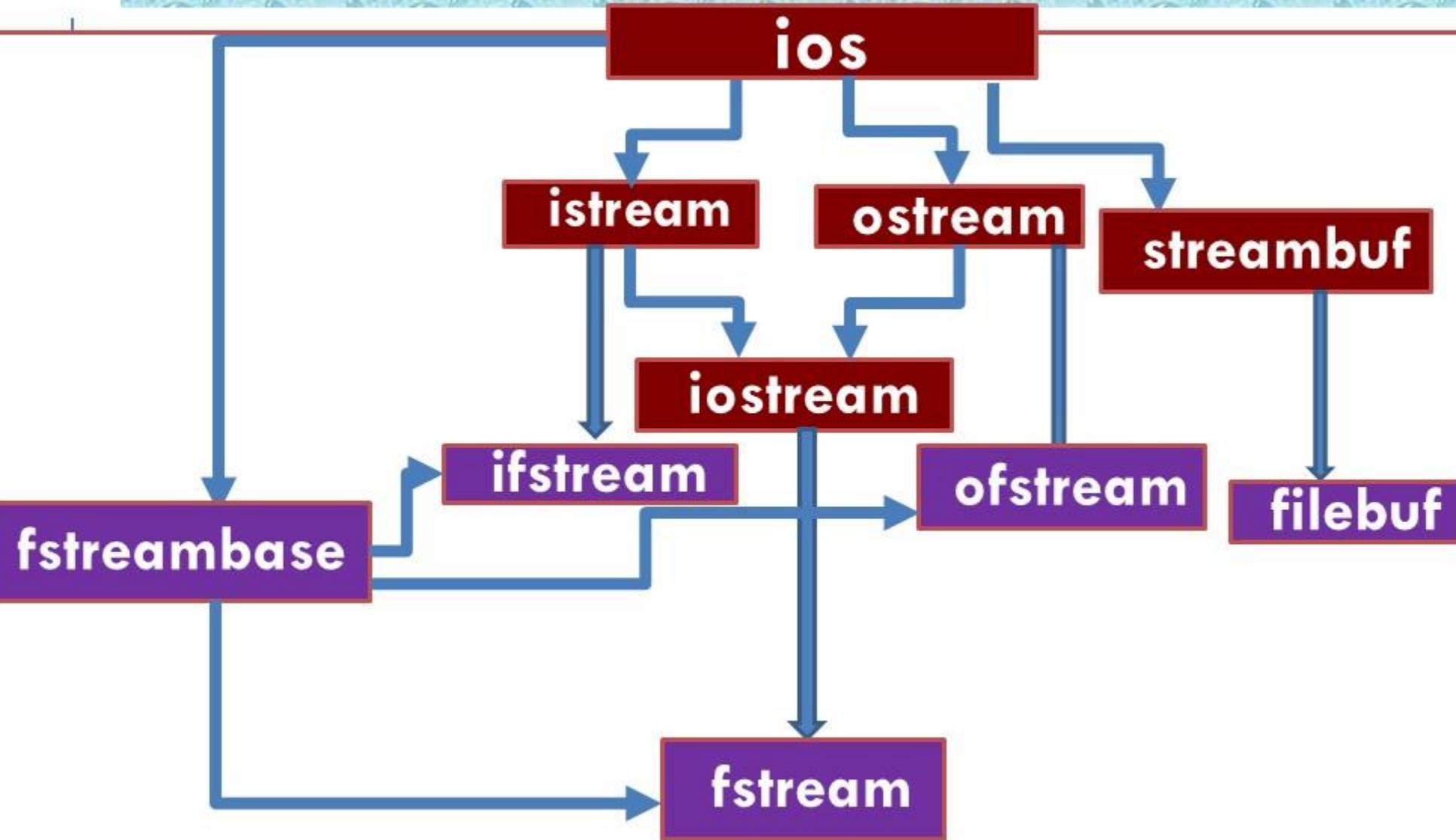


STREAM CLASSES FOR FILE OPERATION

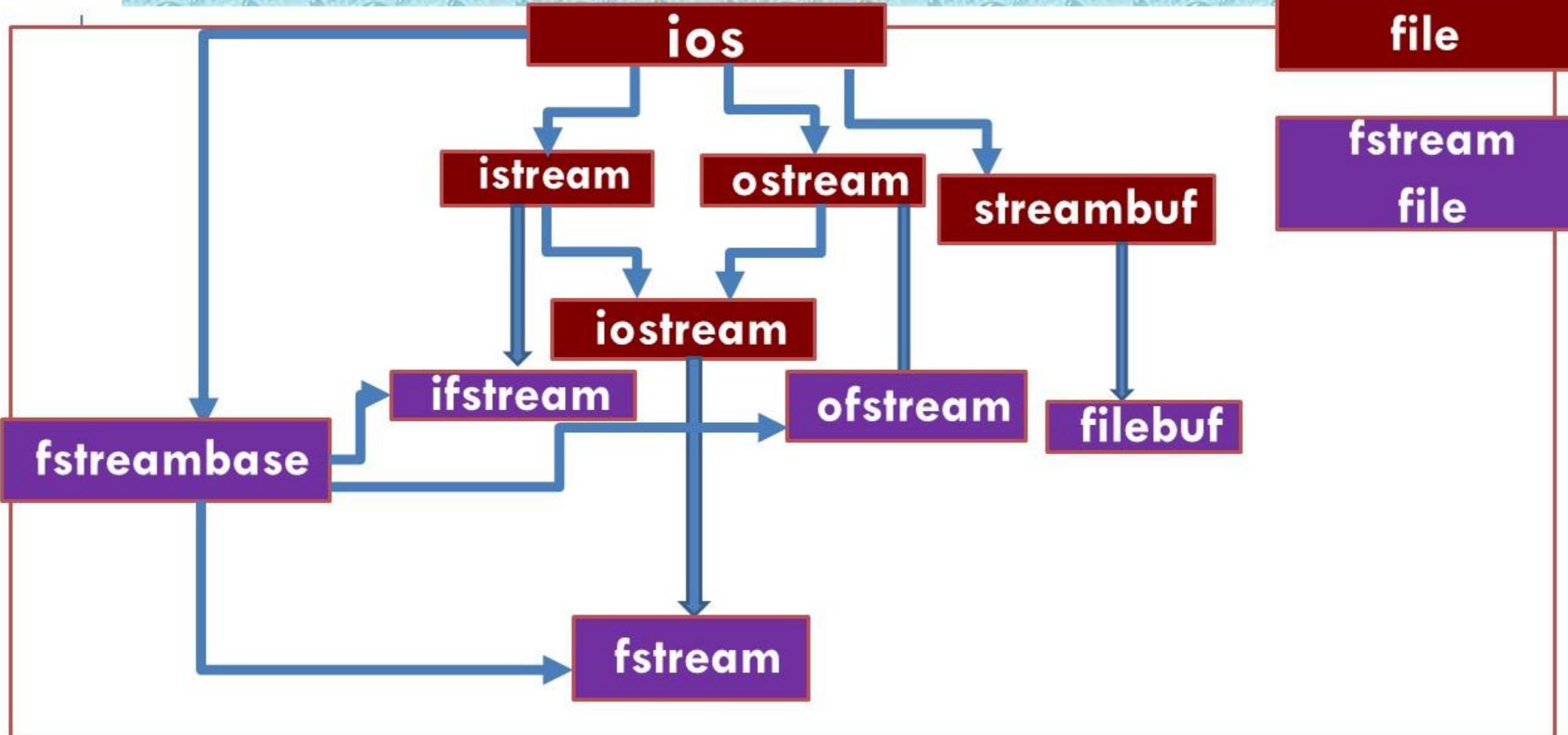


STREAM CLASSES FOR FILE OPERATION

iostream
file



STREAM CLASSES FOR FILE OPERATION



STREAM CLASSES FOR FILE OPERATION

filebuf

fstreambase

fstream

ifstream

ofstream

STREAM CLASSES FOR FILE OPERATION

STREAM CLASSES FOR FILE OPERATION

Class	

STREAM CLASSES FOR FILE OPERATION

Class	Content

STREAM CLASSES FOR FILE OPERATION

Class	Content
filebuf	

STREAM CLASSES FOR FILE OPERATION

Class	Content
filebuf	<ul style="list-style-type: none">• Its purpose is to set the file buffers to read and write.

STREAM CLASSES FOR FILE OPERATION

Class	Content
filebuf	<ul style="list-style-type: none">• Its purpose is to set the file buffers to read and write.• Contains openprot constant used in the open() of file stream classes.

STREAM CLASSES FOR FILE OPERATION

Class	Content
filebuf	<ul style="list-style-type: none">• Its purpose is to set the file buffers to read and write.• Contains openprot constant used in the open() of file stream classes.• Also contains open() and close() as members function.

STREAM CLASSES FOR FILE OPERATION

Class	Content
filebuf	<ul style="list-style-type: none">• Its purpose is to set the file buffers to read and write.• Contains openprot constant used in the open() of file stream classes.• Also contains open() and close() as members function.• What are buffers and openprot?

STREAM CLASSES FOR FILE OPERATION

Class	Content
filebuf	<ul style="list-style-type: none">• Its purpose is to set the file buffers to read and write.• Contains openprot constant used in the open() of file stream classes.• Also contains open() and close() as members function.• What are buffers and openprot? 

WHAT IS BUFFER?

WHAT IS BUFFER?

- Buffer is a region of a physical memory storage used to temporarily store data while it is being moved from one place to another.

WHAT IS BUFFER?

- Buffer is a region of a physical memory storage used to temporarily store data while it is being moved from one place to another.
- A buffer contains data that is stored for a short amount of time, typically in the computer's memory (RAM).

WHAT IS BUFFER?

- Buffer is a region of a physical memory storage used to temporarily store data while it is being moved from one place to another.
- A buffer contains data that is stored for a short amount of time, typically in the computer's memory (RAM).
- The purpose of a buffer is to hold data right before it is used.

WHAT IS BUFFER?

- Buffer is a region of a physical memory storage used to temporarily store data while it is being moved from one place to another.
- A buffer contains data that is stored for a short amount of time, typically in the computer's memory (RAM).
- The purpose of a buffer is to hold data right before it is used.
- The processes of reading and writing data to a disk are relatively slow.

WHAT IS BUFFER?

- Buffer is a region of a physical memory storage used to temporarily store data while it is being moved from one place to another.
- A buffer contains data that is stored for a short amount of time, typically in the computer's memory (RAM).
- The purpose of a buffer is to hold data right before it is used.
- The processes of reading and writing data to a disk are relatively slow.
- Therefore many programs keep track of data changes in a buffer and then copy the buffer to a disk.

BUFFER EXAMPLE

BUFFER EXAMPLE

- For example, when you download an audio or video file from the Internet, it may load the first 20% of it into a buffer and then begin to play.

BUFFER EXAMPLE

- For example, when you download an audio or video file from the Internet, it may load the first 20% of it into a buffer and then begin to play.
- While the clip plays back, the computer continually downloads the rest of the clip and stores it in the buffer.

BUFFER EXAMPLE

- For example, when you download an audio or video file from the Internet, it may load the first 20% of it into a buffer and then begin to play.
- While the clip plays back, the computer continually downloads the rest of the clip and stores it in the buffer.
- Because the clip is being played from the buffer, not directly from the Internet, there is less of a chance that the audio or video will stall or skip when there is network congestion.

BUFFER EXAMPLE

BUFFER EXAMPLE

- For example, word processors employ a buffer to keep track of changes to files.

BUFFER EXAMPLE

- For example, word processors employ a buffer to keep track of changes to files.
- Then when you save the file, the word processor updates the disk file with the contents of the buffer.

BUFFER EXAMPLE

- For example, word processors employ a buffer to keep track of changes to files.
- Then when you save the file, the word processor updates the disk file with the contents of the buffer.
- This is much more efficient than accessing the file on the disk each time you make a change to the file.

BUFFER EXAMPLE

- For example, word processors employ a buffer to keep track of changes to files.
- Then when you save the file, the word processor updates the disk file with the contents of the buffer.
- This is much more efficient than accessing the file on the disk each time you make a change to the file.
- Note that because your changes are initially stored in a buffer, not on the disk, all of them will be lost if the computer fails during an editing session.

BUFFER EXAMPLE

- For example, word processors employ a buffer to keep track of changes to files.
- Then when you save the file, the word processor updates the disk file with the contents of the buffer.
- This is much more efficient than accessing the file on the disk each time you make a change to the file.
- Note that because your changes are initially stored in a buffer, not on the disk, all of them will be lost if the computer fails during an editing session.
- For this reason, it is a good idea to save your file periodically.



WHAT IS OPENPROT CONSTANT?

WHAT IS OPENPROT CONSTANT?

- Openprot means opening a file in protection mode.

WHAT IS OPENPROT CONSTANT?

- Openprot means opening a file in protection mode.
- Openprot has a constant value 0644.

WHAT IS OPENPROT CONSTANT?

- Openprot means opening a file in protection mode.
- Openprot has a constant value 0644.
- 6 is read/write and 4 is read

WHAT IS OPENPROT CONSTANT?

- Openprot means opening a file in protection mode.
- Openprot has a constant value 0644.
- 6 is read/write and 4 is read
- The first number is Owner, the second is Group, and the third is Other.

WHAT IS OPENPROT CONSTANT?

- Openprot means opening a file in protection mode.
- Openprot has a constant value 0644.
- 6 is read/write and 4 is read
- The first number is Owner, the second is Group, and the third is Other.
- 644 means that files are readable and writeable by the owner of the file and readable by users in the group of that file and readable by everyone else.

STREAM CLASSES FOR FILE OPERATION

STREAM CLASSES FOR FILE OPERATION

Class

STREAM CLASSES FOR FILE OPERATION

Class	Content

STREAM CLASSES FOR FILE OPERATION

Class

fstreambase

Content

STREAM CLASSES FOR FILE OPERATION

Class	Content
fstreambase	<ul style="list-style-type: none">•Provides operations common to the file streams.

STREAM CLASSES FOR FILE OPERATION

Class

fstreambase

Content

- Provides operations common to the file streams.
- Serves as a base class for **fstream**, **ifstream** and **ofstream** class.

STREAM CLASSES FOR FILE OPERATION

Class	Content
fstreambase	<ul style="list-style-type: none">• Provides operations common to the file streams.• Serves as a base class for fstream, ifstream and ofstream class.• Contains open() and close() as members function.

STREAM CLASSES FOR FILE OPERATION

STREAM CLASSES FOR FILE OPERATION

Class

STREAM CLASSES FOR FILE OPERATION

Class	Content

STREAM CLASSES FOR FILE OPERATION

Class	Content
fstream	

STREAM CLASSES FOR FILE OPERATION

Class

fstream

Content

- Provides support for simultaneous input and output operations.

STREAM CLASSES FOR FILE OPERATION

Class

fstream

Content

- Provides support for simultaneous input and output operations.
- Inherits all the functions from istream and ostream classes through iostream.

STREAM CLASSES FOR FILE OPERATION

STREAM CLASSES FOR FILE OPERATION

Class	

STREAM CLASSES FOR FILE OPERATION

Class

Content

STREAM CLASSES FOR FILE OPERATION

Class

Content

ifstream

STREAM CLASSES FOR FILE OPERATION

Class	Content
ifstream	<ul style="list-style-type: none">•Provides input operations.

STREAM CLASSES FOR FILE OPERATION

Class	Content
ifstream	<ul style="list-style-type: none">• Provides input operations.• Contains open() with default input mode (ios::in).

STREAM CLASSES FOR FILE OPERATION

Class	Content
ifstream	<ul style="list-style-type: none">• Provides input operations.• Contains open() with default input mode (<code>ios::in</code>).• Inherits the function get(), getline() , read() seekg() and tellg() functions from istream.

STREAM CLASSES FOR FILE OPERATION

STREAM CLASSES FOR FILE OPERATION

Class

STREAM CLASSES FOR FILE OPERATION

Class

Content

STREAM CLASSES FOR FILE OPERATION

Class

Content

ofstream

STREAM CLASSES FOR FILE OPERATION

Class

ofstream

Content

- Provides output operations.

STREAM CLASSES FOR FILE OPERATION

Class

ofstream

Content

- Provides output operations.
- Contains **open()** with default output mode (ios::out).

STREAM CLASSES FOR FILE OPERATION

Class

ofstream

Content

- Provides output operations.
- Contains **open()** with default output mode (ios::out).
- Inherits the function **put()**, **write()**, **seekp()** and **tellp()** functions from **ostream**.

C++ FILES

C++ FILES

- C++ views each files as a sequence of bytes.

C++ FILES

- C++ views each files as a sequence of bytes.
- Each file ends with an **end-of-file** marker.

C++ FILES

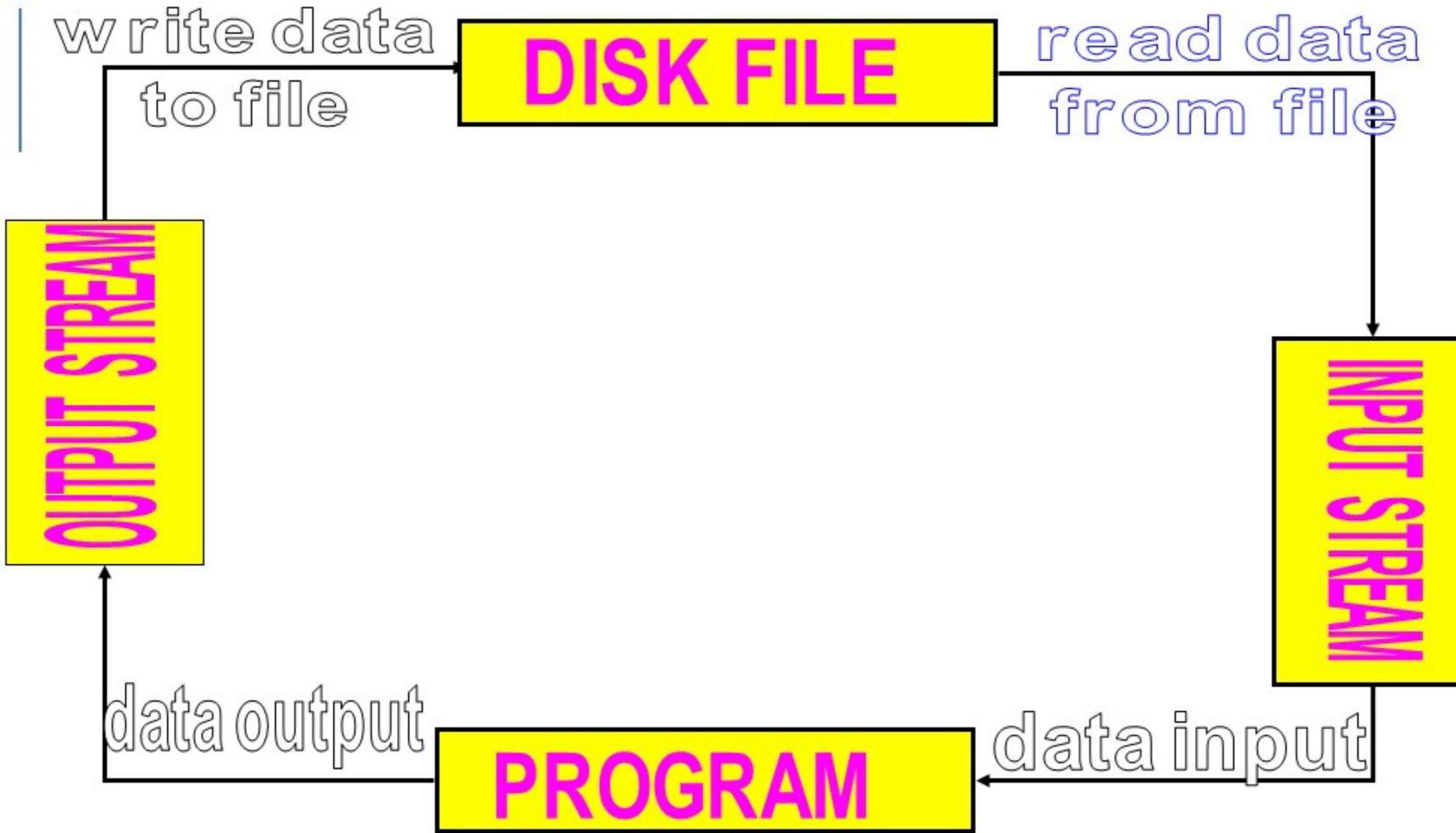
- C++ views each files as a sequence of bytes.
- Each file ends with an **end-of-file** marker.
- When a file is **opened**, an object is created and a stream is associated with the object.

C++ FILES

- C++ views each files as a sequence of bytes.
- Each file ends with an **end-of-file** marker.
- When a file is **opened**, an object is created and a stream is associated with the object.
- To perform file processing in C++, the header files **<iostream>** and **<fstream>** must be included.

STREAMS

- A Stream is sequence of bytes and deals with the flow of data.
 - Streams act as an interface between files and programs.
 - Every stream is associated with a class having member functions and operations for a particular kind of data flow.
-
- ✓ File → Program (Input stream) - reads
 - ✓ Program → File (Output stream) – write



OPENING AND CLOSING A FILE IN C++

OPENING AND CLOSING A FILE IN C++

If programmers want to use a disk file for storing data, they need to decide about the following things about the file and its intended use.

OPENING AND CLOSING A FILE IN C++

If programmers want to use a disk file for storing data, they need to decide about the following things about the file and its intended use.

These points that are to be noted are:

OPENING AND CLOSING A FILE IN C++

If programmers want to use a disk file for storing data, they need to decide about the following things about the file and its intended use.

These points that are to be noted are:

- A name for the file

OPENING AND CLOSING A FILE IN C++

If programmers want to use a disk file for storing data, they need to decide about the following things about the file and its intended use.

These points that are to be noted are:

- A name for the file
- Data type and structure of the file

OPENING AND CLOSING A FILE IN C++

If programmers want to use a disk file for storing data, they need to decide about the following things about the file and its intended use.

These points that are to be noted are:

- A name for the file
- Data type and structure of the file
- Purpose (reading, writing data)

OPENING AND CLOSING A FILE IN C++

If programmers want to use a disk file for storing data, they need to decide about the following things about the file and its intended use.

These points that are to be noted are:

- A name for the file
- Data type and structure of the file
- Purpose (reading, writing data)
- Opening method

OPENING AND CLOSING A FILE IN C++

If programmers want to use a disk file for storing data, they need to decide about the following things about the file and its intended use.

These points that are to be noted are:

- A name for the file
- Data type and structure of the file
- Purpose (reading, writing data)
- Opening method
- Closing the file (after use)

OPENING A FILE

- For opening a file , we must create a file stream and then link it to the filename.
- A file stream can be defined using the classes **ifstream**, **ofstream** and **fstream** that are contained in the header file **fstream**.
- The classes to be used depends upon the purpose,that is , whether we want to read data from the file or write data to it.

OPENING A FILE

OPENING A FILE

A file can be opened in two ways:

OPENING A FILE

A file can be opened in two ways:

1. Using the constructor function of the class.

OPENING A FILE

A file can be opened in two ways:

1. Using the constructor function of the class.

Useful when we use only one file in the stream.

OPENING A FILE

A file can be opened in two ways:

1. Using the constructor function of the class.

Useful when we use only one file in the stream.

2. Using the member function open() of the class.

OPENING A FILE

A file can be opened in two ways:

1. Using the constructor function of the class.

Useful when we use only one file in the stream.

2. Using the member function open() of the class.

Useful when we want to manage multiple files using one stream.

HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN WRITE MODE ?

HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN WRITE MODE ?

1. opening a file using constructor in write mode

HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN WRITE MODE ?

1. opening a file using constructor in write mode

```
fstream out ("abc.txt",ios::out);
```

HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN WRITE MODE ?

1. opening a file using constructor in write mode

```
fstream out ("abc.txt",ios::out);
```

OR

HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN WRITE MODE ?

1. opening a file using constructor in write mode

```
fstream out ("abc.txt",ios::out);
```

OR

2. opening a file using open() member function in write mode

HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN WRITE MODE ?

1. opening a file using constructor in write mode

```
fstream out ("abc.txt",ios::out);
```

OR

2. opening a file using open() member function in write mode

```
fstream out;
```

HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN WRITE MODE ?

1. opening a file using constructor in write mode

```
fstream out ("abc.txt",ios::out);
```

OR

2. opening a file using open() member function in write mode

```
fstream out;
```

```
out.open("abc.txt", ios::out);
```

HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN WRITE MODE ?

1. opening a file using constructor in write mode

```
fstream out (“abc.txt”,ios::out);
```

OR

2. opening a file using open() member function in write mode

```
fstream out;
```

```
out.open(“abc.txt”, ios::out);
```

prog.cpp



HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN WRITE MODE ?

1. opening a file using constructor in write mode

```
fstream out (“abc.txt”,ios::out);
```

OR

2. opening a file using open() member function in write mode

```
fstream out;
```

```
out.open(“abc.txt”, ios::out);
```

prog.cpp



HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN WRITE MODE ?

1. opening a file using constructor in write mode

```
fstream out ("abc.txt",ios::out);
```

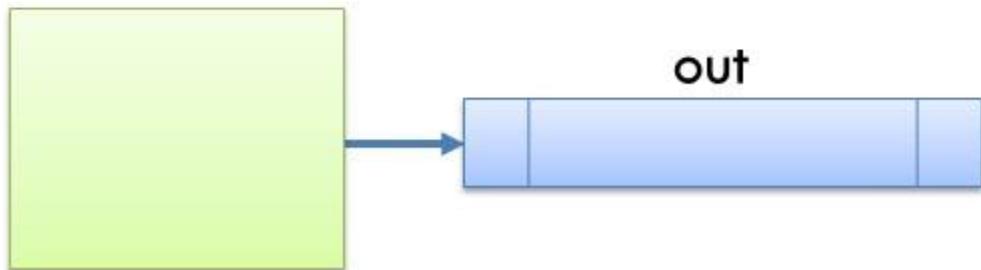
OR

2. opening a file using open() member function in write mode

```
fstream out;
```

```
out.open("abc.txt", ios::out);
```

prog.cpp



HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN WRITE MODE ?

1. opening a file using constructor in write mode

```
fstream out ("abc.txt",ios::out);
```

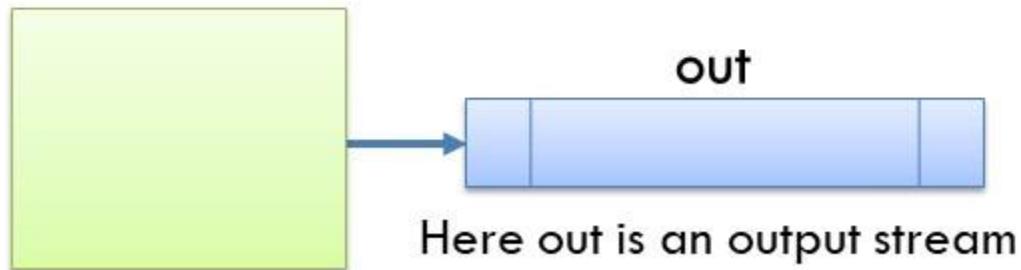
OR

2. opening a file using open() member function in write mode

```
fstream out;
```

```
out.open("abc.txt", ios::out);
```

prog.cpp



HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN WRITE MODE ?

1. opening a file using constructor in write mode

```
fstream out ("abc.txt",ios::out);
```

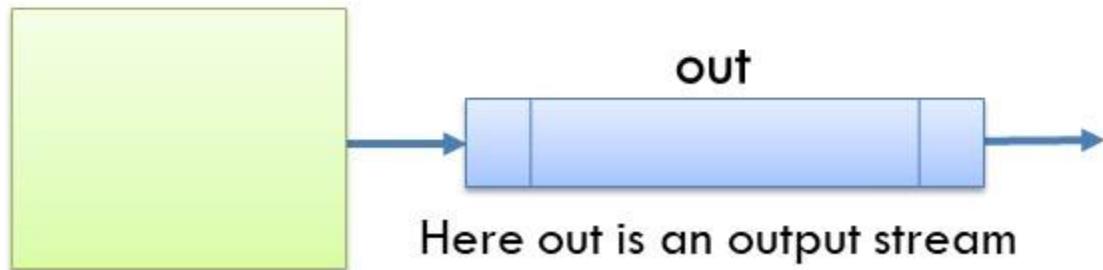
OR

2. opening a file using open() member function in write mode

```
fstream out;
```

```
out.open("abc.txt", ios::out);
```

prog.cpp



HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN WRITE MODE ?

1. opening a file using constructor in write mode

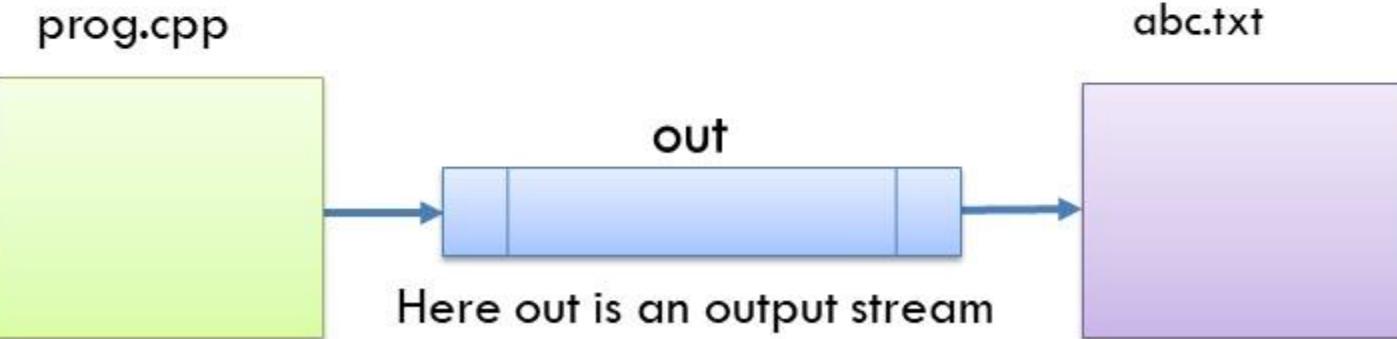
```
fstream out ("abc.txt",ios::out);
```

OR

2. opening a file using open() member function in write mode

```
fstream out;
```

```
out.open("abc.txt", ios::out);
```



HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN READ MODE ?

HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN READ MODE ?

1. opening a file using constructor in read mode

HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN READ MODE ?

1. opening a file using constructor in read mode

```
fstream fin ("abc.txt",ios::in);
```

HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN READ MODE ?

1. opening a file using constructor in read mode

```
fstream fin ("abc.txt",ios::in);
```

OR

HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN READ MODE ?

1. opening a file using constructor in read mode

```
fstream fin ("abc.txt",ios::in);
```

OR

2. opening a file using open() member function in read mode

HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN READ MODE ?

1. opening a file using constructor in read mode

```
fstream fin ("abc.txt",ios::in);
```

OR

2. opening a file using open() member function in read mode

```
fstream fin;
```

HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN READ MODE ?

1. opening a file using constructor in read mode

```
fstream fin ("abc.txt",ios::in);
```

OR

2. opening a file using open() member function in read mode

```
fstream fin;
```

```
fin.open("abc.txt", ios::in);
```

HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN READ MODE ?

1. opening a file using constructor in read mode

```
fstream fin ("abc.txt",ios::in);
```

OR

2. opening a file using open() member function in read mode

```
fstream fin;
```

```
fin.open("abc.txt", ios::in);
```

abc.txt



HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN READ MODE ?

1. opening a file using constructor in read mode

```
fstream fin ("abc.txt",ios::in);
```

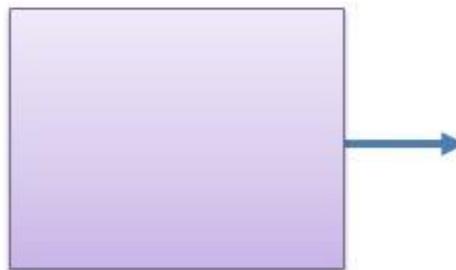
OR

2. opening a file using open() member function in read mode

```
fstream fin;
```

```
fin.open("abc.txt", ios::in);
```

abc.txt



HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN READ MODE ?

1. opening a file using constructor in read mode

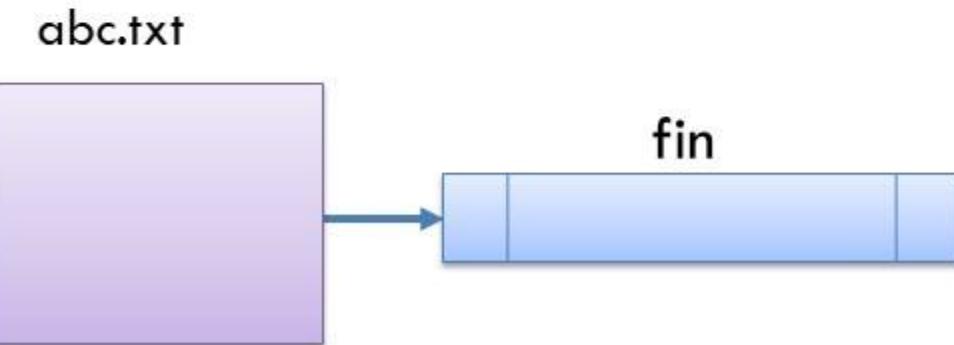
```
fstream fin ("abc.txt",ios::in);
```

OR

2. opening a file using open() member function in read mode

```
fstream fin;
```

```
fin.open("abc.txt", ios::in);
```



HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN READ MODE ?

1. opening a file using constructor in read mode

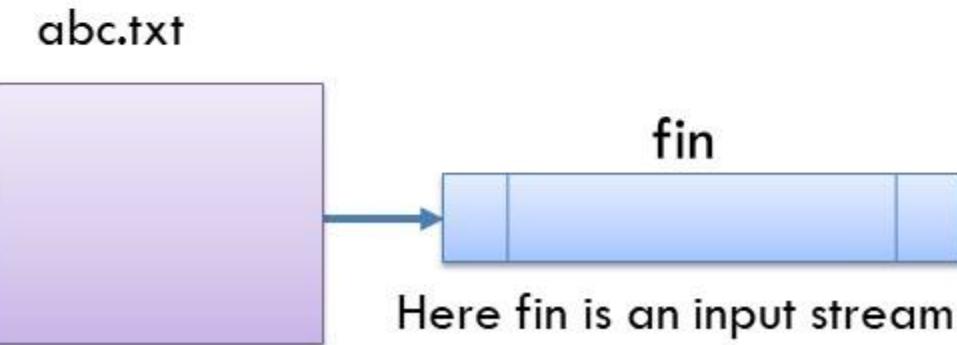
```
fstream fin ("abc.txt",ios::in);
```

OR

2. opening a file using open() member function in read mode

```
fstream fin;
```

```
fin.open("abc.txt", ios::in);
```



HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN READ MODE ?

1. opening a file using constructor in read mode

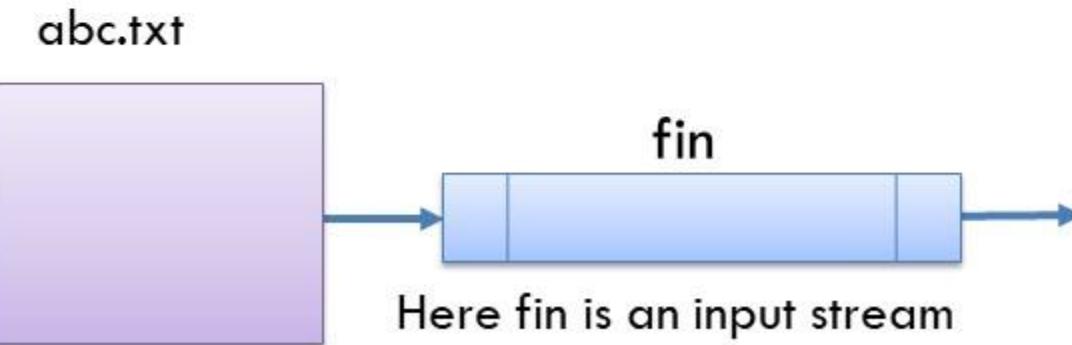
```
fstream fin ("abc.txt",ios::in);
```

OR

2. opening a file using open() member function in read mode

```
fstream fin;
```

```
fin.open("abc.txt", ios::in);
```



HOW TO OPEN A FILE IN C++ USING FSTREAM CLASS IN READ MODE ?

1. opening a file using constructor in read mode

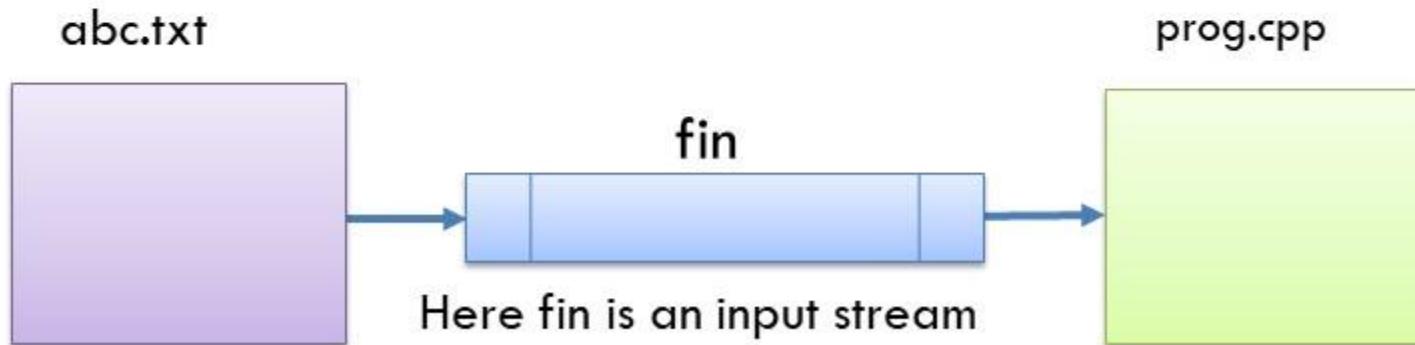
```
fstream fin ("abc.txt",ios::in);
```

OR

2. opening a file using open() member function in read mode

```
fstream fin;
```

```
fin.open("abc.txt", ios::in);
```



HOW TO OPEN A FILE IN C++ USING OFSTREAM CLASS ?

HOW TO OPEN A FILE IN C++ USING OFSTREAM CLASS ?

1. opening a file using constructor

HOW TO OPEN A FILE IN C++ USING OFSTREAM CLASS ?

1. opening a file using constructor

```
ofstream out ("abc.txt");
```

HOW TO OPEN A FILE IN C++ USING OFSTREAM CLASS ?

1. opening a file using constructor

```
ofstream out ("abc.txt");
```

OR

HOW TO OPEN A FILE IN C++ USING OFSTREAM CLASS ?

1. opening a file using constructor

```
ofstream out ("abc.txt");
```

OR

2. opening a file using open() member function

HOW TO OPEN A FILE IN C++ USING OFSTREAM CLASS ?

1. opening a file using constructor

```
ofstream out ("abc.txt");
```

OR

2. opening a file using open() member function

```
ofstream out;
```

HOW TO OPEN A FILE IN C++ USING OFSTREAM CLASS ?

1. opening a file using constructor

```
ofstream out ("abc.txt");
```

OR

2. opening a file using open() member function

```
ofstream out;
```

```
out.open("abc.txt");
```

HOW TO OPEN A FILE IN C++ USING OFSTREAM CLASS ?

1. opening a file using constructor

```
ofstream out ("abc.txt");
```

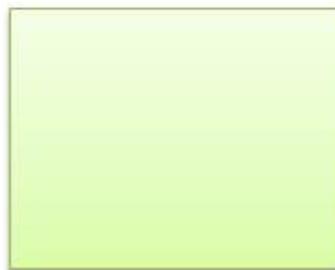
OR

2. opening a file using open() member function

```
ofstream out;
```

```
out.open("abc.txt");
```

prog.cpp



HOW TO OPEN A FILE IN C++ USING OFSTREAM CLASS ?

1. opening a file using constructor

```
ofstream out ("abc.txt");
```

OR

2. opening a file using open() member function

```
ofstream out;
```

```
out.open("abc.txt");
```

prog.cpp



HOW TO OPEN A FILE IN C++ USING OFSTREAM CLASS ?

1. opening a file using constructor

```
ofstream out ("abc.txt");
```

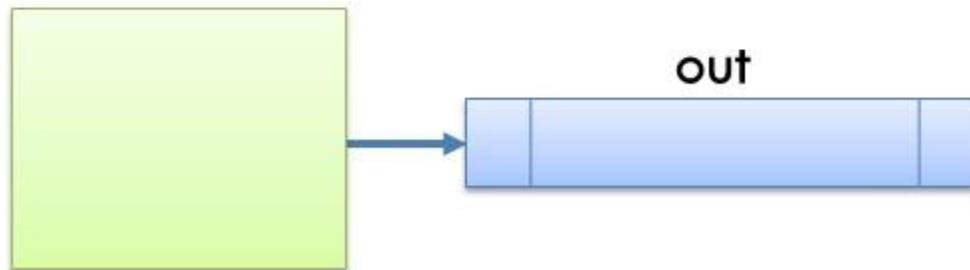
OR

2. opening a file using open() member function

```
ofstream out;
```

```
out.open("abc.txt");
```

prog.cpp



HOW TO OPEN A FILE IN C++ USING OFSTREAM CLASS ?

1. opening a file using constructor

```
ofstream out ("abc.txt");
```

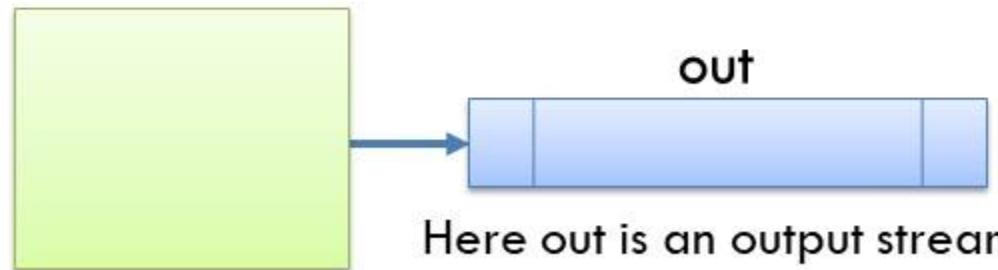
OR

2. opening a file using open() member function

```
ofstream out;
```

```
out.open("abc.txt");
```

prog.cpp



Here out is an output stream

HOW TO OPEN A FILE IN C++ USING OFSTREAM CLASS ?

1. opening a file using constructor

```
ofstream out ("abc.txt");
```

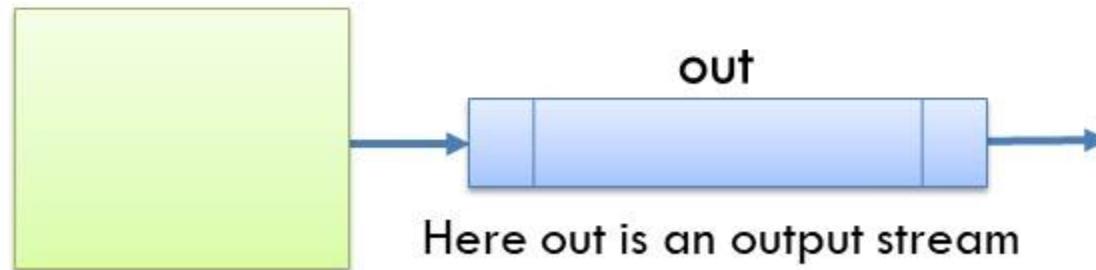
OR

2. opening a file using open() member function

```
ofstream out;
```

```
out.open("abc.txt");
```

prog.cpp



HOW TO OPEN A FILE IN C++ USING OFSTREAM CLASS ?

1. opening a file using constructor

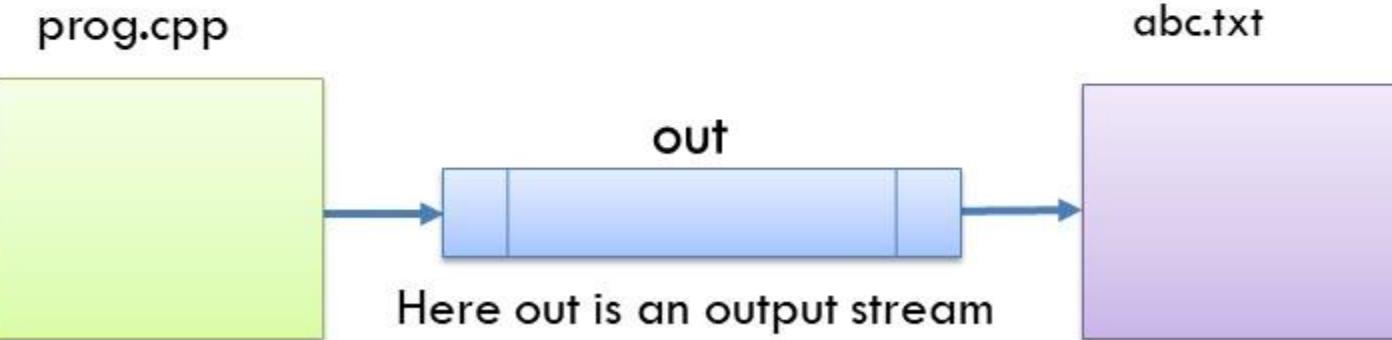
```
ofstream out ("abc.txt");
```

OR

2. opening a file using open() member function

```
ofstream out;
```

```
out.open("abc.txt");
```



HOW TO OPEN A FILE IN C++ USING OFSTREAM CLASS ?

1. opening a file using constructor

```
ofstream out ("abc.txt");
```

OR

2. opening a file using open() member function

```
ofstream out;
```

```
out.open("abc.txt");
```

prog.cpp

out

abc.txt

Here out is an output stream

Note: Here second parameter file mode is not passed as argument because both constructor and open member function of ofstream class has ios::out file mode by default.

HOW TO OPEN A FILE IN C++ USING IFSTREAM CLASS?

HOW TO OPEN A FILE IN C++ USING IFSTREAM CLASS?

1. opening a file using constructor

HOW TO OPEN A FILE IN C++ USING IFSTREAM CLASS?

1. opening a file using constructor

```
ifstream fin ("abc.txt");
```

HOW TO OPEN A FILE IN C++ USING IFSTREAM CLASS?

1. opening a file using constructor

```
ifstream fin ("abc.txt");
```

OR

HOW TO OPEN A FILE IN C++ USING IFSTREAM CLASS?

1. opening a file using constructor

```
ifstream fin ("abc.txt");
```

OR

2. opening a file using open() member function

HOW TO OPEN A FILE IN C++ USING IFSTREAM CLASS?

1. opening a file using constructor

```
ifstream fin ("abc.txt");
```

OR

2. opening a file using open() member function

```
ifstream fin;
```

HOW TO OPEN A FILE IN C++ USING IFSTREAM CLASS?

1. opening a file using constructor

```
ifstream fin ("abc.txt");
```

OR

2. opening a file using open() member function

```
ifstream fin;
```

```
fin.open("abc.txt");
```

HOW TO OPEN A FILE IN C++ USING IFSTREAM CLASS?

1. opening a file using constructor

```
ifstream fin ("abc.txt");
```

OR

2. opening a file using open() member function

```
ifstream fin;
```

```
fin.open("abc.txt");
```

abc.txt



HOW TO OPEN A FILE IN C++ USING IFSTREAM CLASS?

1. opening a file using constructor

```
ifstream fin ("abc.txt");
```

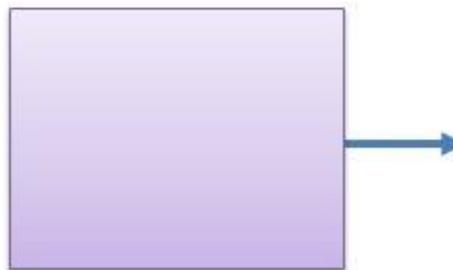
OR

2. opening a file using open() member function

```
ifstream fin;
```

```
fin.open("abc.txt");
```

abc.txt



HOW TO OPEN A FILE IN C++ USING IFSTREAM CLASS?

1. opening a file using constructor

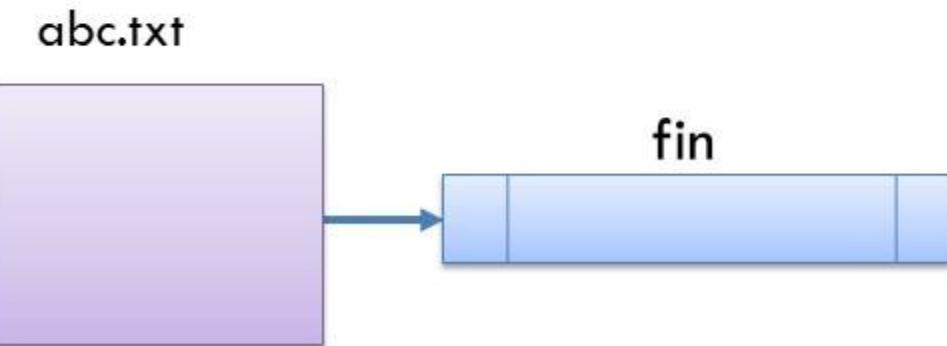
```
ifstream fin ("abc.txt");
```

OR

2. opening a file using open() member function

```
ifstream fin;
```

```
fin.open("abc.txt");
```



HOW TO OPEN A FILE IN C++ USING IFSTREAM CLASS?

1. opening a file using constructor

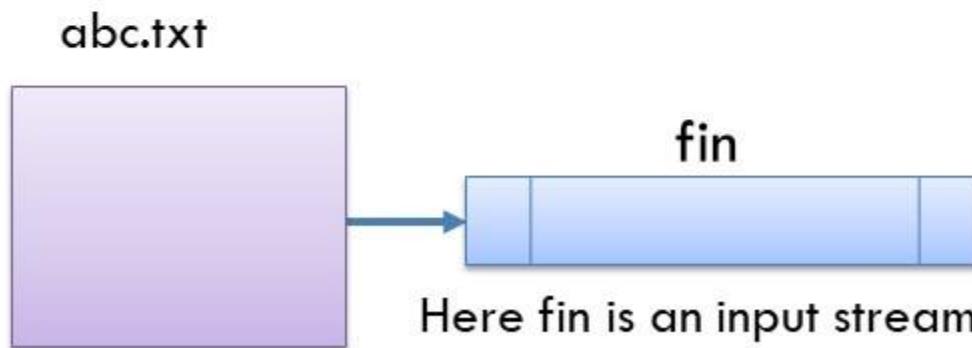
```
ifstream fin ("abc.txt");
```

OR

2. opening a file using open() member function

```
ifstream fin;
```

```
fin.open("abc.txt");
```



HOW TO OPEN A FILE IN C++ USING IFSTREAM CLASS?

1. opening a file using constructor

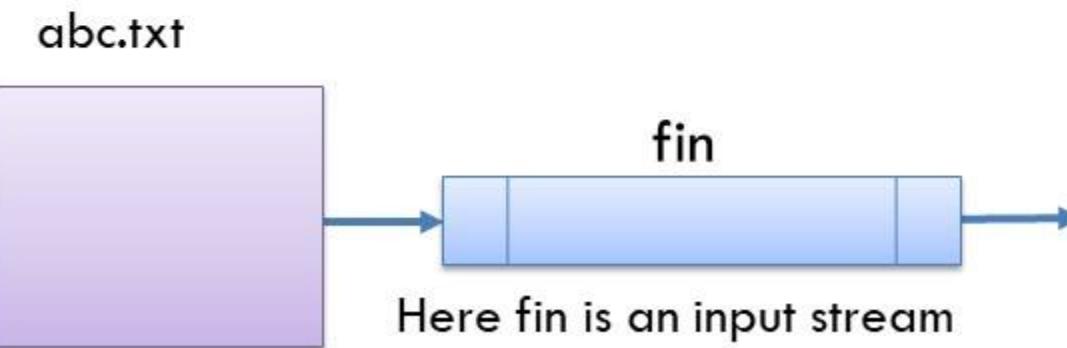
```
ifstream fin ("abc.txt");
```

OR

2. opening a file using open() member function

```
ifstream fin;
```

```
fin.open("abc.txt");
```



HOW TO OPEN A FILE IN C++ USING IFSTREAM CLASS?

1. opening a file using constructor

```
ifstream fin ("abc.txt");
```

OR

2. opening a file using open() member function

```
ifstream fin;
```

```
fin.open("abc.txt");
```



HOW TO OPEN A FILE IN C++ USING IFSTREAM CLASS?

1. opening a file using constructor

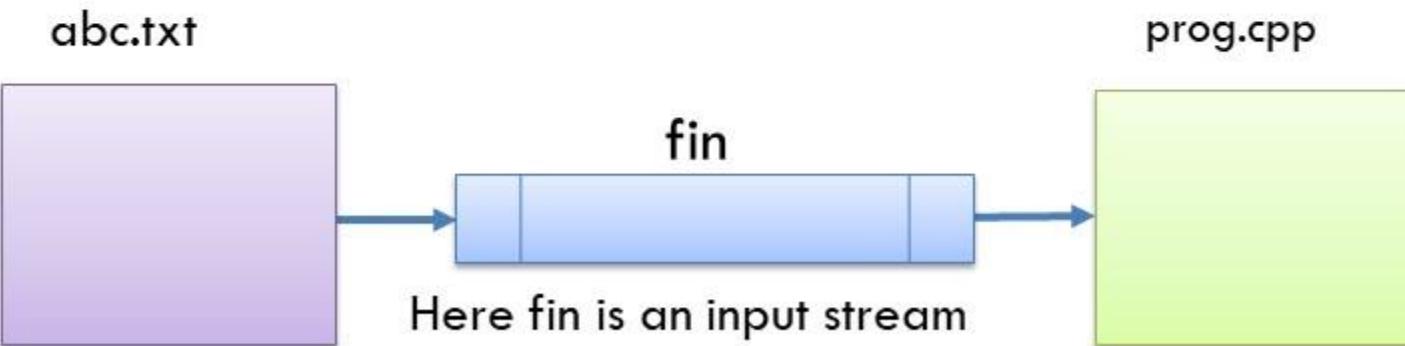
```
ifstream fin ("abc.txt");
```

OR

2. opening a file using open() member function

```
ifstream fin;  
fin.open("abc.txt");
```

Note: Here second parameter file mode is not passed as argument because both constructor and open member function of ifstream class has ios::in file mode by default.



FILE MODE

FILE MODE

- ❖ File mode specifies the purpose for which the file is to be opened.

- ❖ File mode specifies the purpose for which the file is to be opened.
- ❖ The file mode parameter can take one or more of such constants defined in the class **ios**.

FILE MODE PARAMETERS

FILE MODE PARAMETERS

MEANING

FILE MODE PARAMETERS

FILE MODE PARAMETERS	MEANING
1. ios::out	

FILE MODE PARAMETERS

FILE MODE PARAMETERS

1. ios::out

MEANING

1. Open file for writing only.

FILE MODE PARAMETERS

FILE MODE PARAMETERS

1. ios::out
2. ios::in

MEANING

1. Open file for writing only.

FILE MODE PARAMETERS

FILE MODE PARAMETERS

1. ios::out
2. ios::in

MEANING

1. Open file for writing only.
2. Open file for reading only.

FILE MODE PARAMETERS

FILE MODE PARAMETERS	MEANING
1. ios::out	1. Open file for writing only.
2. ios::in	2. Open file for reading only.
3. ios::app	

FILE MODE PARAMETERS

FILE MODE PARAMETERS	MEANING
1. ios::out	1. Open file for writing only.
2. ios::in	2. Open file for reading only.
3. ios::app	3. Append to end-of-file.

FILE MODE PARAMETERS

FILE MODE PARAMETERS	MEANING
1. ios::out	1. Open file for writing only.
2. ios::in	2. Open file for reading only.
3. ios::app	3. Append to end-of-file.
4. ios::ate	

FILE MODE PARAMETERS

FILE MODE PARAMETERS	MEANING
1. ios::out	1. Open file for writing only.
2. ios::in	2. Open file for reading only.
3. ios::app	3. Append to end-of-file.
4. ios::ate	4. Go to end-of-file on opening.

FILE MODE PARAMETERS

FILE MODE PARAMETERS	MEANING
1. ios::out	1. Open file for writing only.
2. ios::in	2. Open file for reading only.
3. ios::app	3. Append to end-of-file.
4. ios::ate	4. Go to end-of-file on opening.
5. ios::binary	

FILE MODE PARAMETERS

FILE MODE PARAMETERS	MEANING
1. ios::out	1. Open file for writing only.
2. ios::in	2. Open file for reading only.
3. ios::app	3. Append to end-of-file.
4. ios::ate	4. Go to end-of-file on opening.
5. ios::binary	5. Creates a binary file.

FILE MODE PARAMETERS

FILE MODE PARAMETERS	MEANING
1. ios::out	1. Open file for writing only.
2. ios::in	2. Open file for reading only.
3. ios::app	3. Append to end-of-file.
4. ios::ate	4. Go to end-of-file on opening.
5. ios::binary	5. Creates a binary file.
6. ios::nocreate	

FILE MODE PARAMETERS

FILE MODE PARAMETERS	MEANING
1. ios::out	1. Open file for writing only.
2. ios::in	2. Open file for reading only.
3. ios::app	3. Append to end-of-file.
4. ios::ate	4. Go to end-of-file on opening.
5. ios::binary	5. Creates a binary file.
6. ios::nocreate	6. Open fails if the file does not exists.

FILE MODE PARAMETERS

FILE MODE PARAMETERS	MEANING
1. ios::out	1. Open file for writing only.
2. ios::in	2. Open file for reading only.
3. ios::app	3. Append to end-of-file.
4. ios::ate	4. Go to end-of-file on opening.
5. ios::binary	5. Creates a binary file.
6. ios::nocreate	6. Open fails if the file does not exists.
7. ios::noreplace	

FILE MODE PARAMETERS

FILE MODE PARAMETERS	MEANING
1. ios::out	1. Open file for writing only.
2. ios::in	2. Open file for reading only.
3. ios::app	3. Append to end-of-file.
4. ios::ate	4. Go to end-of-file on opening.
5. ios::binary	5. Creates a binary file.
6. ios::nocreate	6. Open fails if the file does not exists.
7. ios::noreplace	7. Open fails if the file already exists.

FILE MODE PARAMETERS

FILE MODE PARAMETERS	MEANING
1. ios::out	1. Open file for writing only.
2. ios::in	2. Open file for reading only.
3. ios::app	3. Append to end-of-file.
4. ios::ate	4. Go to end-of-file on opening.
5. ios::binary	5. Creates a binary file.
6. ios::nocreate	6. Open fails if the file does not exists.
7. ios::noreplace	7. Open fails if the file already exists.
8. ios::trunc	

FILE MODE PARAMETERS

FILE MODE PARAMETERS	MEANING
1. ios::out	1. Open file for writing only.
2. ios::in	2. Open file for reading only.
3. ios::app	3. Append to end-of-file.
4. ios::ate	4. Go to end-of-file on opening.
5. ios::binary	5. Creates a binary file.
6. ios::nocreate	6. Open fails if the file does not exists.
7. ios::noreplace	7. Open fails if the file already exists.
8. ios::trunc	8. Delete the contents of the file if it exists.

IMPORTANT POINTS TO REMEMBER ABOUT FILE MODES

IMPORTANT POINTS TO REMEMBER ABOUT FILE MODES

1. Opening a file in **ios::out** mode also opens it in the **ios::trunc** mode by default.

IMPORTANT POINTS TO REMEMBER ABOUT FILE MODES

1. Opening a file in **ios::out** mode also opens it in the **ios::trunc** mode by default.
2. Both **ios::app** and **ios::ate** takes us to the end of the file when it is opened.

IMPORTANT POINTS TO REMEMBER ABOUT FILE MODES

1. Opening a file in **ios::out** mode also opens it in the **ios::trunc** mode by default.
2. Both **ios::app** and **ios::ate** takes us to the end of the file when it is opened.

The difference between the two parameters is that the

IMPORTANT POINTS TO REMEMBER ABOUT FILE MODES

1. Opening a file in **ios::out** mode also opens it in the **ios::trunc** mode by default.
2. Both **ios::app** and **ios::ate** takes us to the end of the file when it is opened.

The difference between the two parameters is that the **ios::app** mode allows us to add data to the end-of-file only, while

IMPORTANT POINTS TO REMEMBER ABOUT FILE MODES

1. Opening a file in **ios::out** mode also opens it in the **ios::trunc** mode by default.
2. Both **ios::app** and **ios::ate** takes us to the end of the file when it is opened.

The difference between the two parameters is that the **ios::app** mode allows us to add data to the end-of-file only, while **ios::ate** mode allows us to add data or to modify the existing data anywhere in the file.

IMPORTANT POINTS TO REMEMBER ABOUT FILE MODES

1. Opening a file in **ios::out** mode also opens it in the **ios::trunc** mode by default.
2. Both **ios::app** and **ios::ate** takes us to the end of the file when it is opened.

The difference between the two parameters is that the **ios::app** mode allows us to add data to the end-of-file only, while **ios::ate** mode allows us to add data or to modify the existing data anywhere in the file.

In both the cases, a file is created by the specified name, if it does not exists.

IMPORTANT POINTS TO REMEMBER ABOUT FILE MODES

IMPORTANT POINTS TO REMEMBER ABOUT FILE MODES

3. The parameter **ios::app** can be used only with the files capable of output.

IMPORTANT POINTS TO REMEMBER ABOUT FILE MODES

3. The parameter **ios::app** can be used only with the files capable of output.
4. Creating a stream using **ifstream** class implies input and

IMPORTANT POINTS TO REMEMBER ABOUT FILE MODES

3. The parameter **ios::app** can be used only with the files capable of output.
4. Creating a stream using **ifstream** class implies input and
Creating a stream using **ofstream** class implies output.

IMPORTANT POINTS TO REMEMBER ABOUT FILE MODES

3. The parameter **ios::app** can be used only with the files capable of output.
4. Creating a stream using **ifstream** class implies input and
Creating a stream using **ofstream** class implies output.
So in these cases, it is not necessary to provide the file mode parameters.

IMPORTANT POINTS TO REMEMBER ABOUT FILE MODES

3. The parameter **ios::app** can be used only with the files capable of output.
4. Creating a stream using **ifstream** class implies input and
Creating a stream using **ofstream** class implies output.
So in these cases, it is not necessary to provide the file mode parameters.
5. The **fstream** class does not provide a mode by default and therefore, we must provide the mode explicitly when using an object of **fstream** class.

IMPORTANT POINTS TO REMEMBER ABOUT FILE MODES

IMPORTANT POINTS TO REMEMBER ABOUT FILE MODES

6. The mode can combine two or more parameters using the bitwise OR operator (symbol |) as shown below:

IMPORTANT POINTS TO REMEMBER ABOUT FILE MODES

6. The mode can combine two or more parameters using the bitwise OR operator (symbol |) as shown below:

```
fout.open("data",ios::app | ios::nocreate);
```

IMPORTANT POINTS TO REMEMBER ABOUT FILE MODES

6. The mode can combine two or more parameters using the bitwise OR operator (symbol |) as shown below:

```
fout.open("data",ios::app | ios::nocreate);
```

This opens the file in the append mode but fails to open the file if it does not exist.

CLOSING A FILE

CLOSING A FILE

When any C++ program terminates,

CLOSING A FILE

When any C++ program terminates,
it automatically flushes out all the streams,

CLOSING A FILE

When any C++ program terminates,
it automatically flushes out all the streams,
releases all the allocated memory and closes all the
opened files.

CLOSING A FILE

When any C++ program terminates,
it automatically flushes out all the streams,
releases all the allocated memory and closes all the
opened files.

But it is good to use the `close()` function to close the file
related streams

CLOSING A FILE

When any C++ program terminates,
it automatically flushes out all the streams,
releases all the allocated memory and closes all the
opened files.

But it is good to use the `close()` function to close the file
related streams
and it is a member of `ifstream`, `ofstream` and `fstream`
objects.

CLOSING A FILE

CLOSING A FILE

For example ,

CLOSING A FILE

For example ,

```
ofstream out ("abc.txt");
```

CLOSING A FILE

For example ,

```
ofstream out ("abc.txt");  
out.close();
```

CLOSING A FILE

For example ,

```
ofstream out ("abc.txt");  
out.close();
```

Disconnects the file abc.txt from the output stream **out**.

CLOSING A FILE

For example ,

```
ofstream out ("abc.txt");  
out.close();
```

Disconnects the file abc.txt from the output stream **out**.

Remember that the object **out** still exists and the abc.txt file may again be connected to **out** later or to any other stream.

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR (PROG1)

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR (PROG1)

Write a Program to

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR (PROG1)

Write a Program to

Open a file named Product.txt in write mode

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR (PROG1)

Write a Program to

Open a file named Product.txt in write mode

Write Product Name and Product Cost in the file Product.txt

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR (PROG1)

Write a Program to

Open a file named Product.txt in write mode

Write Product Name and Product Cost in the file Product.txt

Close the file Product.txt

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR (PROG1)

Write a Program to

Open a file named Product.txt in write mode

Write Product Name and Product Cost in the file Product.txt

Close the file Product.txt

Open the file Product.txt in read mode

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR (PROG1)

Write a Program to

Open a file named Product.txt in write mode

Write Product Name and Product Cost in the file Product.txt

Close the file Product.txt

Open the file Product.txt in read mode

Read the contents of the file and store it in a variable

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR (PROG1)

Write a Program to

Open a file named Product.txt in write mode

Write Product Name and Product Cost in the file Product.txt

Close the file Product.txt

Open the file Product.txt in read mode

Read the contents of the file and store it in a variable

Close the file

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR (PROG1)

Write a Program to

Open a file named Product.txt in write mode

Write Product Name and Product Cost in the file Product.txt

Close the file Product.txt

Open the file Product.txt in read mode

Read the contents of the file and store it in a variable

Close the file

display the contents of the file on output screen (console/Black screen)

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

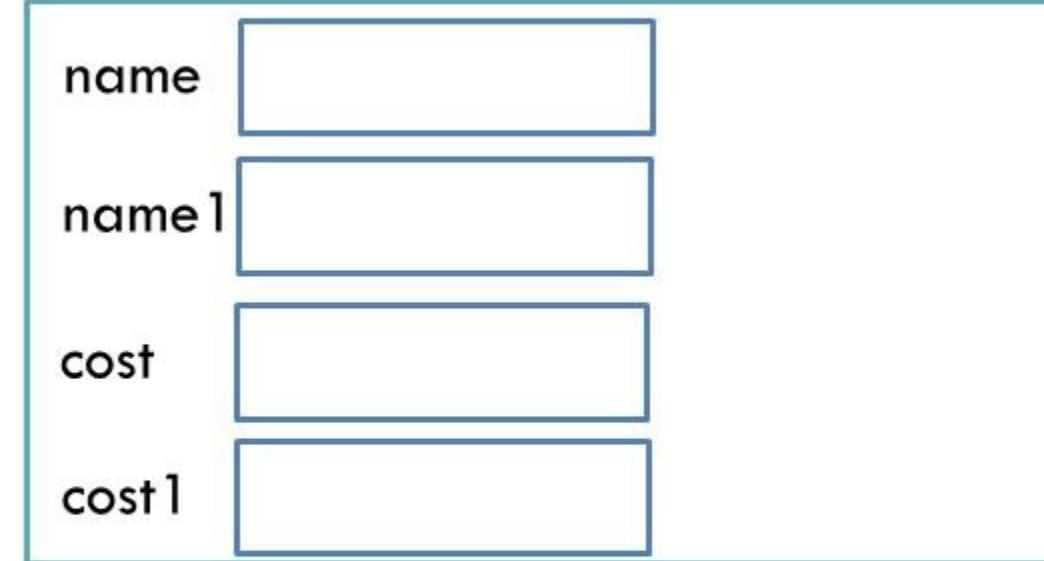
WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

```
#include<iostream>
#include<fstream>
using namespace std;
int main() {
```

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

```
#include<iostream>
#include<fstream>
using namespace std;
int main() {
    string name, name1;
    float cost, cost1;
```

Memory Allocation

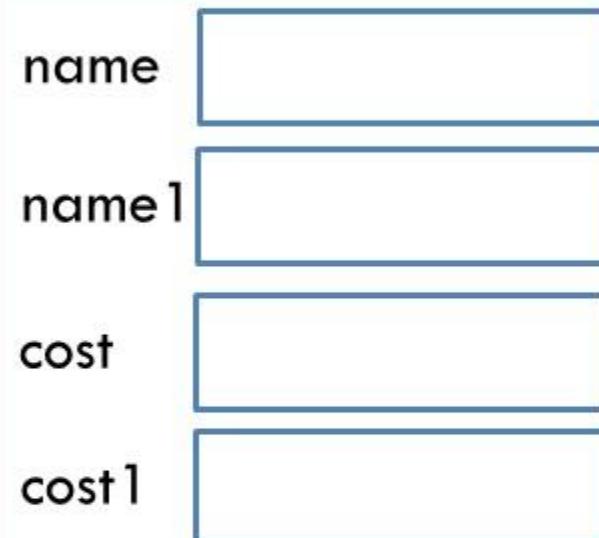


WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

```
#include<iostream>
#include<fstream>
using namespace std;
int main() {
    string name, name1;
    float cost, cost1;

    cout<<"Enter Product Name:";
```

Memory Allocation



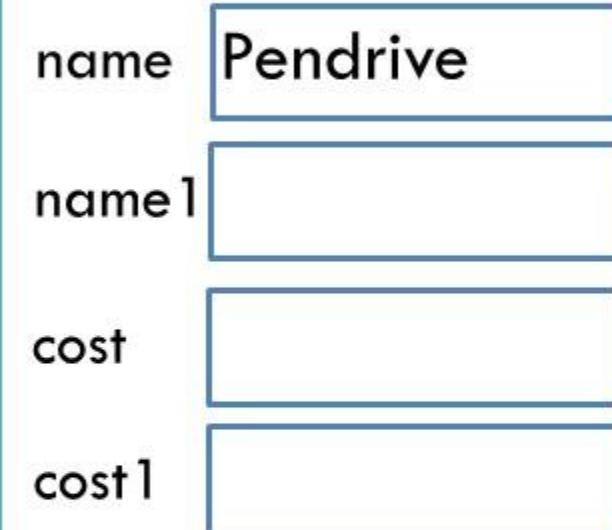
Enter Product Name:

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

```
#include<iostream>
#include<fstream>
using namespace std;
int main() {
    string name, name1;
    float cost, cost1;

    cout<<"Enter Product Name:";
    cin>>name;
```

Memory Allocation



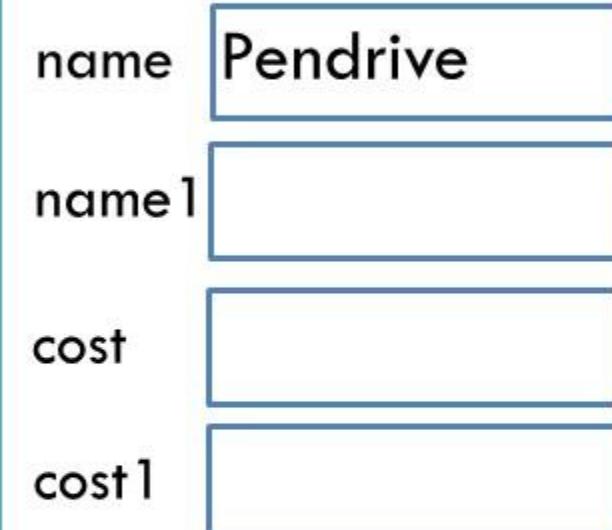
Enter Product Name:Pendrive

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

```
#include<iostream>
#include<fstream>
using namespace std;
int main() {
    string name, name1;
    float cost, cost1;

    cout<<"Enter Product Name:";
    cin>>name;
    cout<<"Enter Product Cost:";
```

Memory Allocation



```
Enter Product Name:Pendrive
Enter Product Cost:
```

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

```
#include<iostream>
#include<fstream>
using namespace std;
int main() {
    string name, name1;
    float cost, cost1;

    cout<<"Enter Product Name:";
    cin>>name;
    cout<<"Enter Product Cost:";
    cin>>cost;
```

Memory Allocation

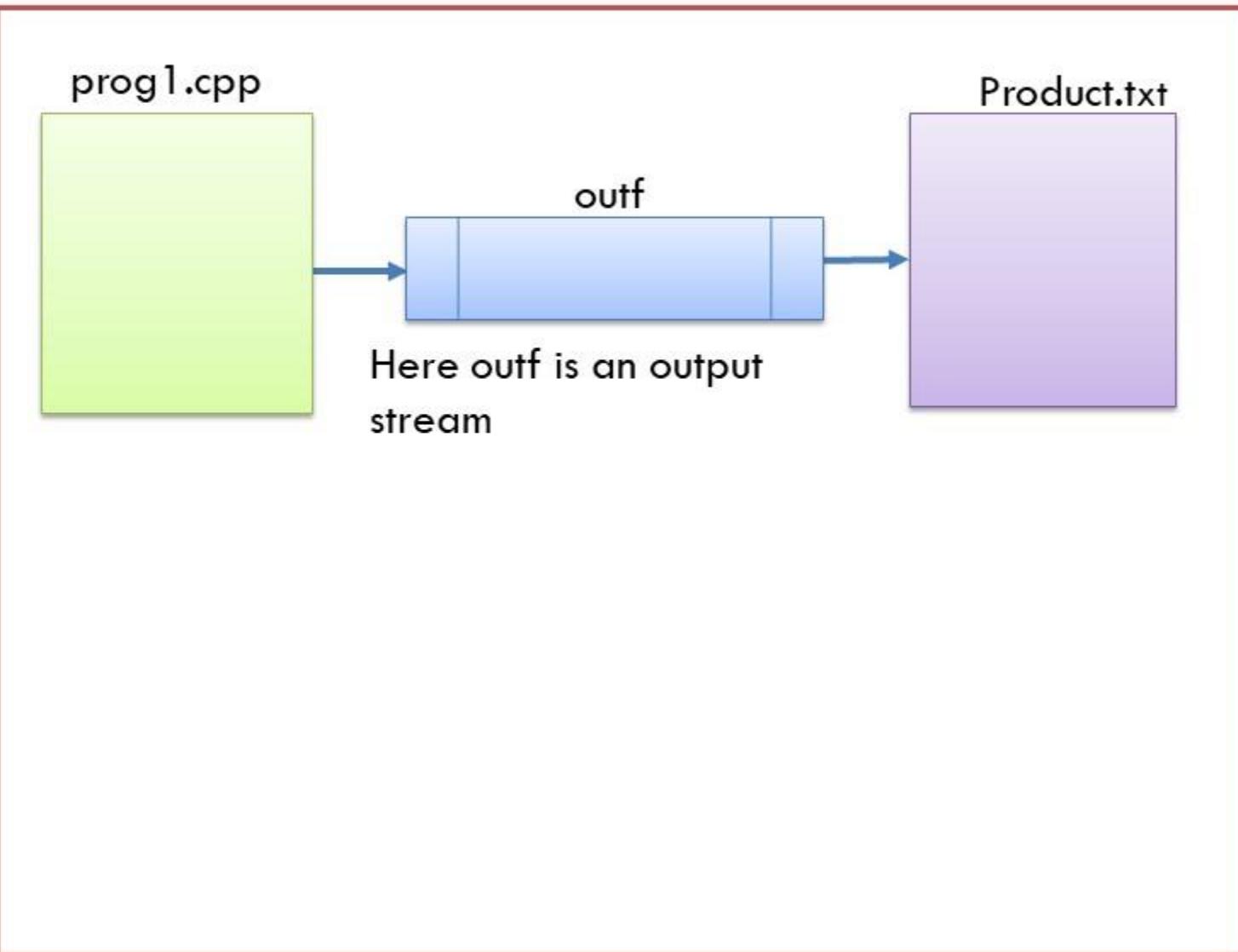
name	Pendrive
name1	
cost	300
cost1	

```
Enter Product Name:Pendrive
Enter Product Cost: 300
```

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

```
ofstream outf ("Product.txt");
```



WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

```
ofstream outf ("Product.txt");
```

```
outf<<name<<endl;
```

prog1.cpp

outf

Here outf is an output
stream

Product.txt

Pendrive

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

```
ofstream outf ("Product.txt");
```

```
outf<<name<<endl;
```

```
outf<<cost<<endl;
```

prog1.cpp



Here outf is an output stream

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

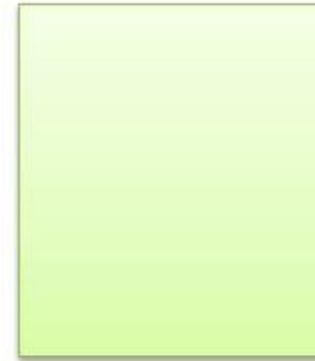
```
ofstream outf ("Product.txt");
```

```
outf<<name<<endl;
```

```
outf<<cost<<endl;
```

```
outf.close();
```

prog1.cpp



outf

Here outf is an output
stream

Product.txt

Pendrive
300

prog1.cpp



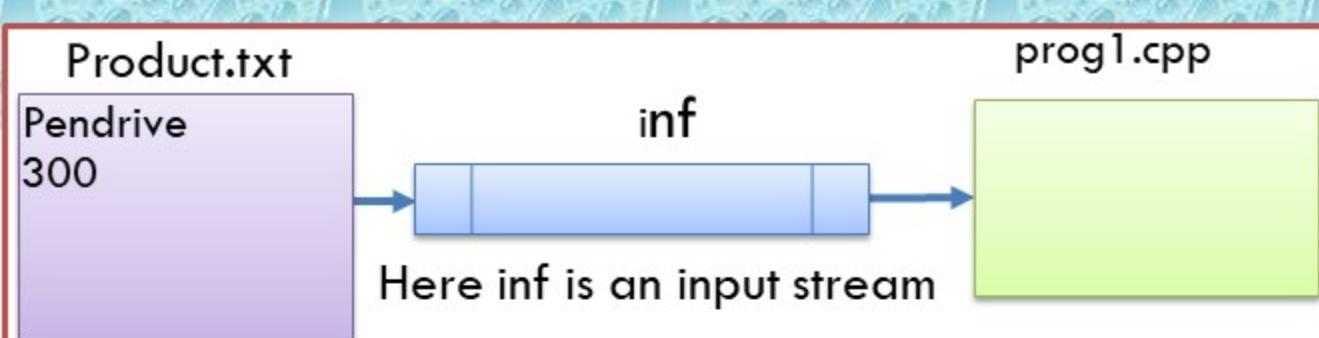
outf

outf is disconnected from
product.txt

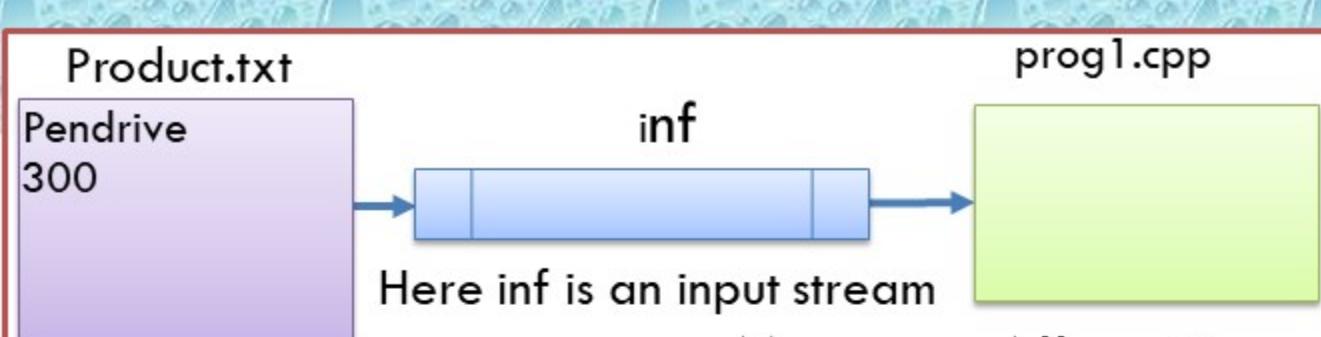
Product.txt

Pendrive
300


```
ifstream inf("Product.txt");
```

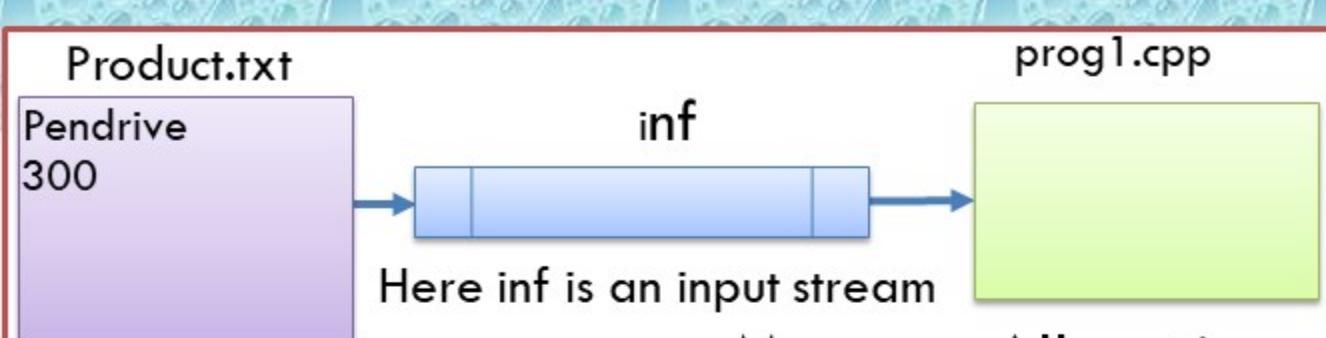


```
ifstream inf("Product.txt");
```



```
ifstream inf("Product.txt");
```

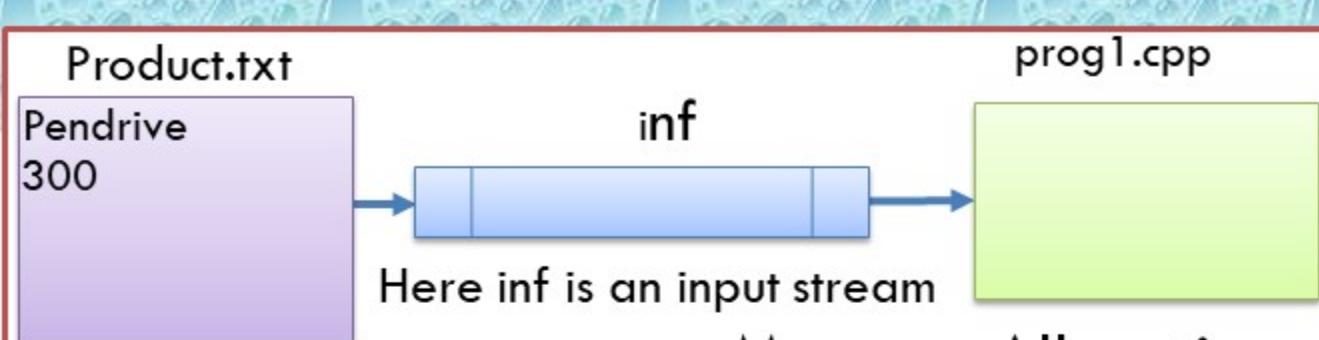
```
inf>>name1;
```



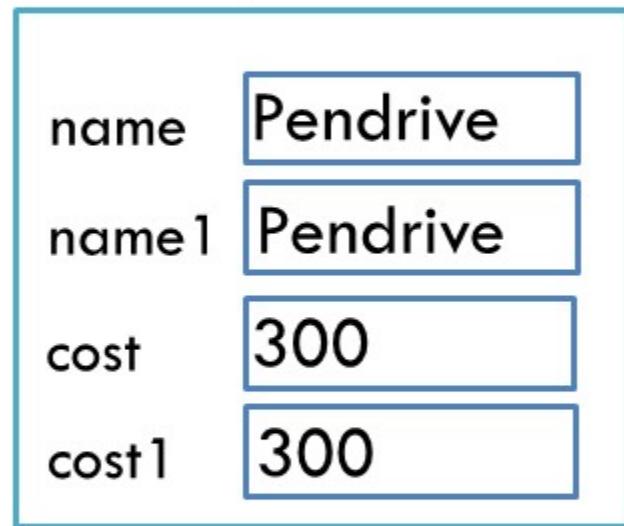
```
ifstream inf("Product.txt");
```

```
inf>>name1;
```

```
inf>>cost1;
```



Memory Allocation

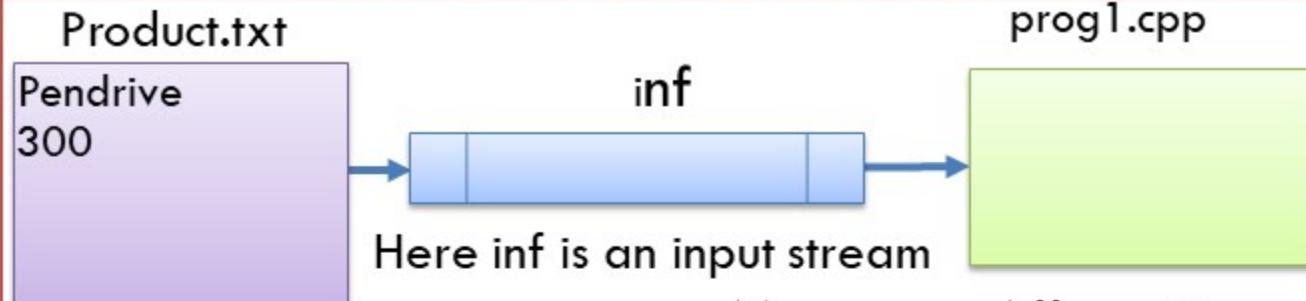


```
ifstream inf("Product.txt");
```

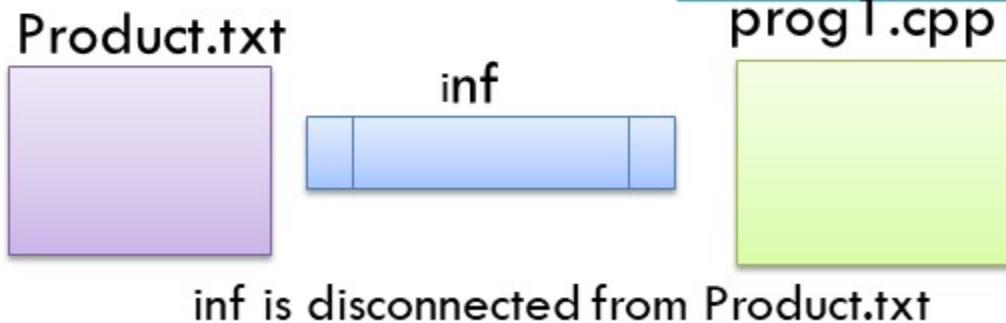
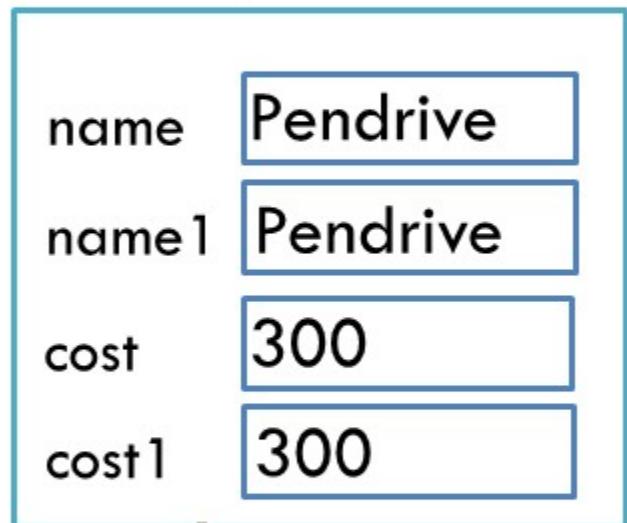
```
inf>>name1;
```

```
inf>>cost1;
```

```
inf.close();
```



Memory Allocation



WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

Enter Product Name: Pendrive

Enter Product Cost: 300

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

```
cout<<endl<<"Product Name:"<<name1<<endl;
```

Enter Product Name:Pendrive

Enter Product Cost: 300

Product Name: Pendrive

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

```
cout<<endl<<"Product Name:"<<name1<<endl;  
  
cout<<endl<<"Product Cost:"<<cost1<<endl;
```

Enter Product Name:Pendrive

Enter Product Cost: 300

Product Name: Pendrive
Product Cost: 300

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

```
cout<<endl<<"Product Name:"<<name1<<endl;  
cout<<endl<<"Product Cost:"<<cost1<<endl;  
return 0;  
}
```

Enter Product Name:Pendrive

Enter Product Cost: 300

Product Name: Pendrive
Product Cost: 300

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

OUTPUT:

```
Enter Product Name:Pendrive
```

```
Enter Product Cost:300
```

```
Product Name:Pendrive
```

```
Product Cost:300
```

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

OUTPUT:

```
Enter Product Name:Pendrive
```

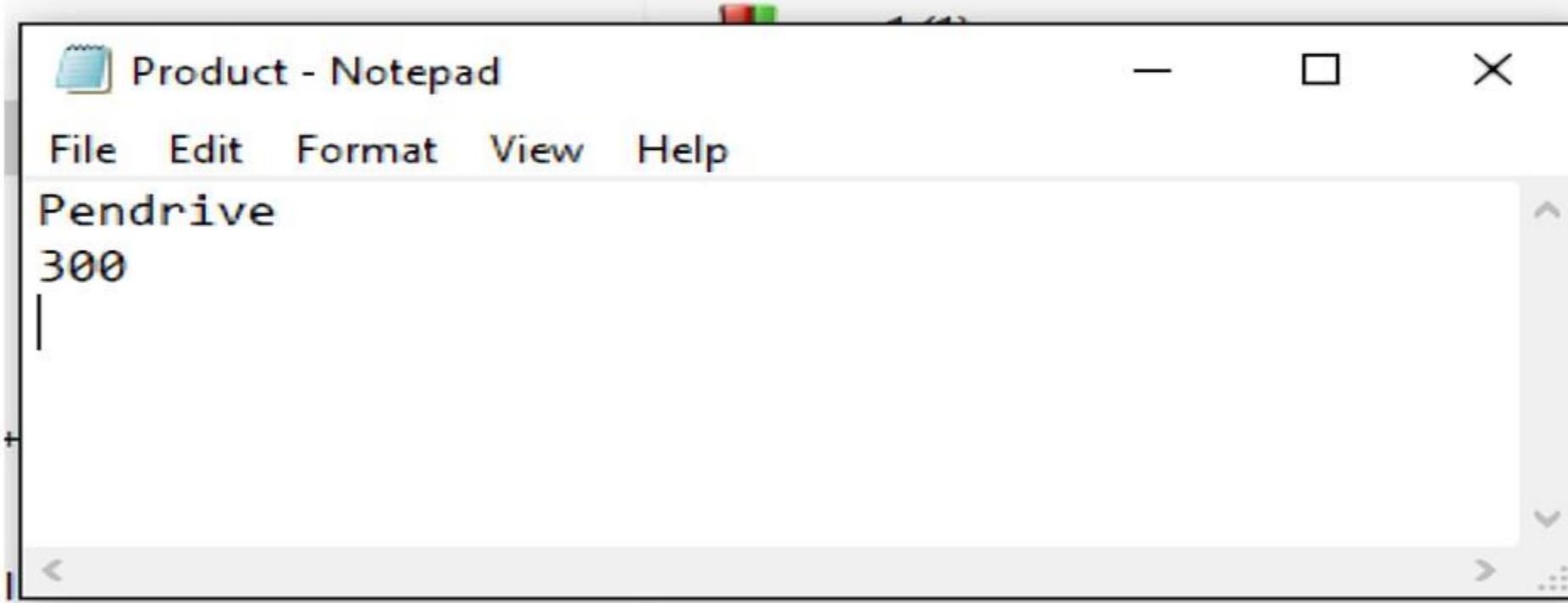
```
Enter Product Cost:300
```

```
Product Name:Pendrive
```

```
Product Cost:300
```

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

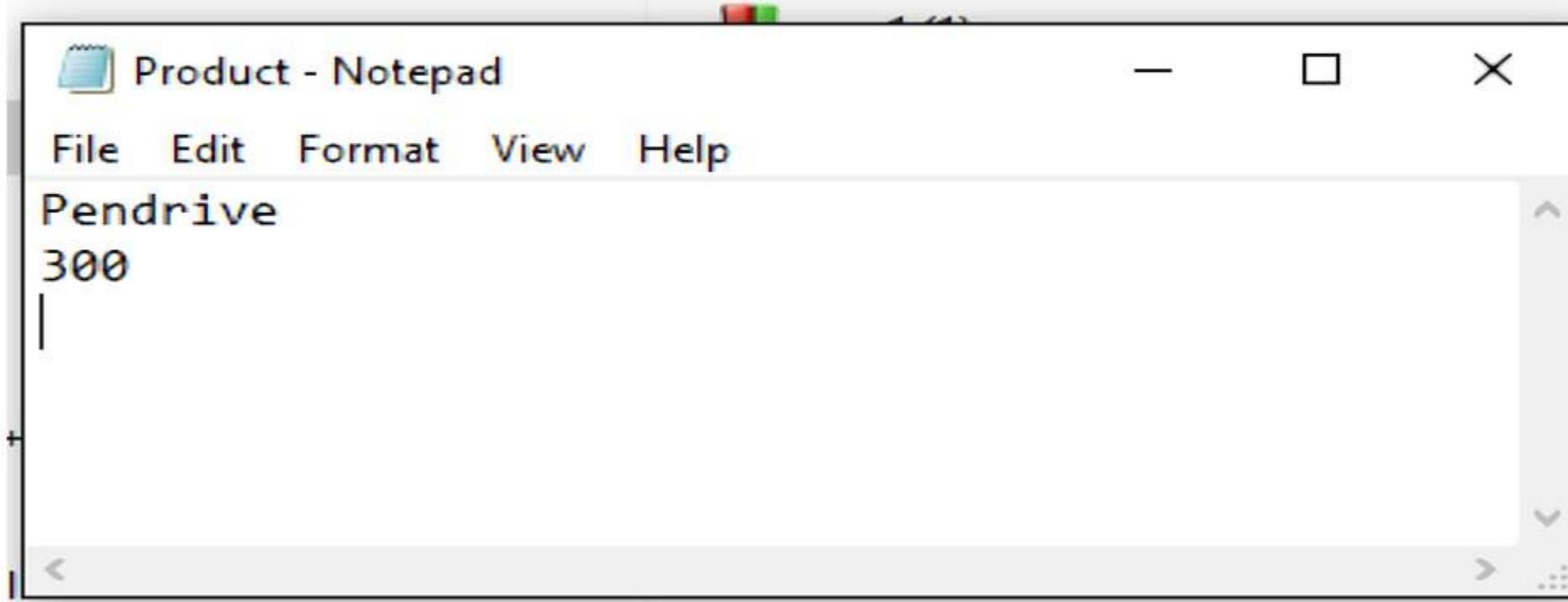
Product.txt file



Prog1.cpp

WORKING WITH SINGLE FILE: CREATING FILES WITH CONSTRUCTOR

Product.txt file



Prog1.cpp

GETLINE FUNCTION FOR CONSOLE

GETLINE FUNCTION FOR CONSOLE

Syntax: `cin.getline(line , size);`

GETLINE FUNCTION FOR CONSOLE

Syntax: `cin.getline(line , size);`

For example,

GETLINE FUNCTION FOR CONSOLE

Syntax: `cin.getline(line , size);`

For example,

```
char name[20];
```

Memory Allocation

name

GETLINE FUNCTION FOR CONSOLE

Syntax: `cin.getline(line , size);`

For example,

```
char name[20];  
cin.getline(name,20);
```

Assume we give following input through keyboard,

Mayuri Thakkar <press RETURN>

Memory Allocation

name	Mayuri Thakkar
------	----------------

GETLINE FUNCTION FOR CONSOLE

Syntax: `cin.getline(line , size);`

For example,

```
char name[20];  
cin.getline(name,20);
```

Assume we give following input through keyboard,

Mayuri Thakkar <press RETURN>

The input will be read correctly and assigned
to the character array **name**

Memory Allocation

name	Mayuri Thakkar
------	----------------

GETLINE FUNCTION FOR CONSOLE

Syntax: `cin.getline(line , size);`

For example,

```
char name[20];  
cin.getline(name,20);
```

Assume we give following input through keyboard,

Mayuri Thakkar <press RETURN>

The input will be read correctly and assigned
to the character array **name**

Let us suppose we give following input

Object Oriented Programming <press RETURN>

Memory Allocation

name Object Oriented Pro

GETLINE FUNCTION FOR CONSOLE

Syntax: `cin.getline(line , size);`

For example,

```
char name[20];  
cin.getline(name,20);
```

Assume we give following input through keyboard,

Mayuri Thakkar <press RETURN>

The input will be read correctly and assigned
to the character array **name**

Let us suppose we give following input

Object Oriented Programming <press RETURN>

As the size given is 20 it will read only first 19 characters from object oriented
Programming and 20th character null '\0'

Memory Allocation

name Object Oriented Pro

DETECTING END-OF-FILE

DETECTING END-OF-FILE

- Detection of the end-of-file condition is necessary for preventing any further attempt to read data from the file.

DETECTING END-OF-FILE

- Detection of the end-of-file condition is necessary for preventing any further attempt to read data from the file.
- This can be done in two ways:

DETECTING END-OF-FILE

- Detection of the end-of-file condition is necessary for preventing any further attempt to read data from the file.
- This can be done in two ways:
 1. An ifstream object such as fin , returns a value of 0 if any error occurs in the file operation including the end-of-file condition.
 - 2.

DETECTING END-OF-FILE

- Detection of the end-of-file condition is necessary for preventing any further attempt to read data from the file.
- This can be done in two ways:
 1. An ifstream object such as fin , returns a value of 0 if any error occurs in the file operation including the end-of-file condition.
For example, `while(fin) { some statements; }`

DETECTING END-OF-FILE

- Detection of the end-of-file condition is necessary for preventing any further attempt to read data from the file.
- This can be done in two ways:
 1. An ifstream object such as fin , returns a value of 0 if any error occurs in the file operation including the end-of-file condition.
For example, `while(fin) { some statements; }`
The while loop terminates when fin returns a value of zero on reaching end of file.

DETECTING END-OF-FILE

- Detection of the end-of-file condition is necessary for preventing any further attempt to read data from the file.
- This can be done in two ways:
 1. An ifstream object such as fin , returns a value of 0 if any error occurs in the file operation including the end-of-file condition.
For example, `while(fin) { some statements; }`
The while loop terminates when fin returns a value of zero on reaching end of file.
This loop may terminate due to other failures as well.

DETECTING END-OF-FILE

DETECTING END-OF-FILE

2. Another method to detect the end-of-file is `eof()` function.

DETECTING END-OF-FILE

2. Another method to detect the end-of-file is `eof()` function.
`eof()` is a member function of `ios` class.

DETECTING END-OF-FILE

2. Another method to detect the end-of-file is `eof()` function.

`eof()` is a member function of `ios` class.

It returns true, (non-zero value) if the end -of- file (EOF) condition is encountered

DETECTING END-OF-FILE

2. Another method to detect the end-of-file is `eof()` function.

`eof()` is a member function of `ios` class.

It returns true, (non-zero value) if the end -of- file (EOF) condition is encountered

And it returns false , (zero value) if end-of-file is not encountered.

DETECTING END-OF-FILE

2. Another method to detect the end-of-file is `eof()` function.

`eof()` is a member function of `ios` class.

It returns true, (non-zero value) if the end -of- file (EOF) condition is encountered

And it returns false , (zero value) if end-of-file is not encountered.

For example,

DETECTING END-OF-FILE

2. Another method to detect the end-of-file is `eof()` function.

`eof()` is a member function of `ios` class.

It returns true, (non-zero value) if the end -of- file (EOF) condition is encountered

And it returns false , (zero value) if end-of-file is not encountered.

For example,

```
if(fin.eof!=0) { exit(1); }
```

DETECTING END-OF-FILE

2. Another method to detect the end-of-file is `eof()` function.

`eof()` is a member function of `ios` class.

It returns true, (non-zero value) if the end -of- file (EOF) condition is encountered

And it returns false , (zero value) if end-of-file is not encountered.

For example,

```
if(fin.eof!=0) { exit(1); }
```

The above statement terminates the program on reaching the end of the file.

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN (PROG2)

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN (PROG2)

Write a Program to

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN (PROG2)

Write a Program to

Create an object of class either ofstream or fstream in write mode

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN (PROG2)

Write a Program to

Create an object of class either ofstream or fstream in write mode

create a file country.txt in write mode using open member function

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN (PROG2)

Write a Program to

Create an object of class either ofstream or fstream in write mode

create a file country.txt in write mode using open member function

Write 3 country names in country.txt file

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN (PROG2)

Write a Program to

Create an object of class either ofstream or fstream in write mode

create a file country.txt in write mode using open member function

Write 3 country names in country.txt file

Close the country file

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN (PROG2)

Write a Program to

Create an object of class either ofstream or fstream in write mode

create a file country.txt in write mode using open member function

Write 3 country names in country.txt file

Close the country file

create a file capital.txt in write mode using open member function

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN (PROG2)

Write a Program to

Create an object of class either ofstream or fstream in write mode

create a file country.txt in write mode using open member function

Write 3 country names in country.txt file

Close the country file

create a file capital.txt in write mode using open member function

Write 3 capital names in capital.txt file

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN (PROG2)

Write a Program to

Create an object of class either ofstream or fstream in write mode

create a file country.txt in write mode using open member function

Write 3 country names in country.txt file

Close the country file

create a file capital.txt in write mode using open member function

Write 3 capital names in capital.txt file

Close the capital file

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

Open the country file in read mode

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

Open the country file in read mode

Read the line from country file using `getline()` function

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

Open the country file in read mode

Read the line from country file using `getline()` function

Display the line in output screen (console/Black screen)

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

Open the country file in read mode

Read the line from country file using `getline()` function

Display the line in output screen (console/Black screen)

Do this upto end of file

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

Open the country file in read mode

Read the line from country file using `getline()` function

Display the line in output screen (console/Black screen)

Do this upto end of file

Close the country file

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

Open the country file in read mode

Read the line from country file using `getline()` function

Display the line in output screen (console/Black screen)

Do this upto end of file

Close the country file

Open the capital file in read mode

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

Open the country file in read mode

Read the line from country file using `getline()` function

Display the line in output screen (console/Black screen)

Do this upto end of file

Close the country file

Open the capital file in read mode

Read the line from capital file using `getline()` function

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

Open the country file in read mode

Read the line from country file using `getline()` function

Display the line in output screen (console/Black screen)

Do this upto end of file

Close the country file

Open the capital file in read mode

Read the line from capital file using `getline()` function

Display the line in output screen (console/Black screen)

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

Open the country file in read mode

Read the line from country file using `getline()` function

Display the line in output screen (console/Black screen)

Do this upto end of file

Close the country file

Open the capital file in read mode

Read the line from capital file using `getline()` function

Display the line in output screen (console/Black screen)

Do this upto end of file

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

Open the country file in read mode

Read the line from country file using `getline()` function

Display the line in output screen (console/Black screen)

Do this upto end of file

Close the country file

Open the capital file in read mode

Read the line from capital file using `getline()` function

Display the line in output screen (console/Black screen)

Do this upto end of file

Close the country file

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

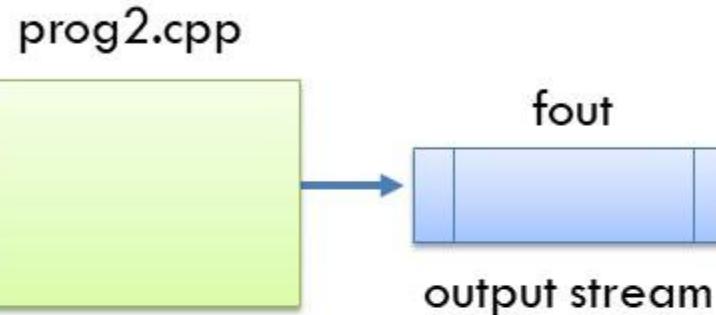
```
#include<iostream>
#include<fstream>
using namespace std;
```

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
#include<iostream>
#include<fstream>
using namespace std;
int main() {
```

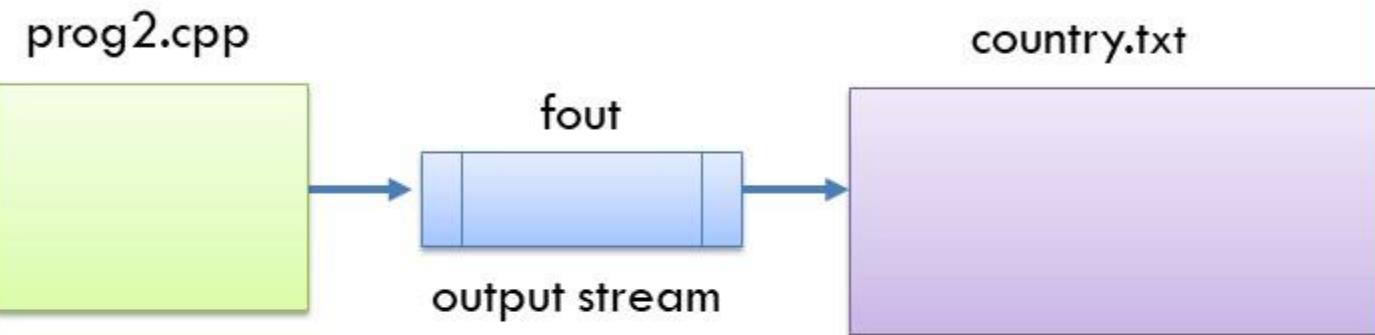
WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
#include<iostream>
#include<fstream>
using namespace std;
int main() {
ofstream fout;
```



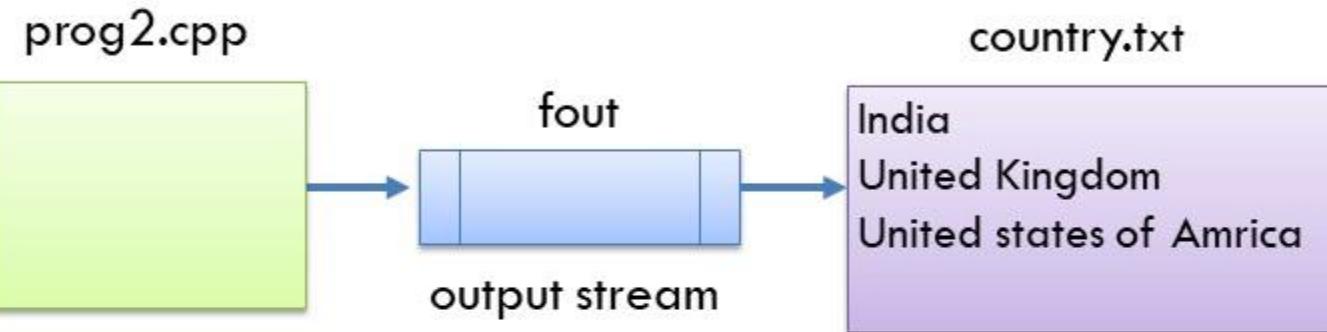
WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
#include<iostream>
#include<fstream>
using namespace std;
int main() {
ofstream fout;
fout.open("country.txt");
```



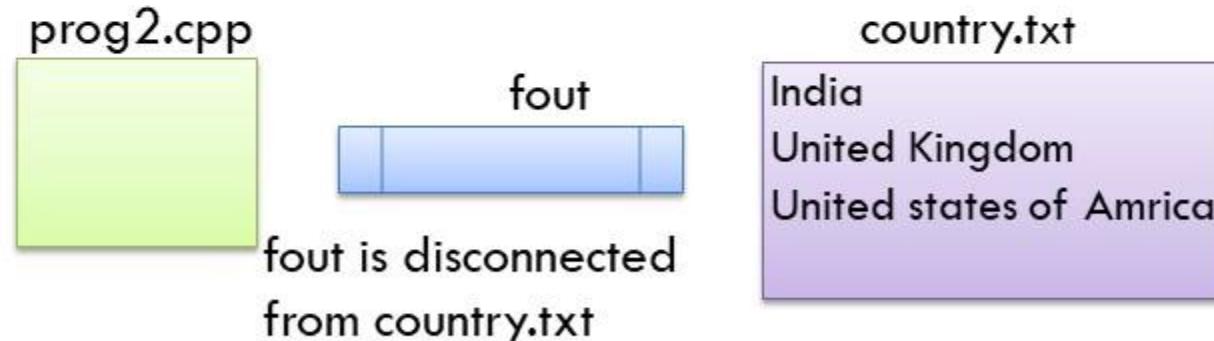
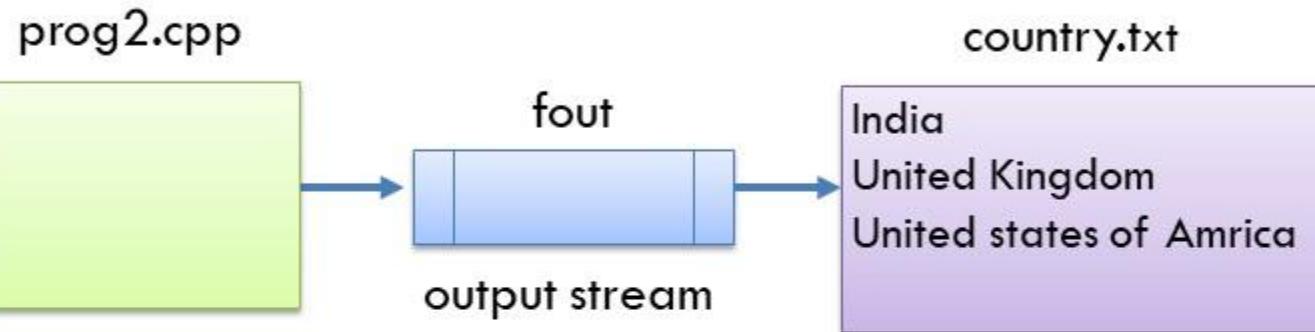
WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
#include<iostream>
#include<fstream>
using namespace std;
int main() {
ofstream fout;
fout.open("country.txt");
fout<<"India"<<endl;
fout<<"United kingdom"<<endl;
fout<<"United states of America"
<<endl;
```



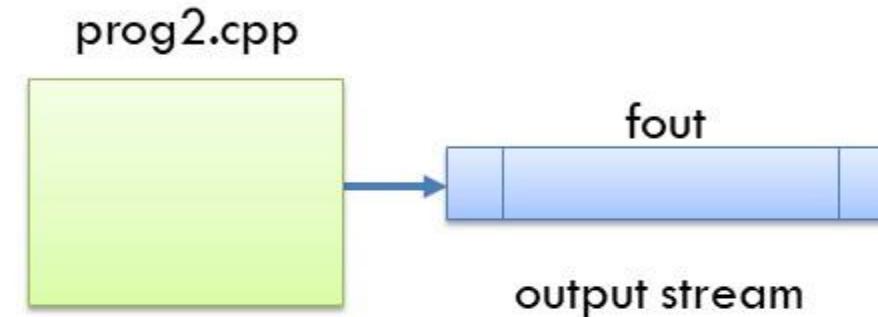
WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
#include<iostream>
#include<fstream>
using namespace std;
int main() {
ofstream fout;
fout.open("country.txt");
fout<<"India"<<endl;
fout<<"United kingdom"<<endl;
fout<<"United states of America"
<<endl;
fout.close();
```



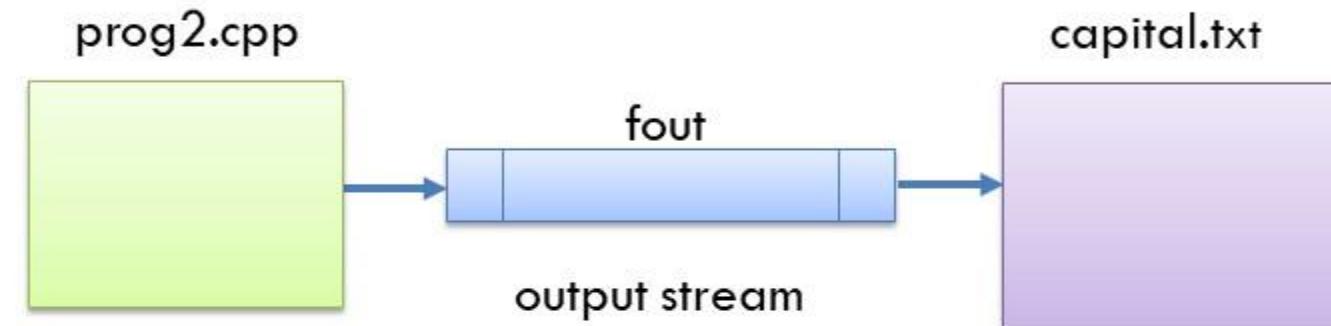
WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN



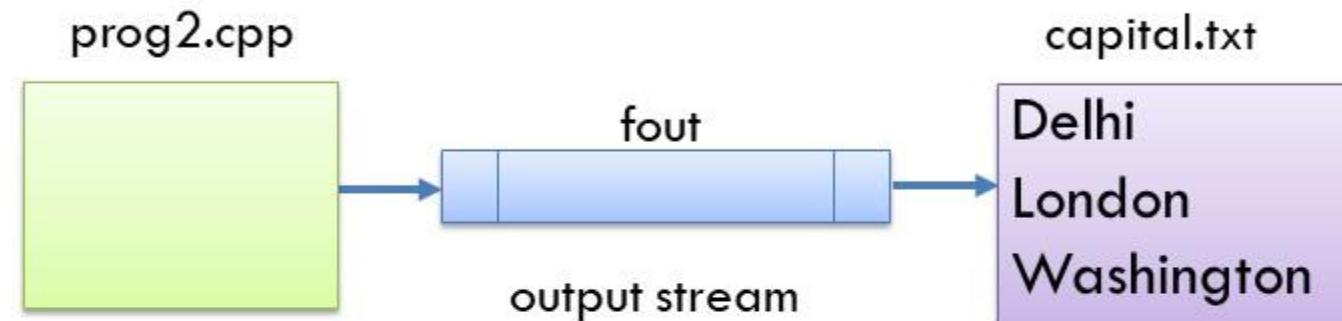
WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
fout.open("capital.txt");
```



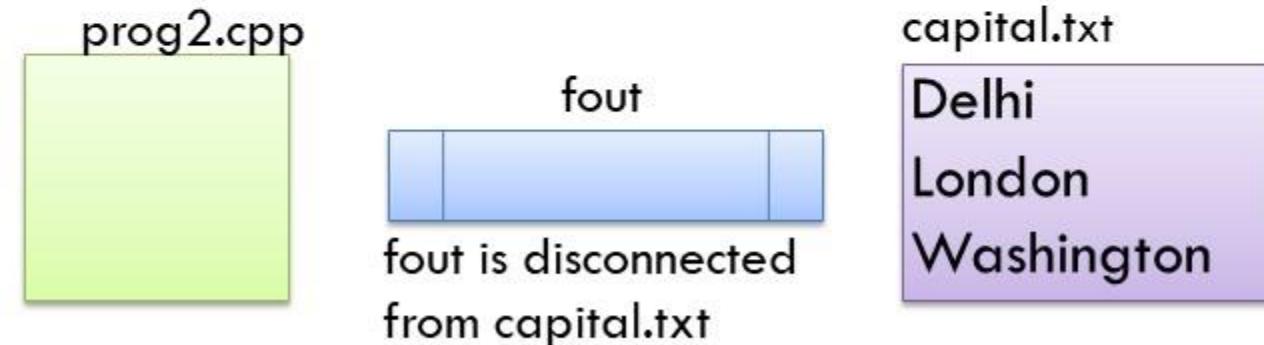
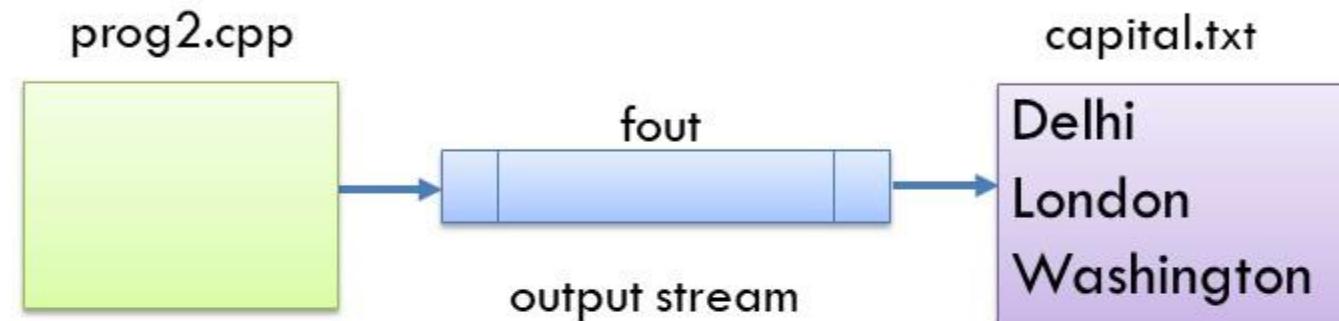
WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
fout.open("capital.txt");
fout<<"Delhi"<<endl;
fout<<"London"<<endl;
fout<<"Washington"<<endl;
```



WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
fout.open("capital.txt");
fout<<"Delhi"<<endl;
fout<<"London"<<endl;
fout<<"Washington"<<endl;
fout.close();
```



WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
char line[30];
```

Memory Allocation

line

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
char line[30];  
ifstream fin;
```

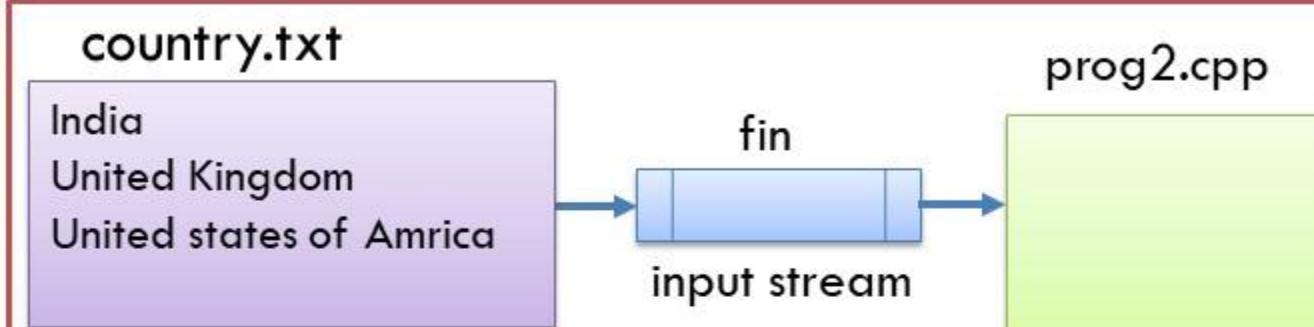


Memory Allocation

line

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
char line[30];  
ifstream fin;  
fin.open("country.txt");
```

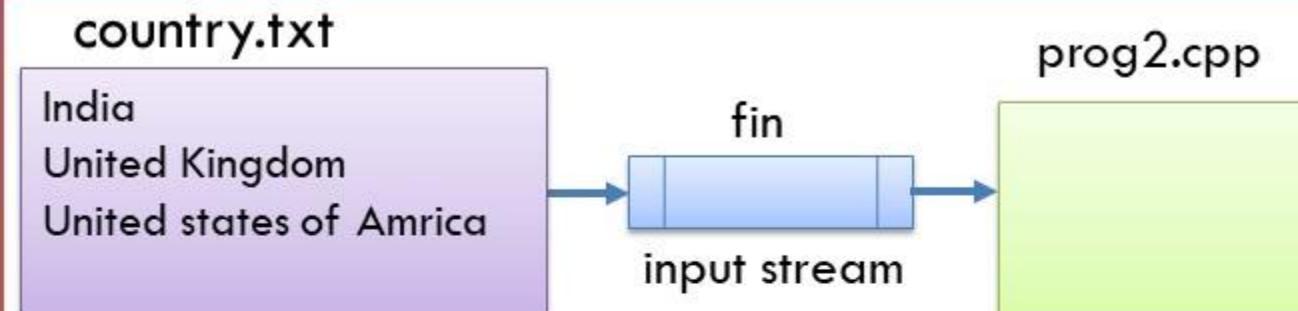


Memory Allocation

line

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
char line[30];  
  
ifstream fin;  
  
fin.open("country.txt");  
  
cout<<"Contents of  
country file"<<endl;
```



OUTPUT:

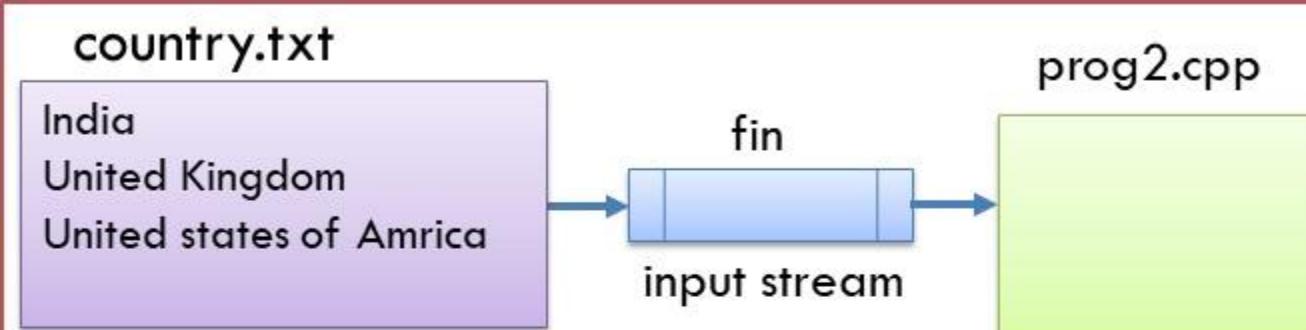
Contents of country file

Memory Allocation

line

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
char line[30];  
  
ifstream fin;  
  
fin.open("country.txt");  
  
cout<<"Contents of  
country file"<<endl;  
  
while(fin)  
{  
  
    fin.getline(line, 30);
```



OUTPUT:

Contents of country file

Memory Allocation

line India

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
char line[30];
ifstream fin;
fin.open("country.txt");
cout<<"Contents of
country file"<<endl;
while(fin)
{
    fin.getline(line, 30);
    cout<<line;
}
```

country.txt

India
United Kingdom
United states of Amrica



prog2.cpp

OUTPUT:

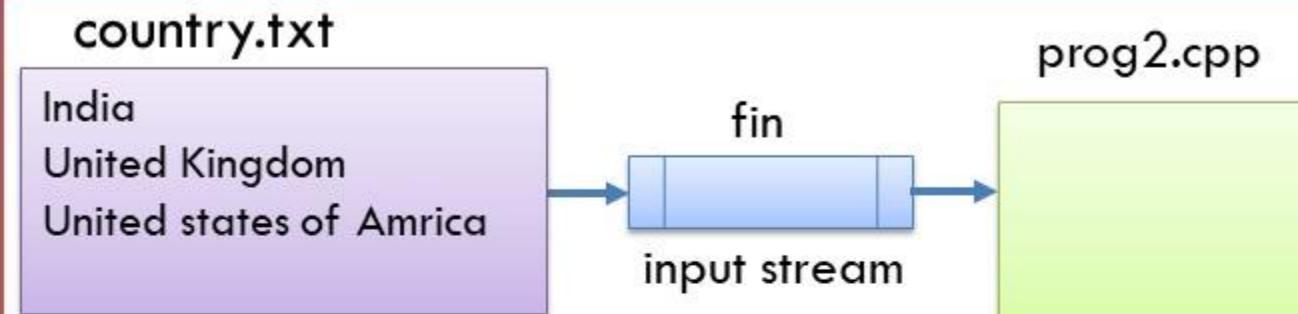
Contents of country file
India

Memory Allocation

line India

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
char line[30];  
  
ifstream fin;  
  
fin.open("country.txt");  
  
cout<<"Contents of  
country file"<<endl;
```



OUTPUT:

Contents of country file
India

Memory Allocation

line India

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
char line[30];  
  
ifstream fin;  
  
fin.open("country.txt");  
  
cout<<"Contents of  
country file"<<endl;  
  
while(fin)  
{
```

country.txt

India
United Kingdom
United states of Amrica



prog2.cpp

OUTPUT:

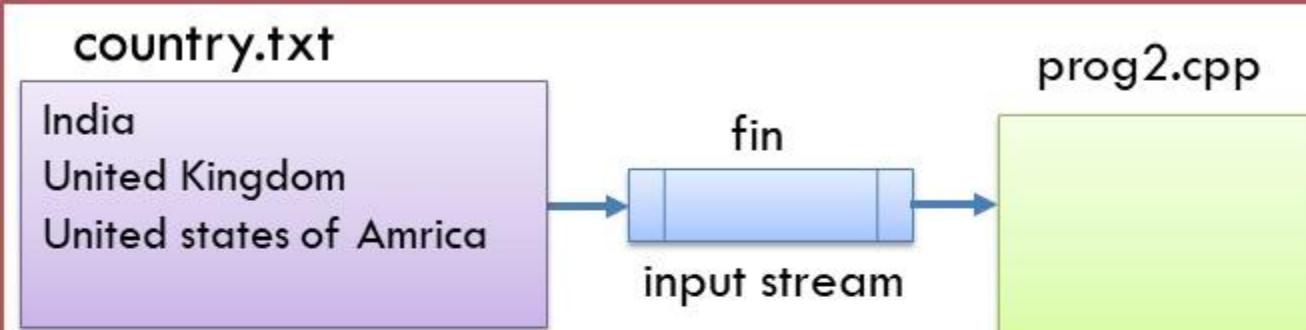
Contents of country file
India

Memory Allocation

line India

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
char line[30];  
  
ifstream fin;  
  
fin.open("country.txt");  
  
cout<<"Contents of  
country file"<<endl;  
  
while(fin)  
{  
  
    fin.getline(line, 30);  
}
```



OUTPUT:

Contents of country file
India

Memory Allocation

line United Kingdom

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
char line[30];
ifstream fin;
fin.open("country.txt");
cout<<"Contents of
country file"<<endl;
while(fin)
{
    fin.getline(line, 30);
    cout<<line;
}
```

country.txt

India
United Kingdom
United states of Amrica



prog2.cpp

OUTPUT:

Contents of country file
India
United Kingdom

Memory Allocation

line United Kingdom

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
char line[30];  
  
ifstream fin;  
  
fin.open("country.txt");  
  
cout<<"Contents of  
country file"<<endl;
```

country.txt

India
United Kingdom
United states of Amrica



prog2.cpp

OUTPUT:

Contents of country file
India
United Kingdom

Memory Allocation

line United Kingdom

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
char line[30];  
  
ifstream fin;  
  
fin.open("country.txt");  
  
cout<<"Contents of  
country file"<<endl;  
  
while(fin)  
{
```

country.txt

India
United Kingdom
United states of Amrica



prog2.cpp

OUTPUT:

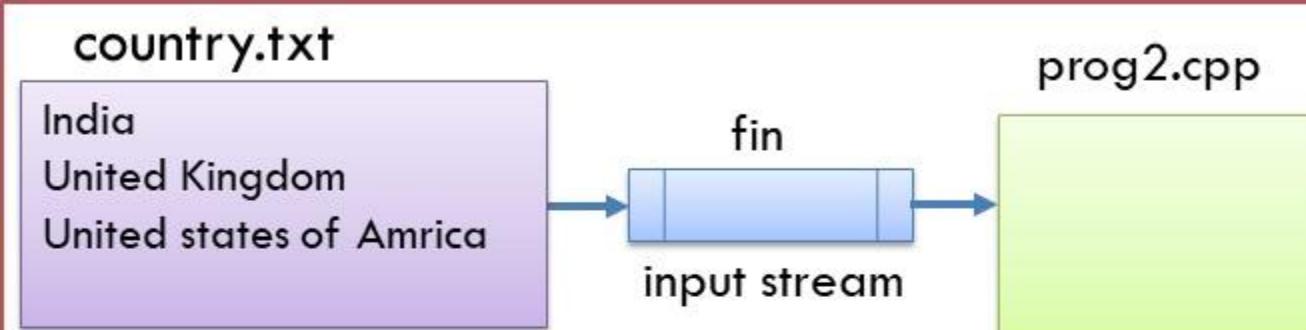
Contents of country file
India
United Kingdom

Memory Allocation

line United Kingdom

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
char line[30];  
  
ifstream fin;  
  
fin.open("country.txt");  
  
cout<<"Contents of  
country file"<<endl;  
  
while(fin)  
{  
  
    fin.getline(line, 30);  
}
```



OUTPUT:

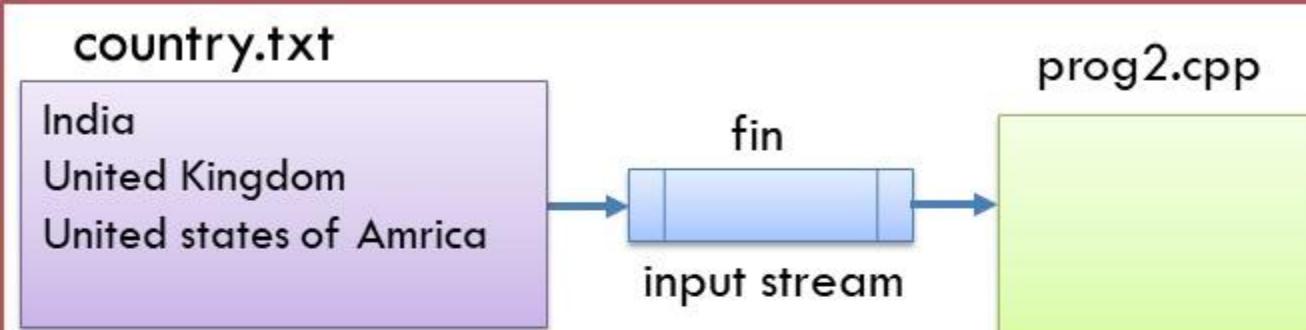
Contents of country file
India
United Kingdom

Memory Allocation

line

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
char line[30];  
  
ifstream fin;  
  
fin.open("country.txt");  
  
cout<<"Contents of  
country file"<<endl;  
  
while(fin)  
{  
  
    fin.getline(line, 30);
```



OUTPUT:

Contents of country file
India
United Kingdom

Memory Allocation

line United states of America

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
char line[30];
ifstream fin;
fin.open("country.txt");
cout<<"Contents of
country file"<<endl;
while(fin)
{
    fin.getline(line, 30);
    cout<<line;
}
```

country.txt

India
United Kingdom
United states of Amrica



prog2.cpp

OUTPUT:

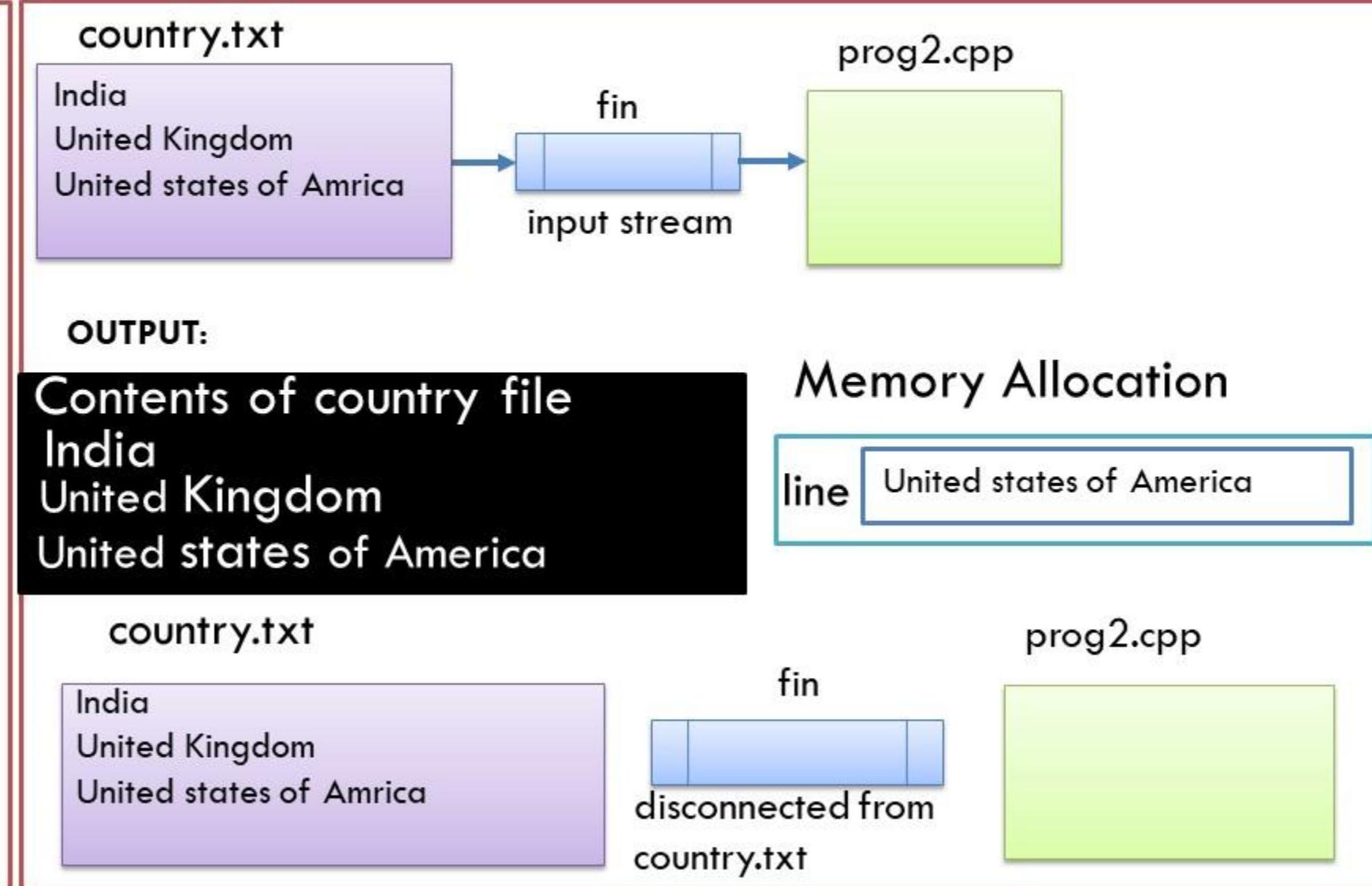
Contents of country file
India
United Kingdom
United states of America

Memory Allocation

line United states of America

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
char line[30];
ifstream fin;
fin.open("country.txt");
cout<<"Contents of
country file"<<endl;
while(fin)
{
    fin.getline(line, 30);
    cout<<line;
}
fin.close();
```



WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
/*char line[30];
```

Memory Allocation

line

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
/*char line[30];  
ifstream fin; */ comment
```

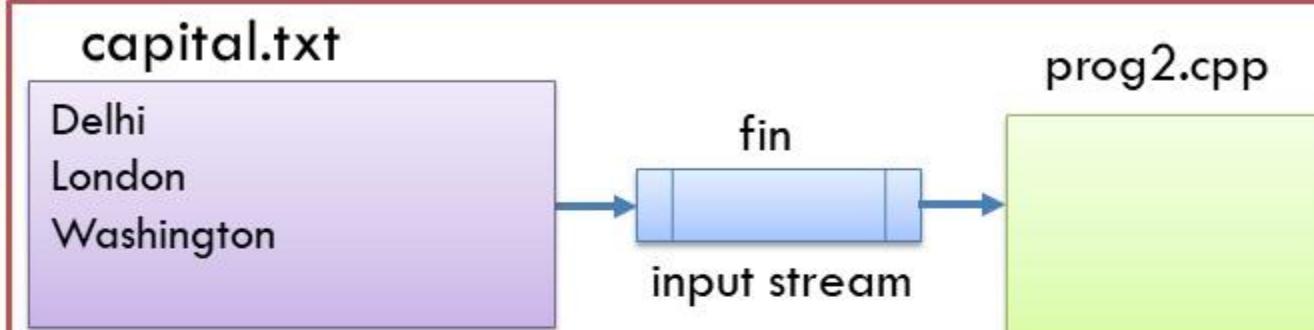


Memory Allocation

line

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
/*char line[30];  
ifstream fin; */ comment  
fin.open("capital.txt");
```

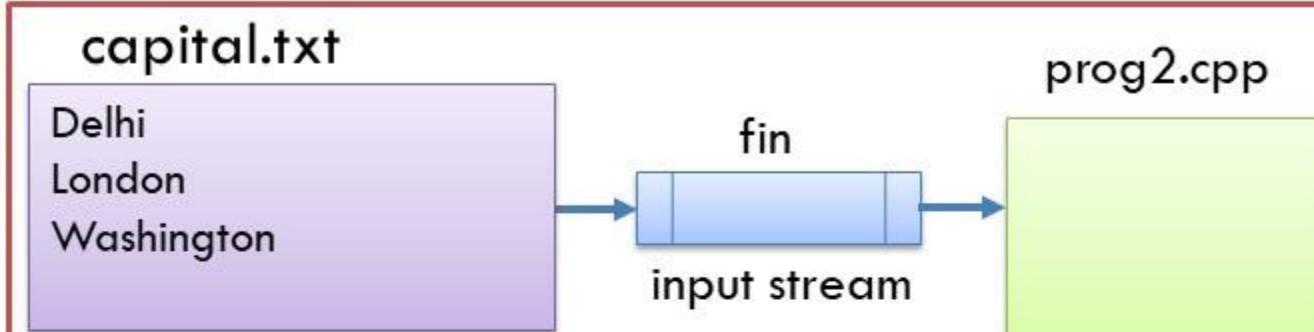


Memory Allocation

line

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
/*char line[30];  
ifstream fin; */ comment  
fin.open("capital.txt");  
cout<<"Contents of  
capital file"<<endl;
```



OUTPUT:

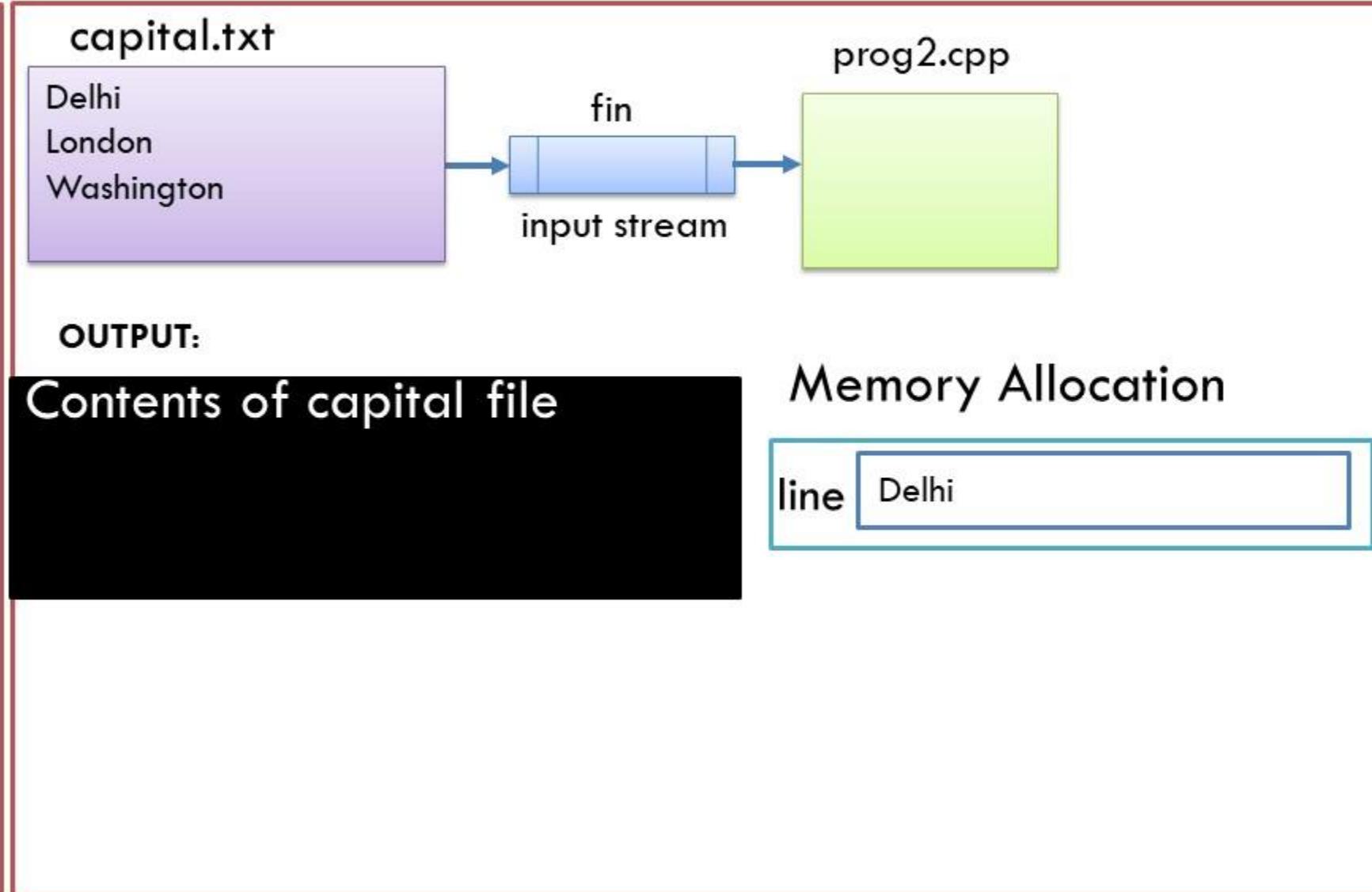
Contents of capital file

Memory Allocation

line

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
/*char line[30];  
ifstream fin; */ comment  
fin.open("capital.txt");  
cout<<"Contents of  
capital file"<<endl;  
while(fin)  
{  
    fin.getline(line, 30);  
}
```



WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
/*char line[30];  
ifstream fin; */ comment  
fin.open("capital.txt");  
cout<<"Contents of  
capital file"<<endl;  
while(fin)  
{  
    fin.getline(line, 30);  
    cout<<line;  
}
```

capital.txt

Delhi
London
Washington



prog2.cpp

OUTPUT:

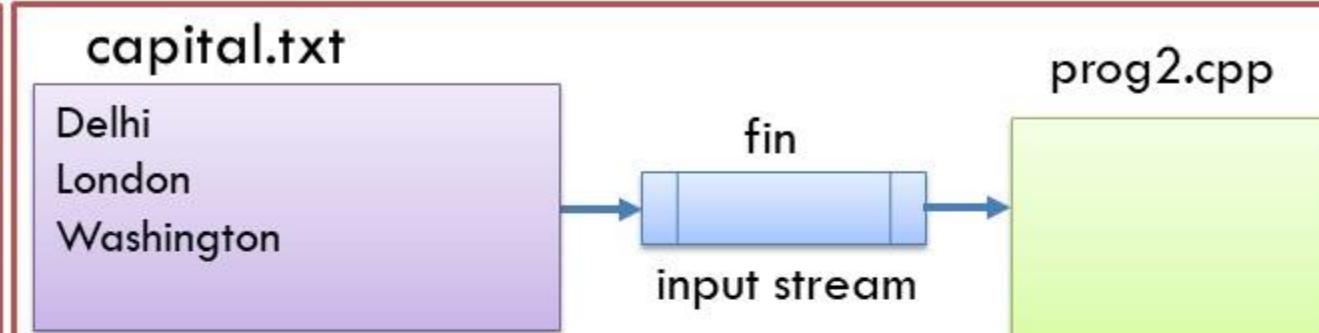
Contents of capital file
Delhi

Memory Allocation

line Delhi

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
/*char line[30];  
ifstream fin; */ comment  
fin.open("capital.txt");  
cout<<"Contents of  
capital file"<<endl;
```



OUTPUT:

Contents of capital file
Delhi

Memory Allocation

line Delhi

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
/*char line[30];  
ifstream fin; */ comment  
fin.open("capital.txt");  
cout<<"Contents of  
capital file"<<endl;  
while(fin)  
{
```

capital.txt

Delhi
London
Washington



prog2.cpp

OUTPUT:

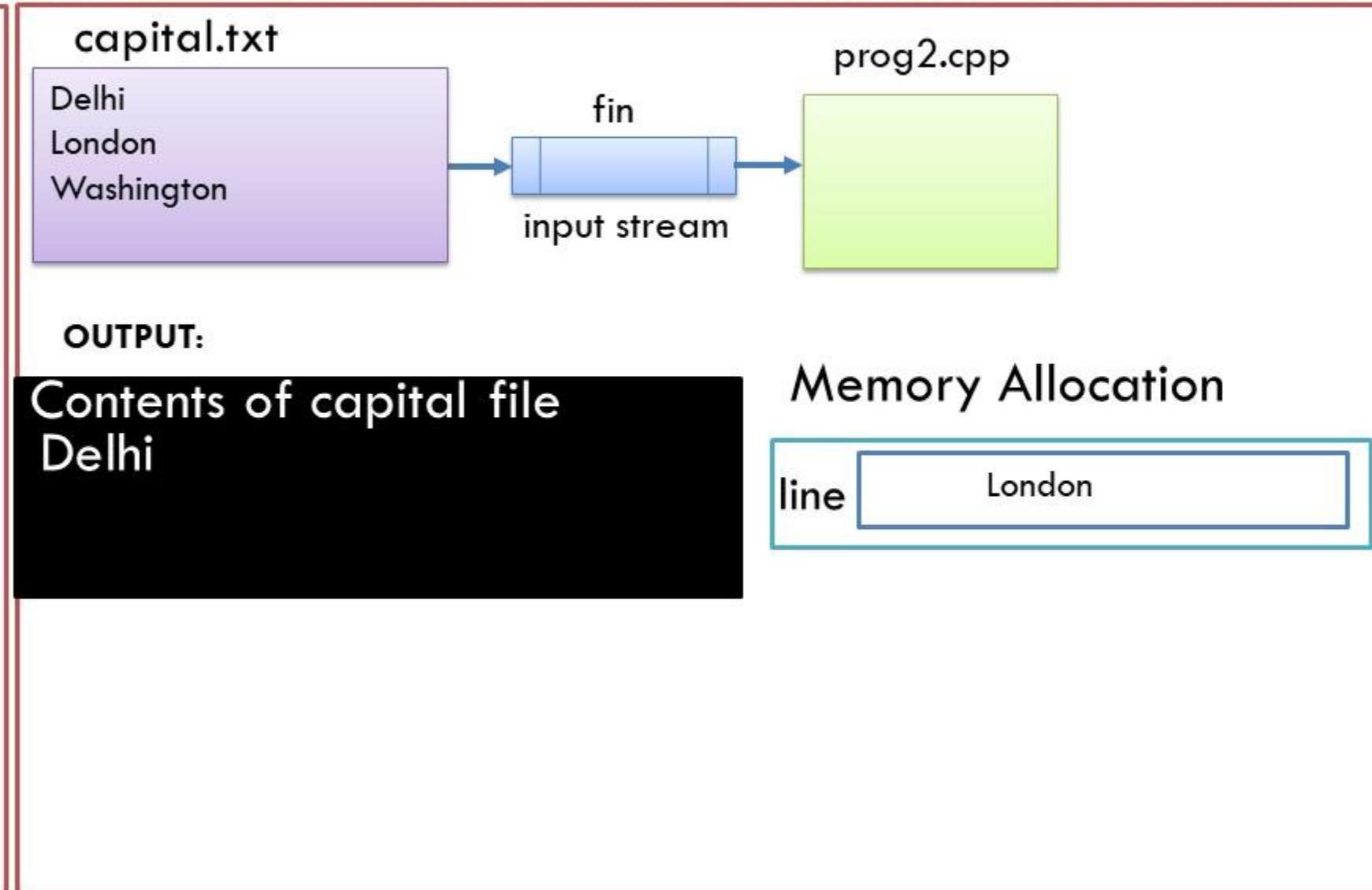
Contents of capital file
Delhi

Memory Allocation

line Delhi

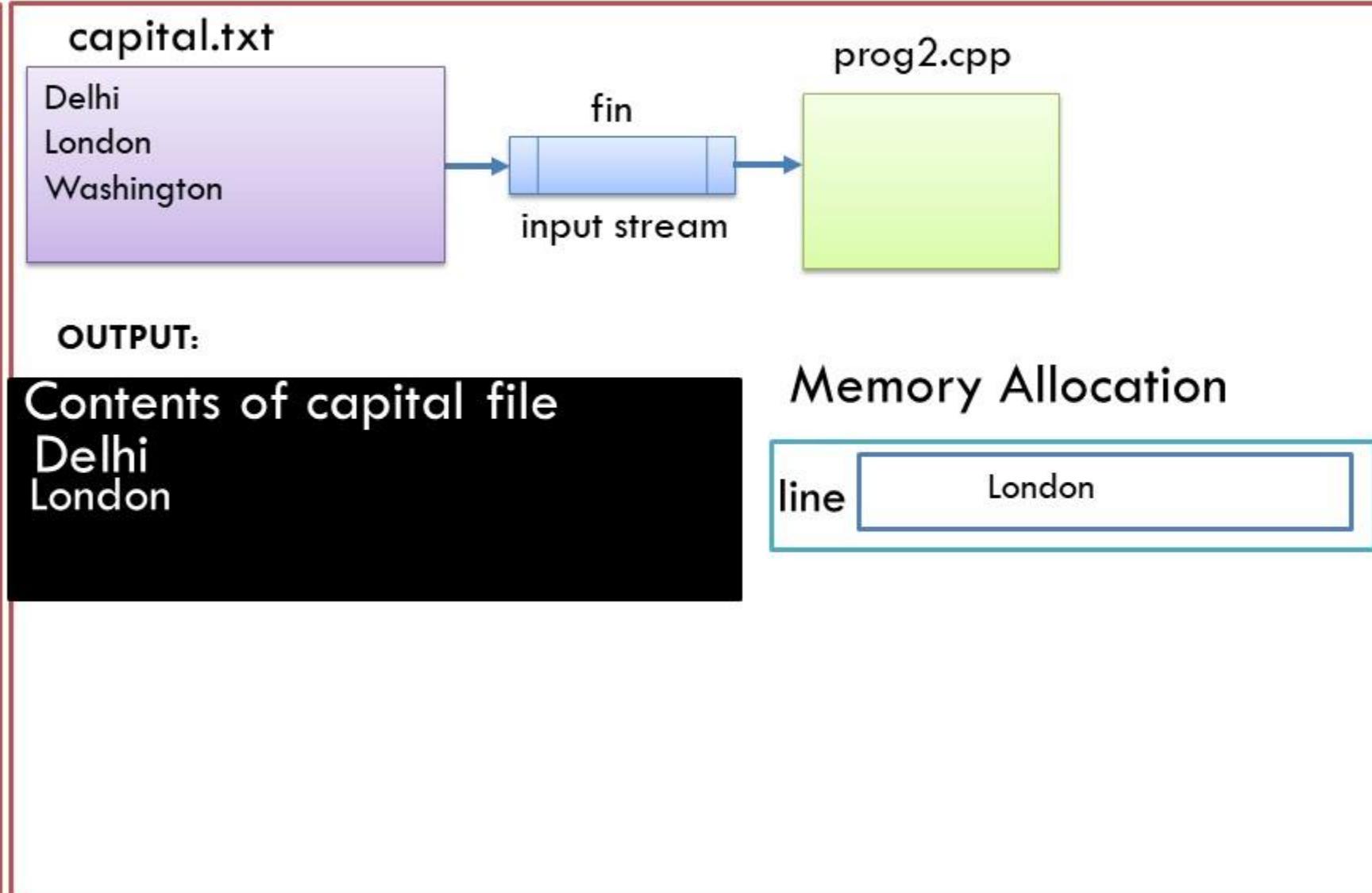
WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
/*char line[30];  
ifstream fin; */ comment  
fin.open("capital.txt");  
cout<<"Contents of  
capital file"<<endl;  
while(fin)  
{  
    fin.getline(line, 30);  
}
```



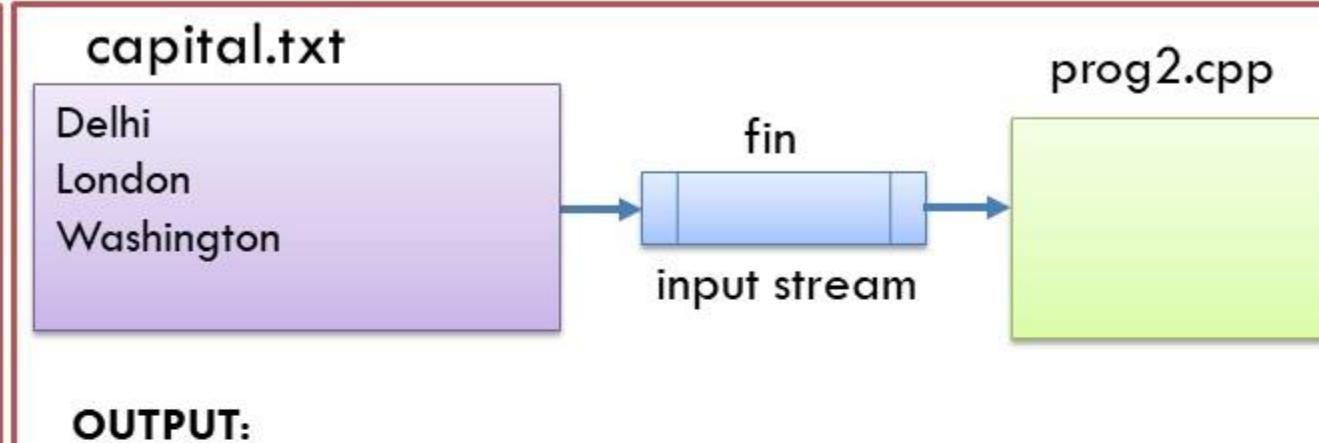
WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
/*char line[30];  
ifstream fin; */ comment  
fin.open("capital.txt");  
cout<<"Contents of  
capital file"<<endl;  
while(fin)  
{  
    fin.getline(line, 30);  
    cout<<line;  
}
```



WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
/*char line[30];  
ifstream fin; */ comment  
fin.open("capital.txt");  
cout<<"Contents of  
capital file"<<endl;
```



OUTPUT:

Contents of capital file
Delhi
London

Memory Allocation

line London

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
/*char line[30];  
ifstream fin; */ comment  
fin.open("capital.txt");  
cout<<"Contents of  
capital file"<<endl;  
while(fin)  
{
```

capital.txt

Delhi
London
Washington



prog2.cpp

OUTPUT:

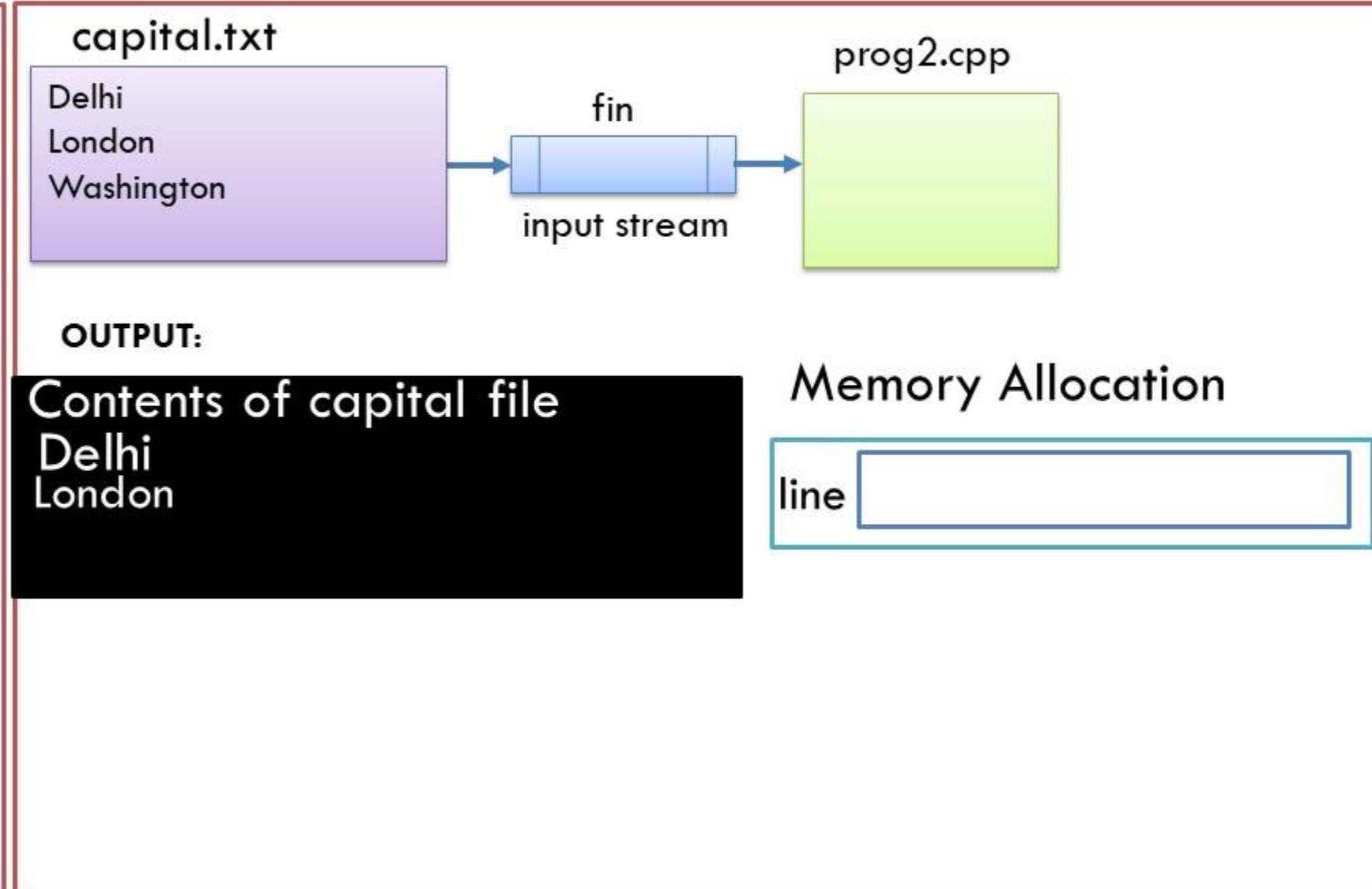
Contents of capital file
Delhi
London

Memory Allocation

line London

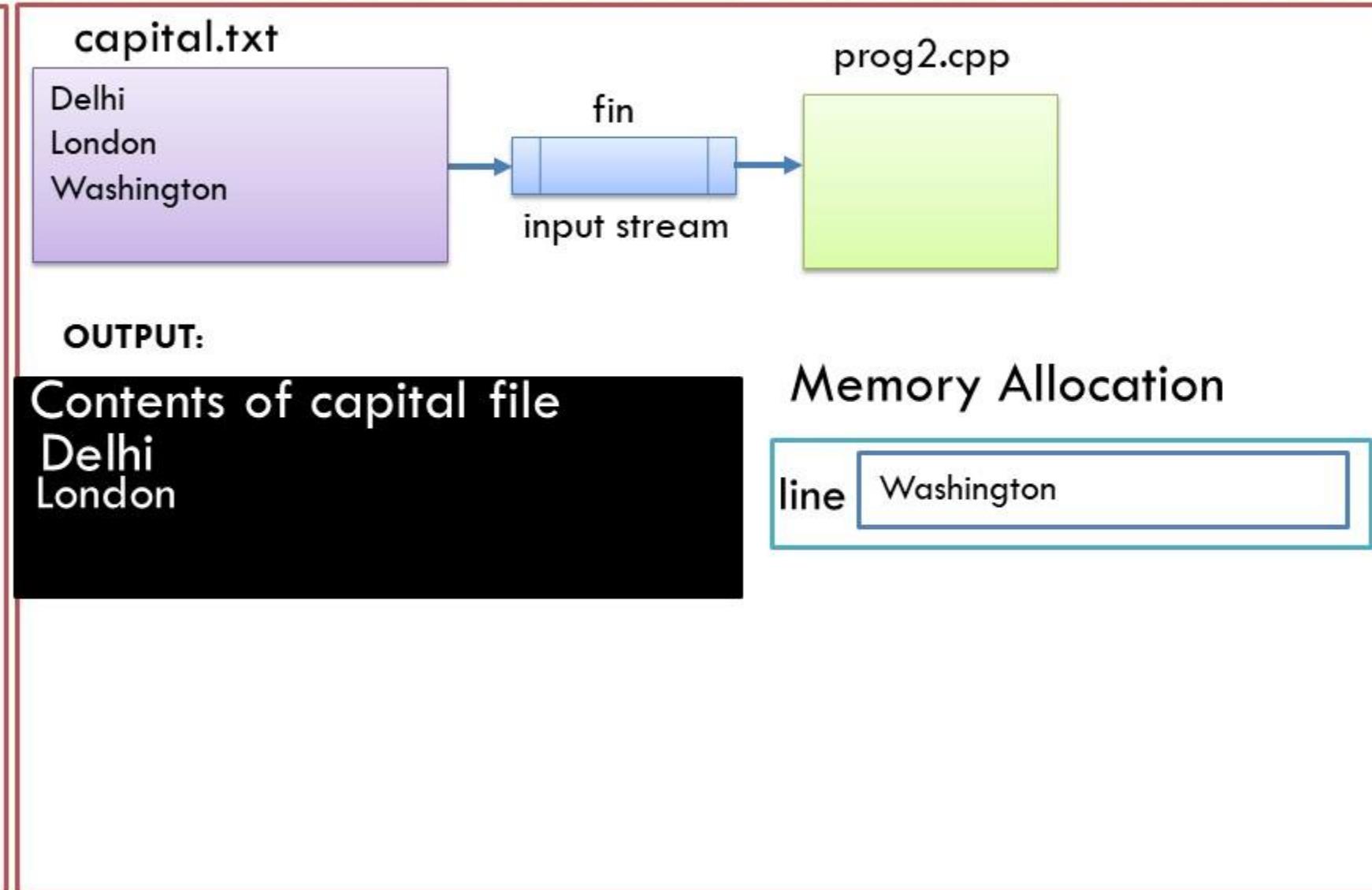
WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
/*char line[30];  
ifstream fin; */ comment  
fin.open("capital.txt");  
cout<<"Contents of  
capital file"<<endl;  
while(fin)  
{  
    fin.getline(line, 30);
```



WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
/*char line[30];  
ifstream fin; */ comment  
fin.open("capital.txt");  
cout<<"Contents of  
capital file"<<endl;  
while(fin)  
{  
    fin.getline(line, 30);  
}
```



WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

```
/*char line[30];  
ifstream fin; */ comment  
fin.open("capital.txt");  
cout<<"Contents of  
capital file"<<endl;  
while(fin)  
{  
    fin.getline(line, 30);  
    cout<<line;  
}
```

capital.txt

Delhi
London
Washington



prog2.cpp

OUTPUT:

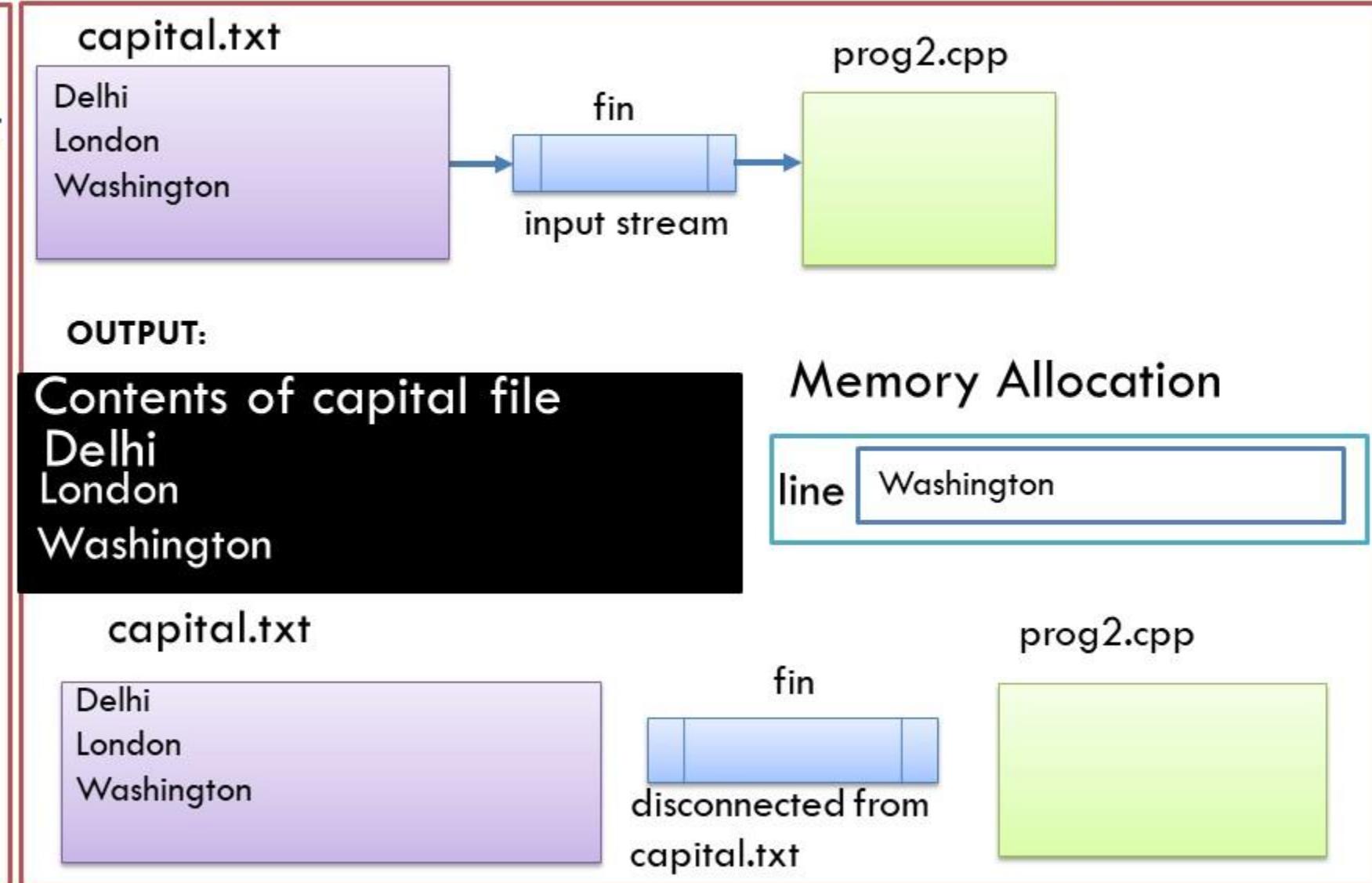
Contents of capital file
Delhi
London
Washington

Memory Allocation

line Washington

WORKING WITH MULTIPLE FILE: CREATING FILES WITH OPEN

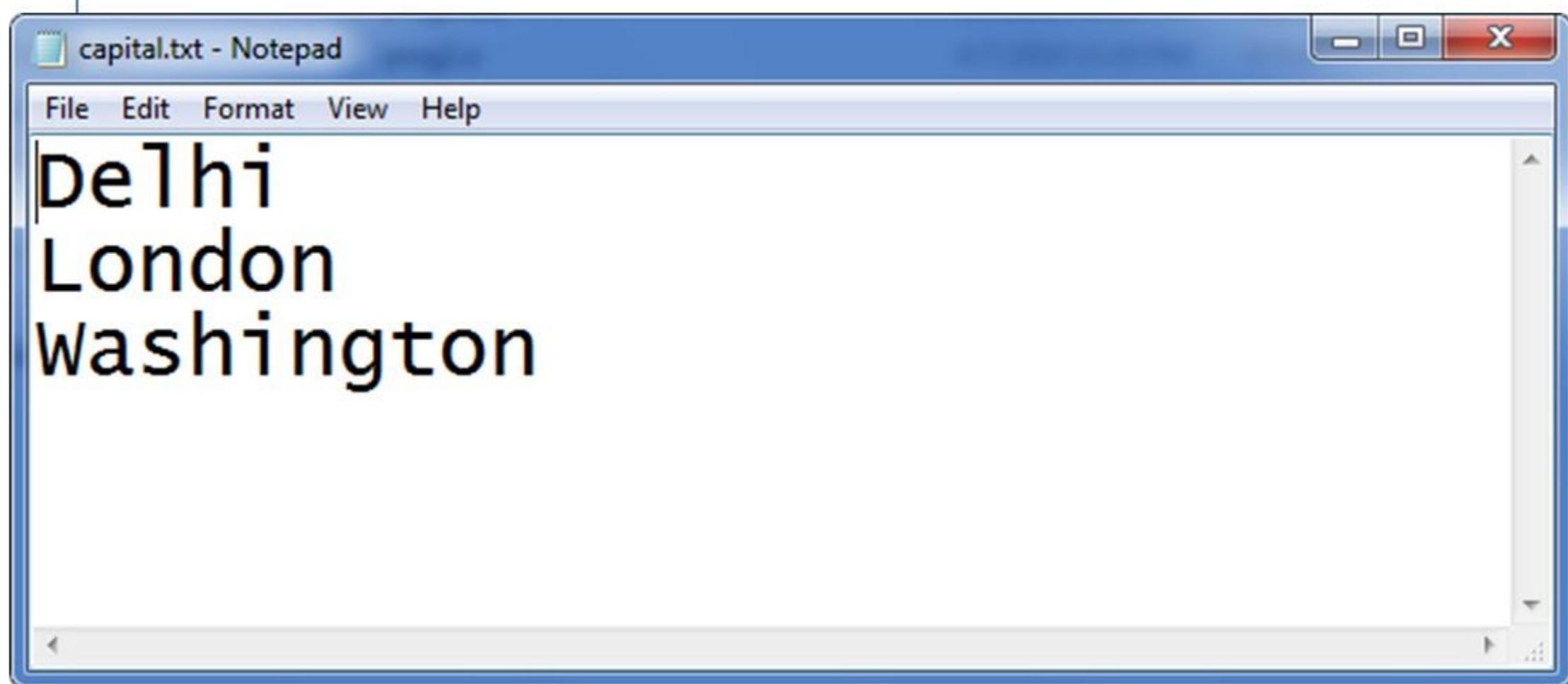
```
/*char line[30];  
ifstream fin; */ comment  
fin.open("capital.txt");  
cout<<"Contents of  
capital file"<<endl;  
while(fin)  
{  
    fin.getline(line, 30);  
    cout<<line;  
}  
fin.close();
```



India

United kingdom

United states of America



Contents of country file

India

United kingdom

United states of America

Contents of capital file

Delhi

London

Washington

READING FROM TWO FILES SIMULTANEOUSLY: PROG3

READING FROM TWO FILES SIMULTANEOUSLY: PROG3

Write a program

READING FROM TWO FILES SIMULTANEOUSLY: PROG3

Write a program

To open a file country.txt and capital.txt in read mode simultaneously

READING FROM TWO FILES SIMULTANEOUSLY: PROG3

Write a program

To open a file country.txt and capital.txt in read mode simultaneously

Read the contents from both the file at a time and display on the output screen.

READING FROM TWO FILES SIMULTANEOUSLY: PROG3

READING FROM TWO FILES SIMULTANEOUSLY: PROG3

```
#include<iostream>
#include<fstream>
#include<stdlib.h>
using namespace std;
int main()
{
```

READING FROM TWO FILES SIMULTANEOUSLY: PROG3

```
#include<iostream>
#include<fstream>
#include<stdlib.h>
using namespace std;

int main()
{
    char line[30];
}
```

Memory
Allocation

line

READING FROM TWO FILES SIMULTANEOUSLY: PROG3

```
#include<iostream>
#include<fstream>
#include<stdlib.h>
using namespace std;

int main()
{
    char line[30];
    ifstream fin1,fin2;
```

Memory Allocation

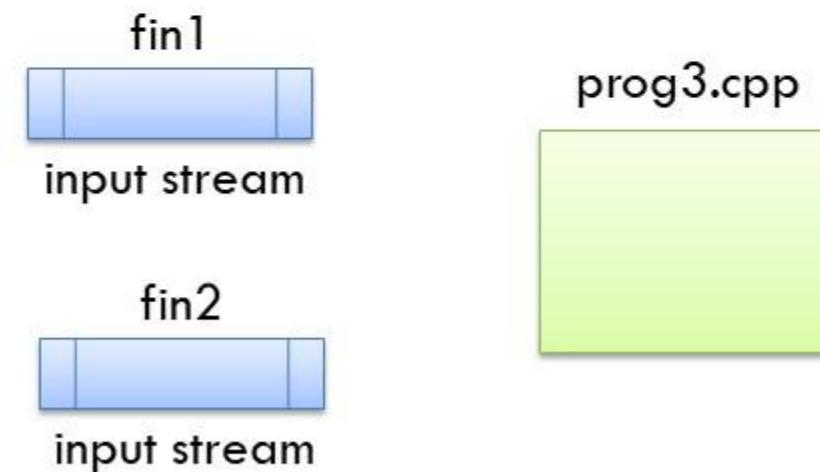


READING FROM TWO FILES SIMULTANEOUSLY: PROG3

```
#include<iostream>
#include<fstream>
#include<stdlib.h>
using namespace std;

int main()
{
    char line[30];
    ifstream fin1,fin2;
```

Memory Allocation

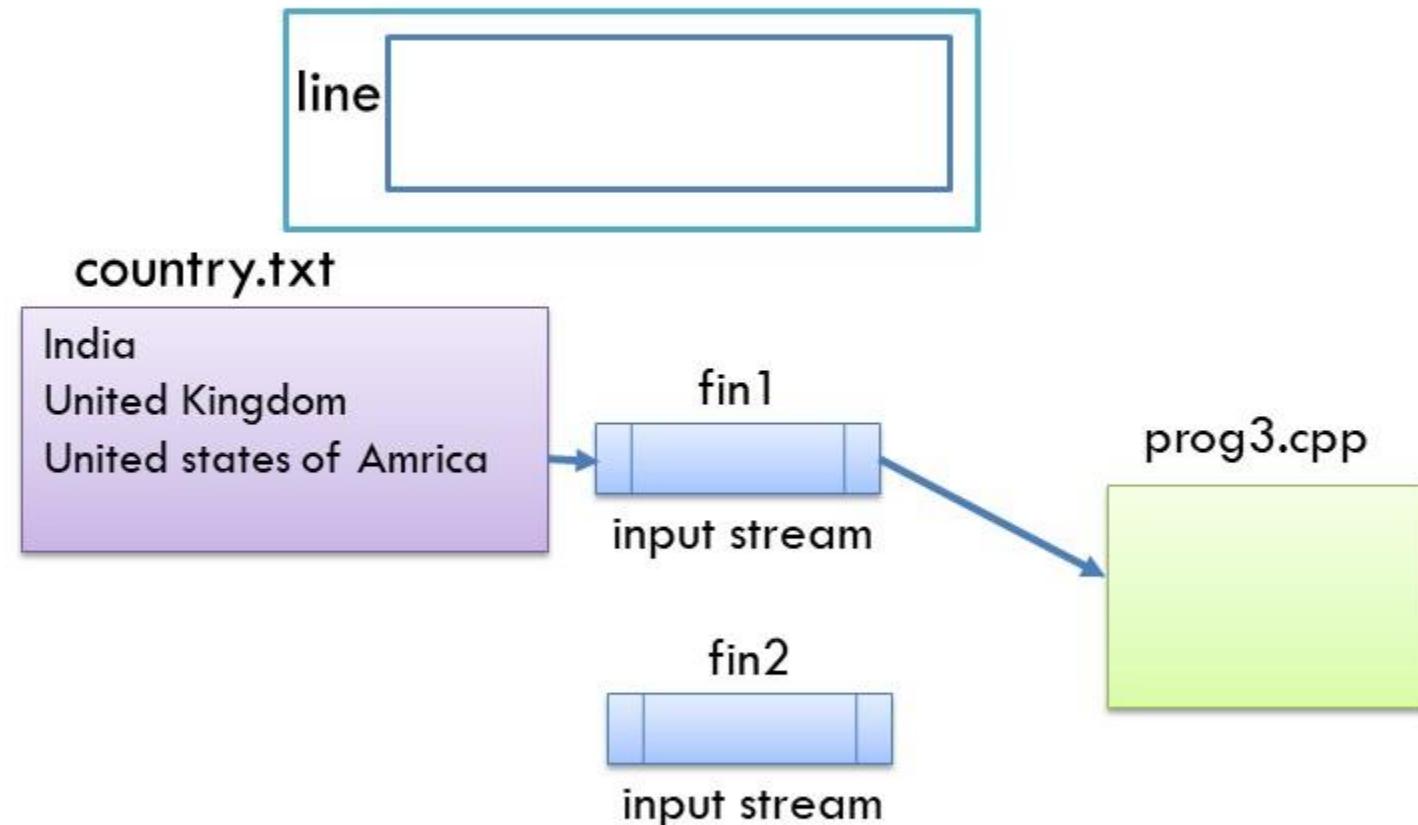


READING FROM TWO FILES SIMULTANEOUSLY: PROG3

```
#include<iostream>
#include<fstream>
#include<stdlib.h>
using namespace std;

int main()
{
    char line[30];
    ifstream fin1,fin2;
    fin1.open("country.txt");
```

Memory Allocation

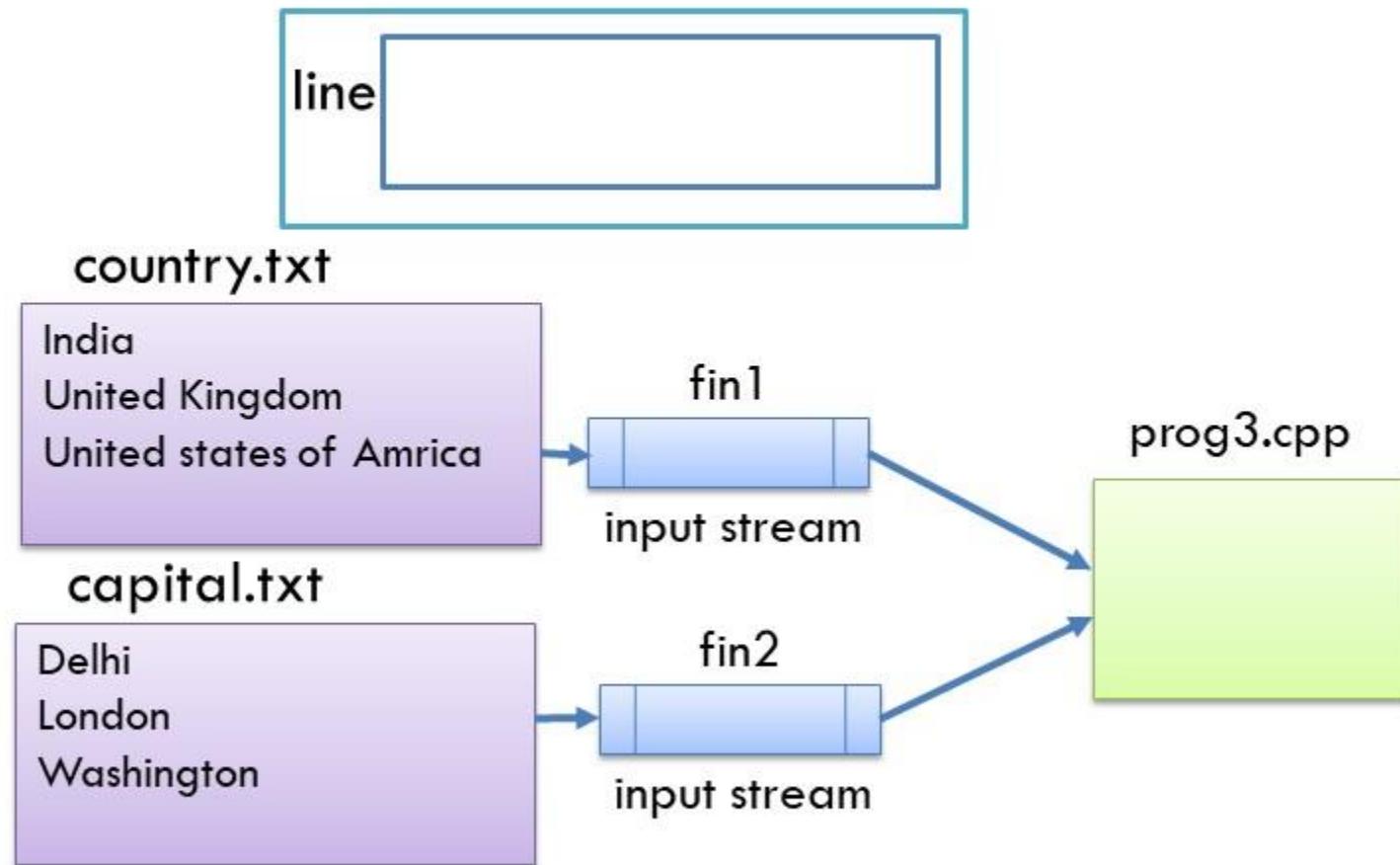


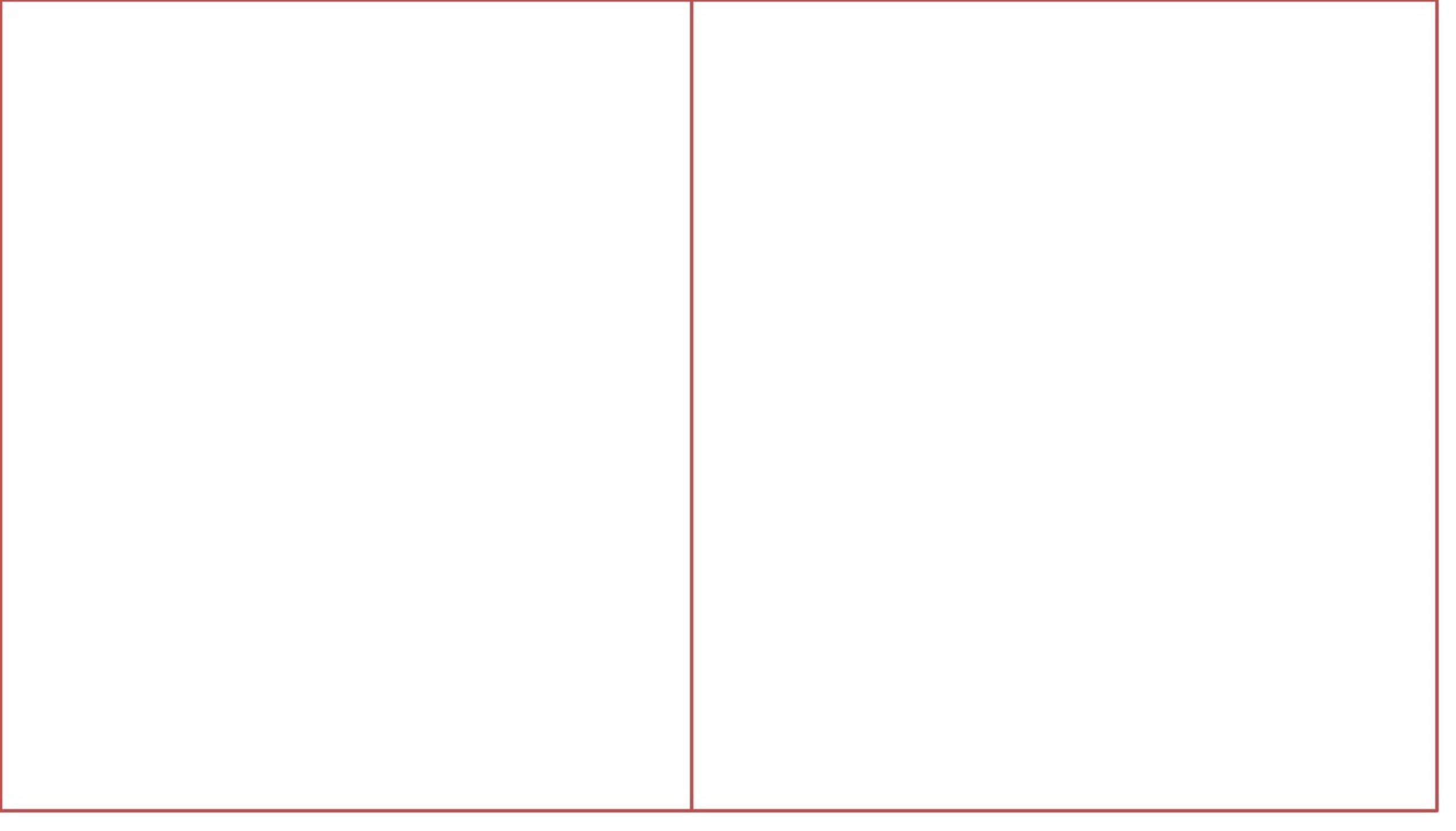
READING FROM TWO FILES SIMULTANEOUSLY: PROG3

```
#include<iostream>
#include<fstream>
#include<stdlib.h>
using namespace std;

int main()
{
    char line[30];
    ifstream fin1,fin2;
    fin1.open("country.txt");
    fin2.open("capital.txt");
```

Memory Allocation





```
for (int i=1; i<=3; i++)
```

```
for (int i=1; i<=3; i++)  
{
```

```
for (int i=1; i<=3; i++)  
{  
    if(fin1.eof()==1)
```

```
for (int i=1; i<=3; i++)
{
    if(fin1.eof()==1)
    {
```

```
for (int i=1; i<=3; i++)
{
    if(fin1.eof()==1)
    {
        cout<<"Exit from
country"<<endl;
```

```
for (int i=1; i<=3; i++)
{
    if(fin1.eof()==1)
    {
        cout<<"Exit from
country"<<endl;
        exit(1);
    }
}
```

```
for (int i=1; i<=3; i++)
{
    if(fin1.eof()==1)
    {
        cout<<"Exit from
country"<<endl;
        exit(1);
    }
}
```

```
for (int i=1; i<=3; i++)
{
    if(fin1.eof()==1)
    {
        cout<<"Exit from
country"<<endl;
        exit(1);
    }
    fin1.getline(line, 30);
```

```
for (int i=1; i<=3; i++)
{
    if(fin1.eof()==1)
    {
        cout<<"Exit from
country"<<endl;
        exit(1);
    }
    fin1.getline(line, 30);
    cout<<"Capital of
"<<line<<endl;
```

```
for (int i=1; i<=3; i++)
{
    if(fin1.eof()==1)
    {
        cout<<"Exit from
country"<<endl;
        exit(1);
    }
    fin1.getline(line, 30);
    cout<<"Capital of
"<<line<<endl;
}

if(fin2.eof()==1)
{
    cout<<"Exit from
capital"<<endl;
    exit(1);
}
fin2.getline(line, 30);
cout<<line<<endl;
}

return 0;
}
```

Capital of India

Delhi

Capital of United kingdom

London

Capital of United states of America

Washington

READING FROM TWO FILES
SIMULTANEOUSLY: PROG3

OUTPUT

Thank
you!!