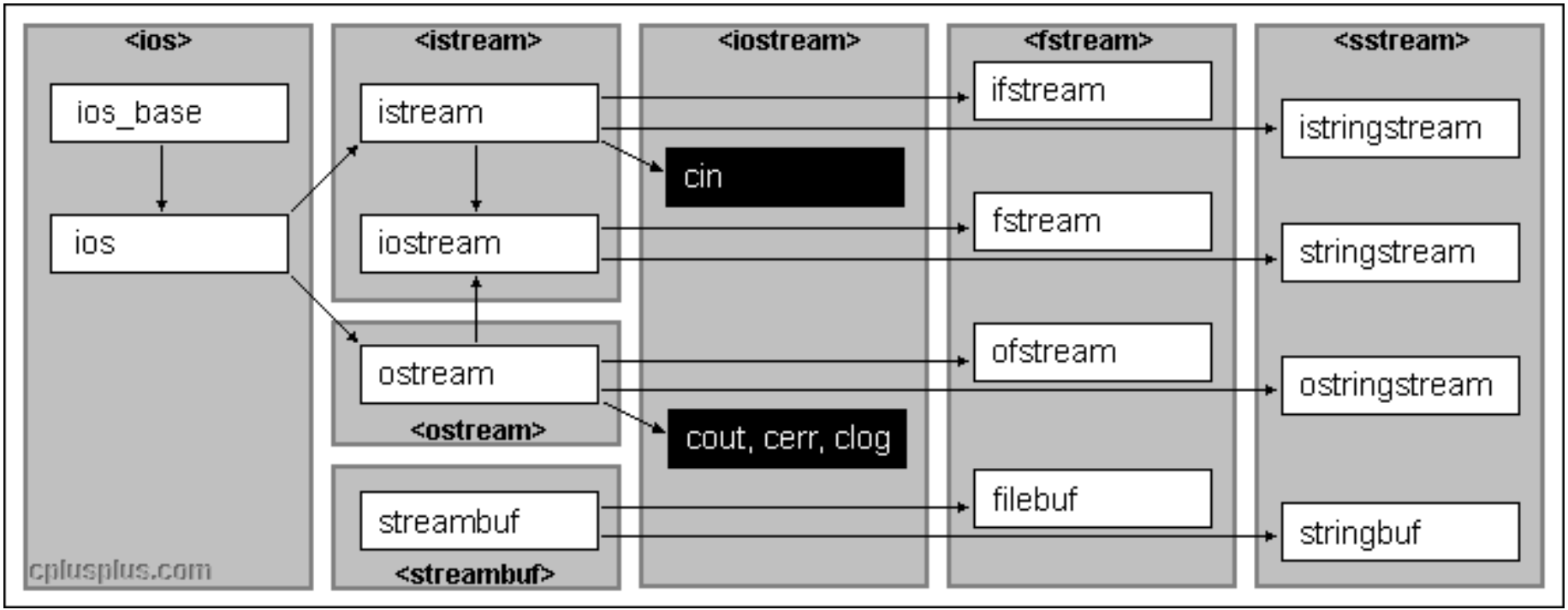
File handling

- Files are used to store data in a storage device permanently.
- A stream is an abstraction that represents a device on which operations of input and output are performed.
- Stands for the manipulation of files storing relevant data using a programming language
- Store the data in permanent storage that exists even after the program performs file handling
- C++ offers the library fstream for file-handling.


<ios>
ios_base
ios
<istream>
istream
iostream
<ostream>
ostream
<streambuf>
streambuf
<iostream>
cin
cout, cerr, clog
<fstream>
ifstream
fstream
ofstream
filebuf
<sstream>
istringstream
stringstream
ostringstream
stringbuf
cplusplus.com

The iostream library is an object-oriented library that provides input and output functionality using streams.

A stream is an abstraction that represents a device on which input and ouput operations are performed.

A stream can basically be represented as a source or destination of characters of indefinite length.

Streams are generally associated to a physical source or destination of characters, like a disk file, the keyboard, or the console, so the characters read or written to/from our abstraction called stream are physically input/output to the physical device.

The fstream library provides 3 different classes,

Ofstream:

- Used to writing data into the files present/absent in the system.
- It opens the file specified by the user using the class object or constructor.
- The various opening modes enable us to either write on an existing file or to create a new file and write on it.

Ifstream:

- used to read the data contents stored in a file present in the system.
- Whenever it tries to read a file that does not exist in the system an error is thrown to the user.

fstream:

- This class helps us to implement the above 2 classes through one class.
- We can read as well as write the data on the 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.

File handling takes place in 3 stages :

1. Opening the file - object contains 0 if failed to open else none zero value
2. Reading/Writing the file
3. Closing the file

Input stream serves as a path for data that is to be inputted into the main program.

This data is searched from the given file/s and then passed through the stream.

output stream serves as a path for any data that needs to be transferred from the main program to the file/s.

This data is provided by the user which is stored in some variables.

The program later passes this data through the output stream to the file/s where it is received and written on the file.

Some functions we need to know to read and write data to and from file as an i/p / o/p device

Read function

get()

getline()

read()

get()

reading one character at a time until the file is good.

The good() function returns true if the end of the file is not reached and there is no failure.

```
while (myfile.good()    ) {
        mychar = myfile.get();
         cout << mychar;
    }
```

```
while(in.get(ch))
  cout << ch;
```

get() returns the stream in, and in will be false when the end of the file is encountered.

getline()

store each line of file through the getline function in string.

```
#include <fstream>
#include<iostream>
using namespace std;
int main ()
{
     ifstream myfile("my_file.txt");
    if (myfile.is_open()) {
        char mychar;
        while (myfile.good()    ) {
            mychar = myfile.get();
             cout << mychar;
        }
    }
    return 0;
}
```

```
#include <fstream>
#include<iostream>
#include<string>
using namespace std;
int main ()
{
     ifstream myfile("fileHand.cpp");
    if (myfile.is_open()) {
         string myline;
        while (myfile.good()) {
             getline (myfile, myline);
             cout << myline <<  endl;
        }
    }
    return 0;
}
```

put()

The *put*() function writes ch to the stream and returns a reference to the stream.

eof()

```
bool eof() const;
```

Returns true if reached eof else false

```
// Display a file using get().
#include <iostream>
#include <fstream>
using namespace std;

int main()

{
  char ch;//   w   w  w .  d  e   mo   2 s   .   co    m

  ifstream in("sample.txt", ios::in | ios::binary);
  if(!in) {
     cout << "Cannot open file.\n";
     return 1;
  }

//   while(!in.eof()) { // in will be false when eof is reached
//      in.get(ch);
//      if(in) cout << ch;
//   }
  while(in.get(ch))
  cout << ch;

  in.close();

  return 0;
}
```

```
while(!in.eof()/*in*/) { // in will be false when eof is reached
    in.get(ch);
    if(in) cout << ch;
 }
```

File Position Pointers

There are two types of pointers get and put.

We can find the position of these pointers by associated functions tellg() and tellp().

Also, we can seek(change) the position of the pointer with the function seekg() and seekp().

There, we can do either read or write after seeking to a particular position.

Methods like seekg(), seekp() takes parameters as long integers and seek directions.

Const. Values od dir.

ios::beg (for positioning in the beginning of a stream)

ios::cur (for positioning relative to the current position of a stream)

ios::end (to position relative to the end of a stream)

Example

2 syntax

```
myFile.seekg(6, ios::beg);
myFile.seekg(6);
```

tellp() & tellg()

tellp() returns the current position of put pointer

used with output streams while writing the data to the file.

tellg() returns the current position of get pointer

used with input streams while receiving the data from the file.

How to Open Files

The files existing on the local system can be opened using 2 basic approaches :

1. Open the file using the **constructor function** of the fstream and related classes

2. **open() function** provided for this purpose.

Open function:

Performs the same task as the constructor while opening a file.

It takes two arguments one is the **filename** and second is the **mode of the opening** of the file.

Opening Modes:

The opening modes are the second parameter passed to file handling objects to determine the type of actions that would be taken on the file.

The user can also combine different opening modes using the ‘|’ (OR) operator. For example, fstream(filename, ios::out | ios::app)

Mode Flag	Description
ios::in	The files passed to the object are opened for the user to read its contents if it exists in the system.
ios::out	The files passed to the object are opened for the user to write data in it. If the file does not exist in the system then a new file is created.
ios::app	The files passed to the object are opened such that all the previous contents of the file remain unchanged and new data is added at the end of the file. If the file does not exist in the system then a new file is created.

ios::ate	The files passed to the object are opened to move the read/write control to the end of the file if the file exists on the system.
ios::trunc	If the file exists in the system then all its contents will be erased and new content will be written. If the file does not exist in the system then a new file will be created and new content will be written.

- The open() is a public member function of all these classes, its syntax is shown below.
- `void open (const char* filename, ios_base::openmode mode);`

- open() method takes two arguments one is the file name and the other is the mode in which the file going to open.
- The is_open() method is used to check whether the stream is associated with a file or not, it returns true if the stream is associated with some file otherwise returns false.
- `bool is_open();`

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
	fstream my_file;
	my_file.open("my_file.txt", ios::out);
	if (!my_file) {
		cout << "File not created!";
	}
	else {
		cout << "File created successfully!";
		my_file << "Hello Good morning";
		my_file.close();
	}
	return 0;
}
```

By Constructor Function

Open a file through constructor while creating stream object

ofstream streamObject(file name, mode);

```
ofstream streamObject("myFile.txt");
```

Open in append mode

```
#include <fstream>
#include<iostream>
#include<string>

int main ()
{
    std::ofstream myfile("sample.txt", std::ios_base::app);

    myfile << "\nThis content was appended in the File.";
    return 0;
}
```

Writing in Truncate Mode

```
#include <fstream>
#include<iostream>
#include<string>

int main ()
{
    std::ofstream myfile("sample.txt", std::ios_base::trunc);

    myfile << "Only this line will appear in the file.";
    return 0;
}
```

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
   fstream my_file("my_file.txt", ios::in);
   if (!my_file) {
       cout << "No such file";
   }
   else {
       char ch;
       while (1) {
           my_file >> ch;
           if (my_file.eof())
               Break;
           cout << ch;
       }
   }
   my_file.close();
   return 0;
}
```