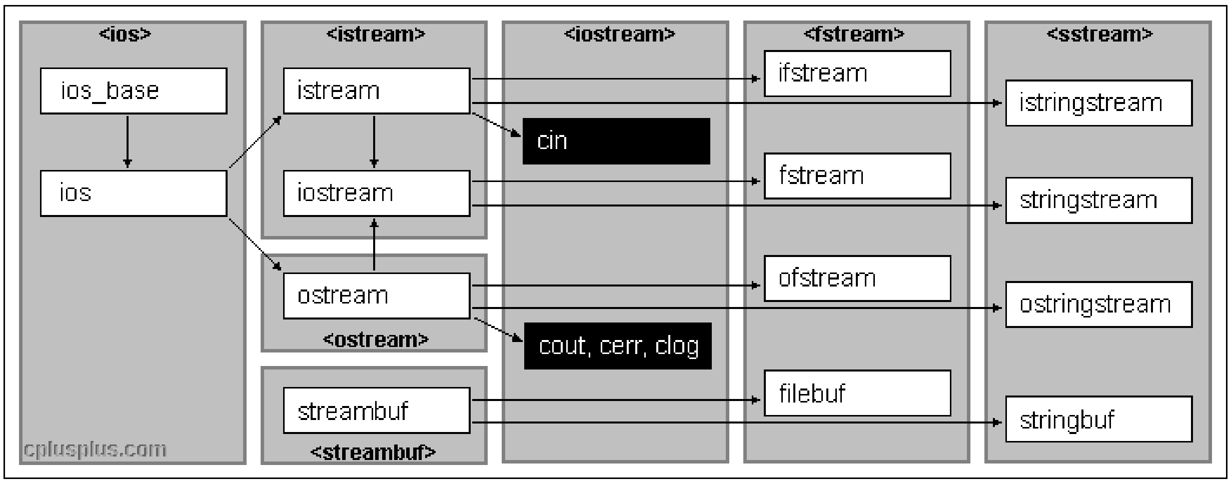
File Handling:

C++ provides the following classes to perform output and input of characters to/from files:

- ofstream: Stream class to write on files
- ifstream: Stream class to read from files
- fstream: Stream class to both read and write from/to files.

Input/Output library



Open a file:

The first operation generally performed on an object of one of these classes is to associate it to a real file. This procedure is known as to open a file.

#include <fstream>→ Add this header file

In order to open a file with a stream object we use its member function open:

open (filename, mode);

Where filename is a string representing the name of the file to be opened, and mode is an optional parameter with a combination of the following flags:

ios::in	Open for input operations.
ios::out	Open for output operations.
ios::binary	Open in binary mode.
ios::ate	Set the initial position at the end of the file. If this flag is not set, the initial position is the beginning of the file.
ios::app	All output operations are performed at the end of the file, appending the content to the current content of the file.
ios::trunc	If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one.

All these flags can be combined using the bitwise operator OR (|). For example, if we want to open the file example.bin in binary mode to add data we could do it by the following call to member function open:

```
ofstreammyfile;
myfile.open ("example.bin", ios::out | ios::app | ios::binary);
```

Each of the open member functions of classes ofstream, ifstream and fstream has a default mode that is used if the file is opened without a second argument:

class	**default mode parameter**
ofstream	ios::out
ifstream	ios::in
fstream	ios::in | ios::out

Important Point:

File streams opened in *binary mode* perform input and output operations independently of any format considerations. Non-binary files are known as *text files*, and some translations may occur due to formatting of some special characters (like newline and carriage return characters).

To check if a file stream was successful opening a file, you can do it by calling to member **is_open**. This member function returns a bool value of true in the case that indeed the stream object is associated with an open file, or false otherwise:

```
if (myfile.is_open()) { /* proceed if file is opened */ }
```

Closing a file:

When we are finished with our input and output operations on a file we shall close it so that the operating system is notified and its resources become available again. For that, we call the stream's member function close. This member function takes flushes the associated buffers and closes the file:

```
myfile.close();
```

Once this member function is called, the stream object can be re-used to open another file, and the file is available again to be opened by other processes.

In case that an object is destroyed while still associated with an open file, the destructor automatically calls the member function close.

Text files:

Text file streams are those where the **ios::binary** flag is not included in their opening mode. These files are designed to store text and thus all values that are input or output from/to them can suffer some formatting transformations, which do not necessarily correspond to their literal binary value.

Writing operations on text files are performed in the same way we operated with **cout**:

Examples:

Program to write to Files:

```
#include<fstream>
#include<iostream>
usingnamespace std;

int main()
{
        ofstreamofs;
        ofs.open("test.txt");
        ofs<<"This is a file";
ofs.close();
        cout<<endl;
        return 0;
}
```

File Output:

```
This is a file
```

Program to read from Files:

```
#include<fstream>
#include<iostream>
usingnamespace std;

int main()
{
        ifstream ifs;
        ifs.open("test.txt",ios::in);
        if (ifs.is_open())
// Runs only if file is opened
        {
                charch[20];
                ifs.getline(ch,20);
                cout<<ch;
        }
ifs.close();
        cout<<endl;
        return 0;
}
```

Console Output:

```
This is a file
```

Flags:

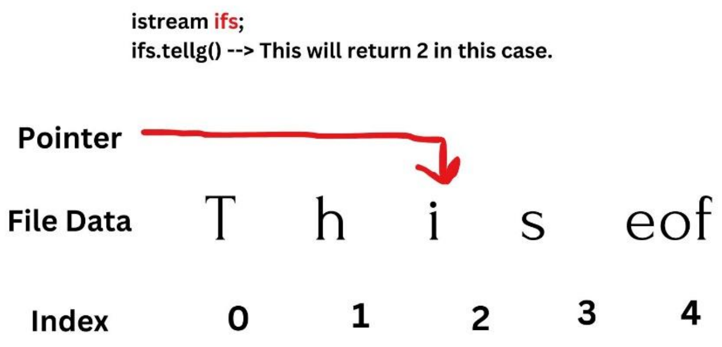
bad()	Returns true if a reading or writing operation fails. For example, in the case that we try to write to a file that is not open for writing or if the device where we try to write has no space left.
fail()	Returns true in the same cases as bad(), but also in the case that a format error happens, like when an alphabetical character is extracted when we are trying to read an integer number.
eof()	Returns true if a file open for reading has reached the end.
good()	It is the most generic state flag: it returns false in the same cases in which calling any of the previous functions would return true. Note that good and bad are not exact opposites (good checks more state flags at once).
The member function clear() can be used to reset the state flags.	

Get & Put:

File has a pointer to read and write data. The data will be read from and write to a file where the pointer is placed.

Get → Returns the index of the pointer in the file from where the data will be read.

Put→ Returns the index of the pointer in the file where the data will be written.



```
istream ifs;
ifs.tellg() --> This will return 2 in this case.
```

tellg() and tellp()

tellg()→ Returns get position.

tellp()→ Returns put position.

seekg() and seekp()

These functions allow to change the location of the get and put positions.

Both functions are overloaded with two different prototypes.

The first form is:

seekg (position);
seekp (position);

The Second form is:

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

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

ie:

ifs.seekg(ios::cur, 2) → This will move the get pointer from current location to forward by 2.
ie: if current pointer location is 3 then the get pointer would be at position 5 in the stream.

Binary files:

For binary files, reading and writing data with the extraction and insertion operators (<< and >>) and functions like getline is not efficient, since we do not need to format any data and data is likely not formatted in lines.

File streams include two member functions specifically designed to read and write binary data sequentially: write and read. The first one (write) is a member function of ostream (inherited by ofstream). And read is a member function of istream (inherited by ifstream). Objects of class fstream have both.

Their prototypes are:

```
istream& read (char* s, streamsize n);
ostream&write (char* s, streamsize n);
```