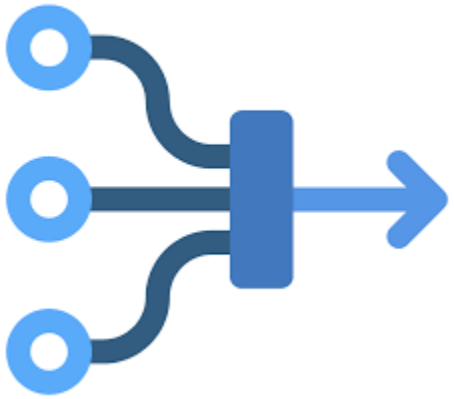
I/O and File Management

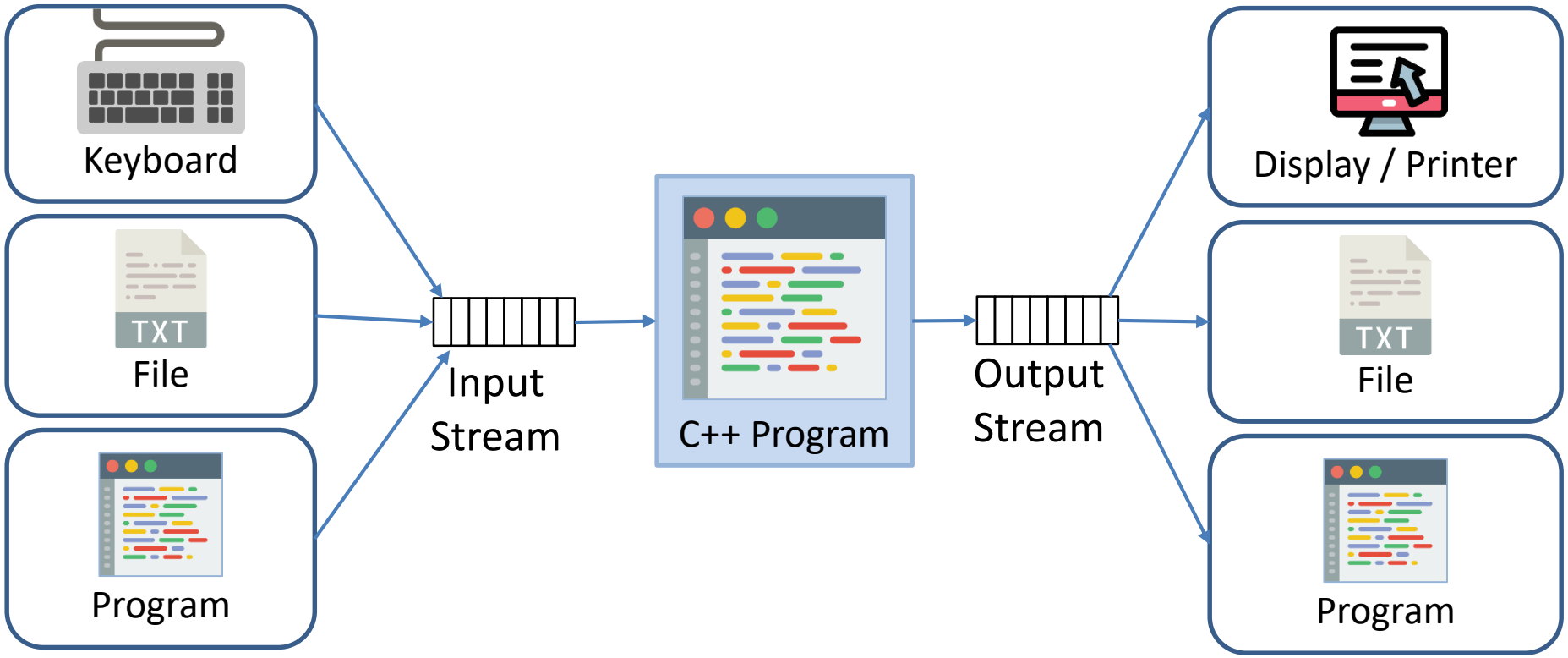
I/O and File Management

- Concept of streams
- cin and cout objects
- C++ stream classes
- Unformatted and formatted I/O
- Manipulators
- File stream
- C++ File stream classes
- File management functions
- File modes
- Binary and random Files

Concepts of Streams



Concept of Streams



Input source to stream

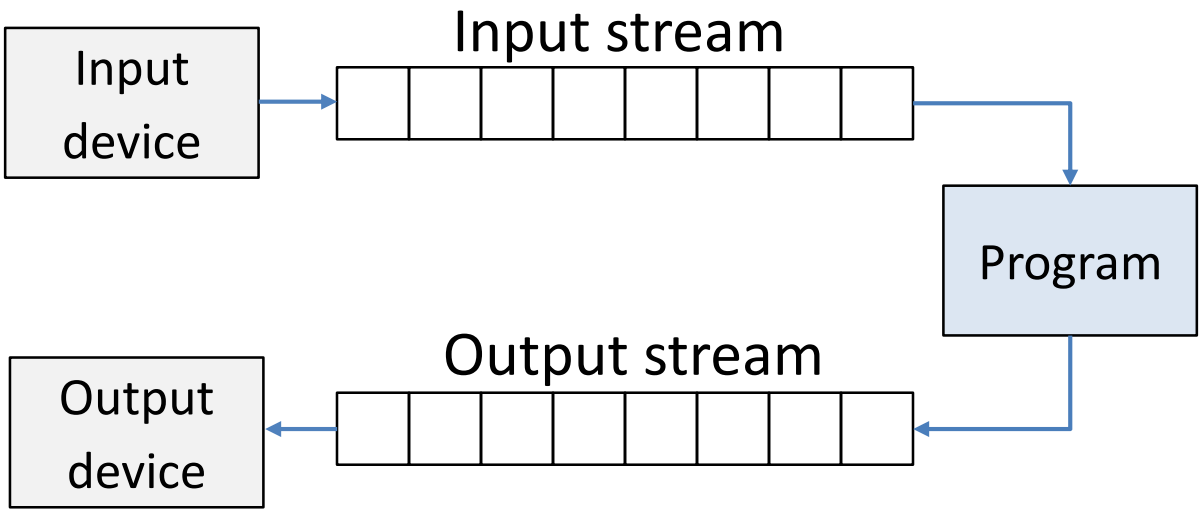
Output target from stream

Concept of streams(Cont…)

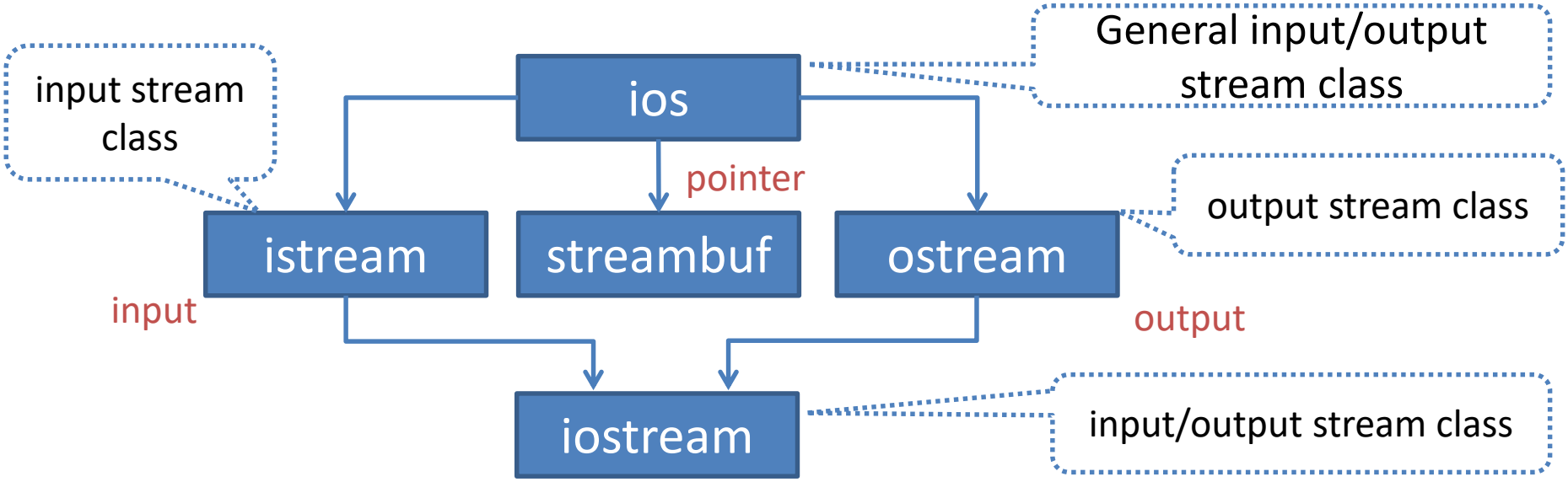
- A **stream** is a general name given to a flow of data.
- A **stream** is a sequence of bytes.
- The source stream that provides data to programs is called **input stream**.
- The destination stream receives output from the program is called **output stream**.

- In header **<iostream>**, a set of class is defined that supports I/O operations.
- The classes used for input/output to the **devices** are declared in the **IOSTREAM** file.
- The classes used for **disk file** are declared in the **FSTREAM** file.

Input/Output streams



Stream class for console I/O operations



- **ios** class contains basic facilities that are used by all other input and
- **ostream** class inherits properties of **ios**
- Insertion operator <<, put() and write() are members of **ostream** class

Unformatted and Formatted I/O

put(), get(), getline(), write() - Unformatted I/O Operations

```
char ch;
cin.get(ch);
ch=cin.get();
cin>>ch;
cout.put(ch);
cout.put('x');

char name[20];
cin.getline(name,10);

cin>>name;

cout.write(name,10);
```

Get a character from keyboard
Similar to cin.get(ch);

The operator >> can also be used to read a character but it will skip the white spaces and newline character.

put() function can be used to display value of variable ch or character.

line size

getline() reads whole line of text that ends with newline character or upto (size-1).

cin can read strings that do not contain white spaces

write() displays string of given size, if the size is greater than the length of line, then it displays the bounds of line.

ios Format Functions

Function	Task
width()	To specify the required field size for displaying an output value
precision()	To specify number of digits to be displayed after the decimal point of a float value.
fill()	To specify a character that is used to fill the unused portion of a field.
setf()	To specify format flags that can control the form of output.
unsetf()	To clear the flags specified

Example:

```
cout.setf(ios::left,ios::adjustfield);
cout.width(6);
cout.fill('#');
cout<<"543";
```

output:

5	4	3	#	#	#

Flags and bit fields

Format required	Flag (arg1)	Bit-field (arg2)
Left justified output	ios::left	ios::adjustfield
Right justified output	ios::right	ios::adjustfield
Scientific notation	ios::scientific	ios::floatfield
Fixed point notation	ios::fixed	ios::floatfield
Decimal base	ios::dec	ios::basefield
Octal base	ios::oct	ios::basefield
Hexadecimal base	ios::hex	ios::basefield

`setf(arg1, arg2)`

arg-1: one of the formatting flags.

arg-2: bit field specifies the group to which the formatting flag belongs.

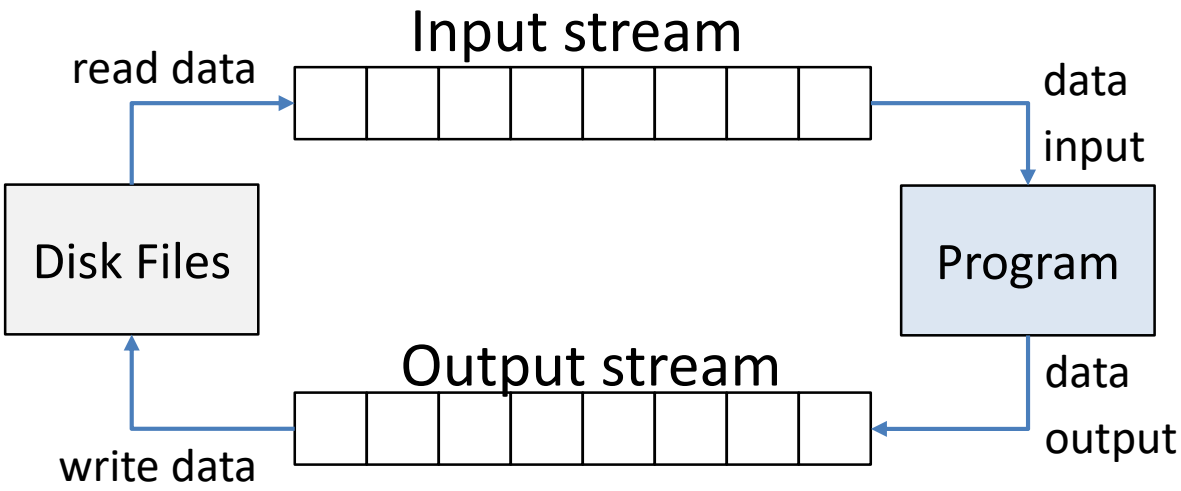
Manipulators for formatted I/O operations

- **Manipulators** are special functions that can be included in the I/O statements to alter the format parameters of a stream.
- To access manipulators, the file `<iomanip>` should be included in the program.

Function	Manipulator	Meaning
width()	**setw()**	Set the field width.
precision()	**setprecision()**	Set the floating point precision.
fill()	**setfill()**	Set the fill character.
setf()	**setiosflags()**	Set the format flag.
unsetf()	**resetiosflags()**	Clear the flag specified.
"\n"	**endl**	Insert a new line and flush stream.

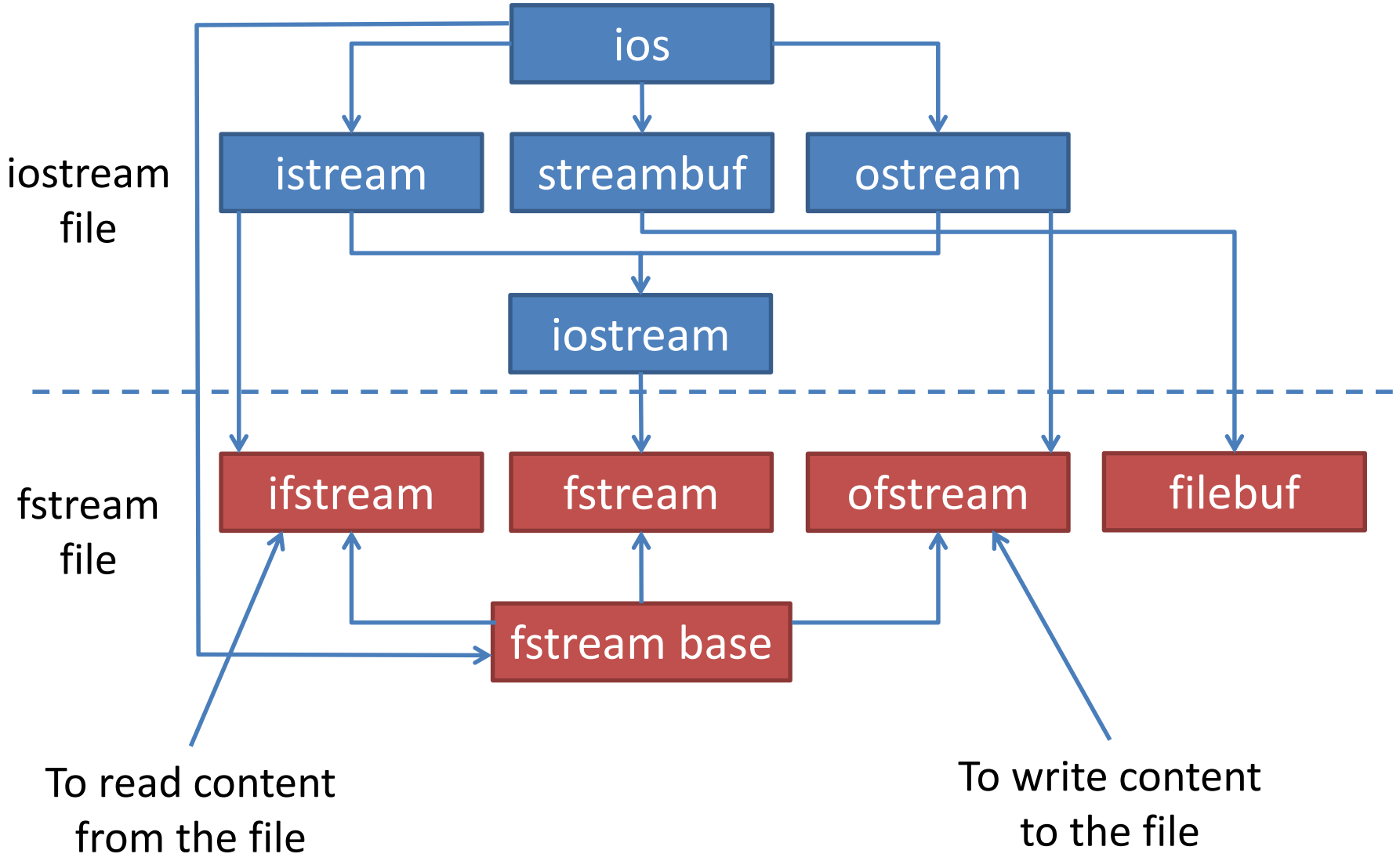
File stream classes

File input output streams



File input output streams

File stream classes for file operations



File stream classes

class	contents
fstreambase	▪ Provides operations common to the file streams. ▪ Contains **open()** and **close()** functions.
ifstream	▪ Provides input operations. ▪ Contains **open()** with default input mode. ▪ Inherits **get(), getline(), read(), seekg()** and **tellg()** functions from istream.
ofstream	▪ Provides output operations. ▪ Contains **open()** with default output mode. ▪ Inherits **put(), seekp(), tellp()** and **write()** functions from **ostream**.
fstream	▪ Provides support for simultaneous input and output operations. ▪ Inherits all the functions from istream and ostream from iostream.
filebuf	▪ Its purpose is to set the file buffers to read and write.

File handling steps

1. Open / Create a file
2. Read / Write a file
3. Close file

Create and Write File (Output)

Create object of **ofstream** class

```
ofstream send;
```

Call **open()** function using **ofstream** object to open a file

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

This will open existing file, if not exist then it will create file.

Write content in file using **ofstream** object

```
send<<"Hello, this is India";
```

Call **close()** function using **ofstream** object to close file

```
send.close();
```

Open and Read File (Input)

Create object of **ifstream** class

```
ifstream rcv;
```

Call **open()** function using **ifstream** object to open a file

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

Read content of file using **ifstream** object

```
rcv>>name;
```

```
rcv.getline(name);
```

Call **close()** function using **ifstream** object to close file

```
rcv.close();
```

Opening a file

```
ofstream outFile("sample.txt");  //output only
ifstream inFile("sample.txt");  //input only

ofstream outFile;
outFile.open("sample.txt");
ifstream inFile;
inFile.open("sample.txt");
```

This creates **outFile** as an **ofstream** object that manages the output stream.

This object can be any valid C++ name such as myfile, o_file .

- Syntax file **open()** function:

```
stream-object.open("filename", mode);
```

- By default **ofstream** opens file for writing only and **ifstream** opens file for reading only.

File open() function

File open() function

Syntax:

```
stream-object.open("filename", mode);
```

- By default **ofstream** opens file for writing only
- By default **ifstream** opens file for reading only.

Three ways to create a file

1

```
ofstream send("abc.txt");  //constructor
```

2

```
ofstream send;
send.open("abc.txt");  //open() function
```

3

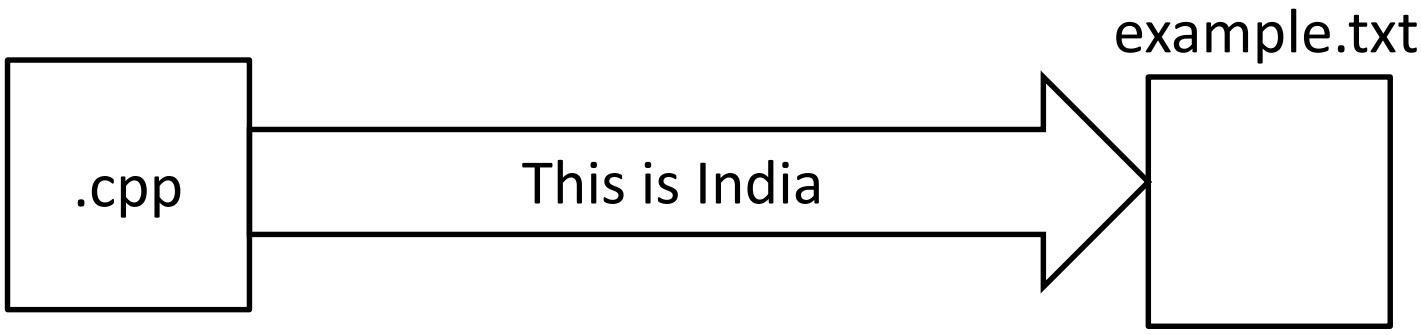
```
ofstream send;
send.open("abc.txt",ios::out); //open()
function with mode
```

File opening modes

Parameter	Meaning
ios :: in	Open file for reading only
ios :: out	Open file for writing only
ios :: app	Append to end-of-file
ios :: ate	Go to end-of-file on opening
ios :: binary	Binary file
ios :: trunc	Delete content of file if exists
ios :: nocreate	Open fails if the file does not exists
ios :: noreplace	Open fails if the file already exists

File operations

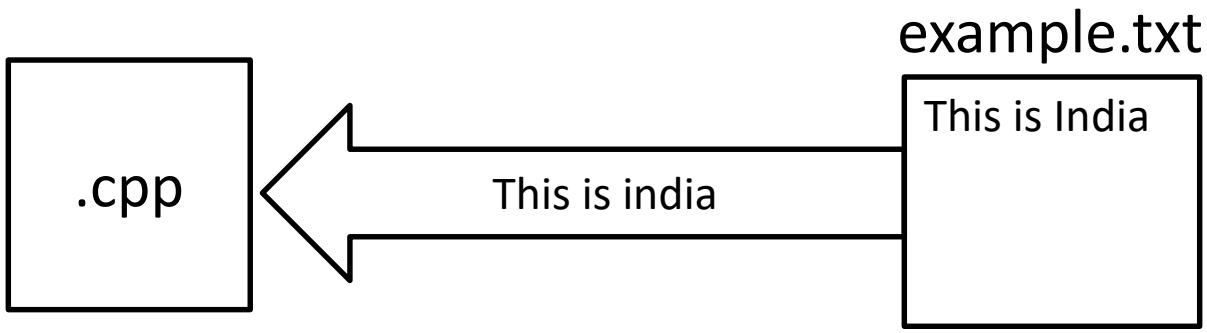
```
#include <iostream>
#include <fstream>
using namespace std;
int main ()
{
  ofstream myfile;
  myfile.open("example.txt",ios::out);
  myfile << "This is India.\n";
  myfile.close();
}
```



File operations (Cont..)

```
int main ()
{
   char line[50];
   ifstream rfile;
   rfile.open("example.txt",ios::in)
   rfile.getline(line,50);
   // rfile>>line  is also valid;

   cout<<line;
   rfile.close();
}
```



File operations program

```
int main()
{
    char product[20];
    int price;
    cout<<"Enter product name=";
    cin>>product;
    cout<<"Enter price=";
    cin>>price;

    ofstream outfile("stock.txt");
    outfile<<product<<endl;
    outfile<<price;

    ifstream infile("stock.txt");
    infile>>product;
    infile>>price;

    cout<<product<<endl;
    cout<<price;
}
```

Opening a file to write data into file

Opening a file to read data from file

File handling Program

- Write a program that opens **two text files** for reading data.
- It creates a **third file** that contains the text of first file and then that of second file

  (text of second file to be appended after text of the first file, to produce the third file).

```
int main() {
   fstream file1,file2,file3;
   file1.open("one.txt",ios::in);
   file2.open("two.txt",ios::in);
   file3.open("three.txt",ios::app);
   char ch1,ch2;
   while(!file1.eof())
   {
      file1.get(ch1); cout<<ch1<<endl;
      file3.put(ch1);
   }
   file1.close();
   while(!file2.eof())
   {
      file2.get(ch2); cout<<ch2<<endl;
      file3.put(ch2);
   }
   file2.close(); file3.close();
}
```

File pointers

- Each file has two associated pointers known as the **file pointers**.
- One of them is called **input pointer (or get pointer)** and the other is called **output pointer (or put pointer)**.
- **Input pointer** is used for reading the content of a given file location.
- **Output pointer** is used for writing to a given file location.

Functions for manipulation of file pointers

Function	Meaning
seekg()	Moves get pointer (input) to specified location
seekp()	Moves put pointer (output) to specified location
tellg()	Gives current position of the get pointer
tellp()	Gives current position of the put pointer

```
ifstream rcv;
ofstream send;

rcv.seekg(30); //move the get pointer to byte number 30 in the file
send.seekp(30);//move the put pointer to byte number 30 in the file
int posn = rcv.tellg();
int posn = send.tellp();
```

Functions for manipulation of file pointers

Another prototype

```
seekg ( offset, direction );
seekp ( offset, direction );
```

Function	Meaning
ios::beg	offset counted from the beginning of the stream
ios::cur	offset counted from the current position of the stream pointer
ios::end	offset counted from the end of the stream

write() and read() functions

- The functions **write()** and **read(),** different from the functions **put()** and **get(),** handle the data in binary form.

```
infile.read ((char * ) &V,sizeof(V));
outfile.write ((char *) &V ,sizeof(V));
```

- These functions take two arguments. The first is the address of the variable V, and the second is the length of that variable in bytes.
- The address of the variable must be cast to type **char***(i.e pointer to character type).

Reading & Writing class objects

```
class inventory
{
    char name[10];
    float cost;
    public:
    void readdata()
    {
        cout<<"Enter Name=";
        cin>>name;
        cout<<"Enter cost=";
        cin>>cost;
    }
    void displaydata()
    {
        cout<<"Name="<<name<<endl;
        cout<<"Cost="<<cost;
    }
};
```

Reading & Writing class objects

```
int main()
{
    inventory ob1;
    cout<<"Enter details of product\n";

    fstream file;
    file.open("stock.txt",ios::in | ios::app);

    ob1.readdata();
    file.write((char *)&ob1,sizeof(ob1));

    file.read((char *)&ob1,sizeof(ob1));

    ob1.displaydata();
    file.close();
}
```

Thank You