File I/O

File

- A computer file
  - is stored on a secondary storage device (e.g., disk);
  - is permanent;
  - can be used to
    - provide input data to a program
    - or receive output data from a program
    - or both;

Stream

- C++ I/O system operates through streams.
- A stream is a logical device (a sequence of characters) that either produces or consumes information.
- A stream is linked to a physical device by the I/O system.
- All streams behave in the same way with no matter what kind of actual physical devices we can have.
- For example, we can use the same function that writes to a file to write to the printer or to the screen.

The C++ Stream Classes

- Standard C++ provides support for its I/O system in <iostream>.
- Low-level I/O class called basic_streambuf, it is basic class that offers low-level input and output operations, and provides the underlying support for the entire C++ I/O system.
- Class derived from **basic_ios** class a high-level I/O class that provides formatting, error checking, and status information related to stream I/O.
- **basic_ios** class is used as a base for several derived classes, including basic_istream, basic_ostream, and basic_iostream.
- These classes are used to create streams capable of input, output, and input/output, respectively.
- Classes used are 8-bit character I/O.

The C++ Stream Classes

Template Class	Character-based Class	Wide-Character-based Class
basic_streambuf	streambuf	wstreambuf
basic_ios	ios	wios
basic_istream	istream	wistream
basic_ostream	ostream	wostream
basic_iostream	iostream	wiostream
basic_fstream	fstream	wfstream
basic_ifstream	ifstream	wifstream
basic_ofstream	ofstream	wofstream

C++'s Predefined Streams

Stream	Meaning	Default Device
cin	Standard input	Keyboard
cout	Standard output	Screen
cerr	Standard error output	Screen
clog	Buffered version of cerr	Screen

Streams cin, cout, and cerr correspond to C's stdin, stdout, and stderr.

<fstream> and the File Classes

- For file I/O, we need to include the header <fstream> in our program.
- “fstream” defines many classes, including ifstream, ofstream, and fstream
- ifstream, ofstream, and fstream are derived from istream, ostream, and iostream
  - interactive (iostream)
    - **cin** - input stream associated with **keyboard.**
    - **cout** - output stream associated with **display**
  - file (fstream)
    - **ifstream** - defines new input stream (normally associated with a file).
    - **ofstream** - defines new output stream (normally associated with a file)

Opening and Closing a File

- In C++, to open a file, we need to link it to a stream.
- First obtain a stream, and then use it to open the file

- ifstream in; // input - object of ifstream class
- ofstream out; // output
- fstream io; // input and output

Open ()

- Opening a file associates a file stream variable declared in the program with a physical file at the source, such as a disk.

- In the case of an input file:
  - The file must exist before the open statement executes.
  - If the file does not exist, the open statement fails and the input stream enters the fail state.

- An output file does not have to exist before it is opened;
  - If the output file does not exist, the computer prepares an empty file for output.
  - If the designated output file already exists, by default, the old contents are erased when the file is opened.

Open () Cont..

- void ifstream::open(const char *filename, ios::openmode mode = ios::in);
- void ofstream::open(const char *filename, ios::openmode mode = ios::out | ios::trunc);
- void fstream::open(const char *filename, ios::openmode mode = ios::in | ios::out);
- “filename” is the name of the file including the path of the file.
- The value of **mode** determines how the file is opened.

Modes

Name	Description
ios::in	Open file to read
ios::out	Open file to write
ios::app	All the data you write, is put at the end of the file. It calls ios::out
ios::ate	All the data you write, is put at the end of the file. It does not call ios::out I/O operations can still occur anywhere within the file
ios::trunc	Deletes all previous content in the file (empties the file)
ios::binary	Opens the file in binary mode. By default, file gets opened in text mode.

- All defined in **ios** class through its base class **ios_base**.

- We can combine two or more of these values by ORing them together.

E.g. ios::ate | ios::binary

Various ways to Open file

- Using “open” function
  - ofstream out;
  - out.open("test", ios::out);
  - out.open("test"); // defaults to output and normal file

```
if(!mystream.is_open()) { bool is_open()
cout << "File is not open.\n";
// ...
```

- Using Constructors
- ifstream mystream("myfile"); // open file for input
- if(!mystream) {

```
        cout << "Cannot open file.\n";
        // handle error  }
```

- mystream.close(); // closing the file

General File I/O Steps

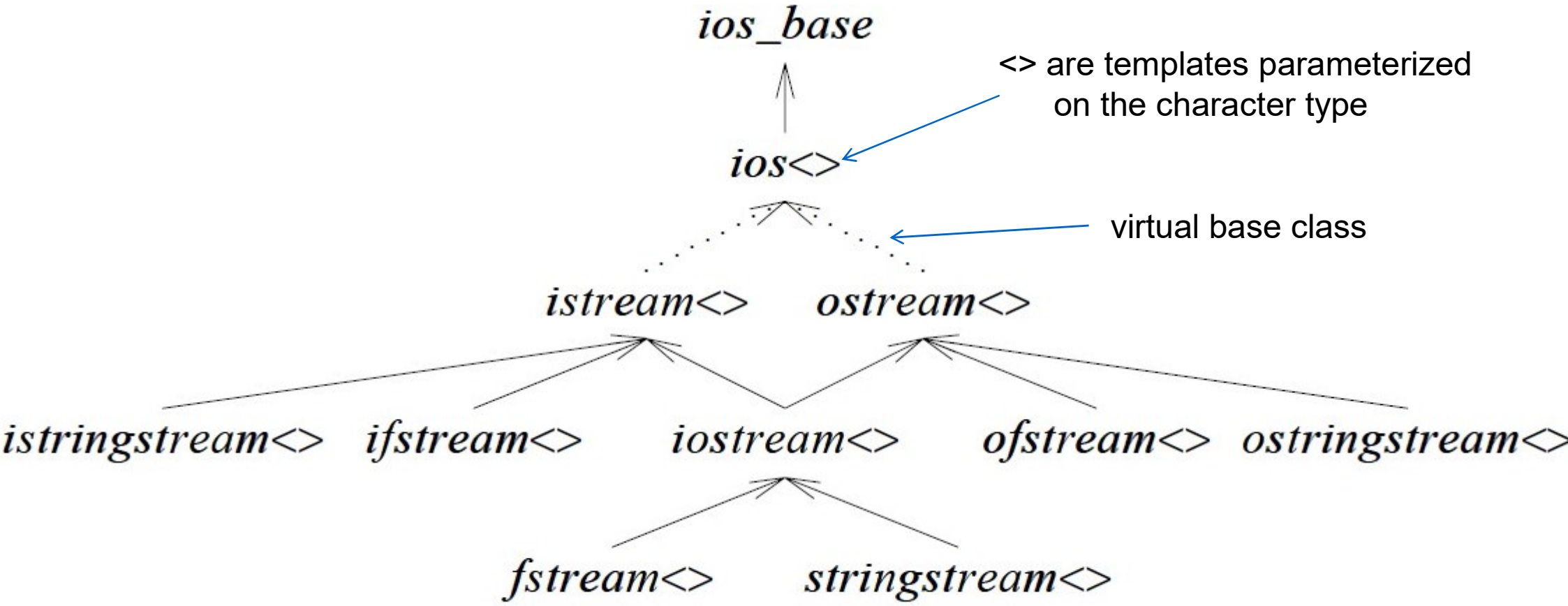
- Include the header file **fstream** in the program.
- Declare file stream variables.
- Associate the file stream variables with the input/output sources.
- Open the file
- Use the file stream variables with >>, <<, or other input/output functions.
- Close the file.

```
template <class Ch, class Tr = char_traits<Ch> >
class basic_ostream : virtual public basic_ios<Ch,Tr> {
public:
        // ...

        basic_ostream& operator<<(short n);
        basic_ostream& operator<<(int n);
        basic_ostream& operator<<(long n);

        basic_ostream& operator<<(unsigned short n);
        basic_ostream& operator<<(unsigned int n);
        basic_ostream& operator<<(unsigned long n);

        basic_ostream& operator<<(float f);
        basic_ostream& operator<<(double f);
        basic_ostream& operator<<(long double f);

        basic_ostream& operator<<(bool n);
        basic_ostream& operator<<(const void* p);

        basic_ostream& put(Ch c);          // write c
        basic_ostream& write(const Ch* p, streamsize n)

        // ...
;
```

n ***istream*** provides operator >> for the built-in types:

```
template <class Ch, class Tr = char_traits<Ch> >
class basic_istream : virtual public basic_ios<Ch, Tr> {
public:
        // ...
        // formatted input:

        basic_istream& operator>>(short& n);              // read into n
        basic_istream& operator>>(int& n);
        basic_istream& operator>>(long& n);

        basic_istream& operator>>(unsigned short& u);     // read into u
        basic_istream& operator>>(unsigned int& u);
        basic_istream& operator>>(unsigned long& u);

        basic_istream& operator>>(float& f);              // read into f
        basic_istream& operator>>(double& f);
        basic_istream& operator>>(long double& f);

        basic_istream& operator>>(bool& b);               // read into b
        basic_istream& operator>>(void* & p);             // read pointer value into

        // ...
};
```

Hierarchy of Standard Stream Classes



Unformatted and Binary I/O

- Efficient way to handle files is carried using unformatted (raw) binary data, not the text.
- To perform binary operations on a file, we need to open it using the ios::binary mode specifier.
- put( ) and get( ) functions operate on characters and will allow to write and read a character at a time.
- ofstream &put(char ch): writes a single character to the stream and returns a reference to the stream.
- ifstream &get(char &ch): reads a single character from the invoking stream and puts that value in ch and returns a reference to the stream.

put ( ) Example

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
int i;
ofstream out("CHARS", ios::out |
ios::binary);
if(!out) {
cout << "Cannot open output file.\n";
return 1;
}
// write all characters to disk
for(i=0; i<256; i++) out.put((char) i);
out.close();
return 0;
}
```

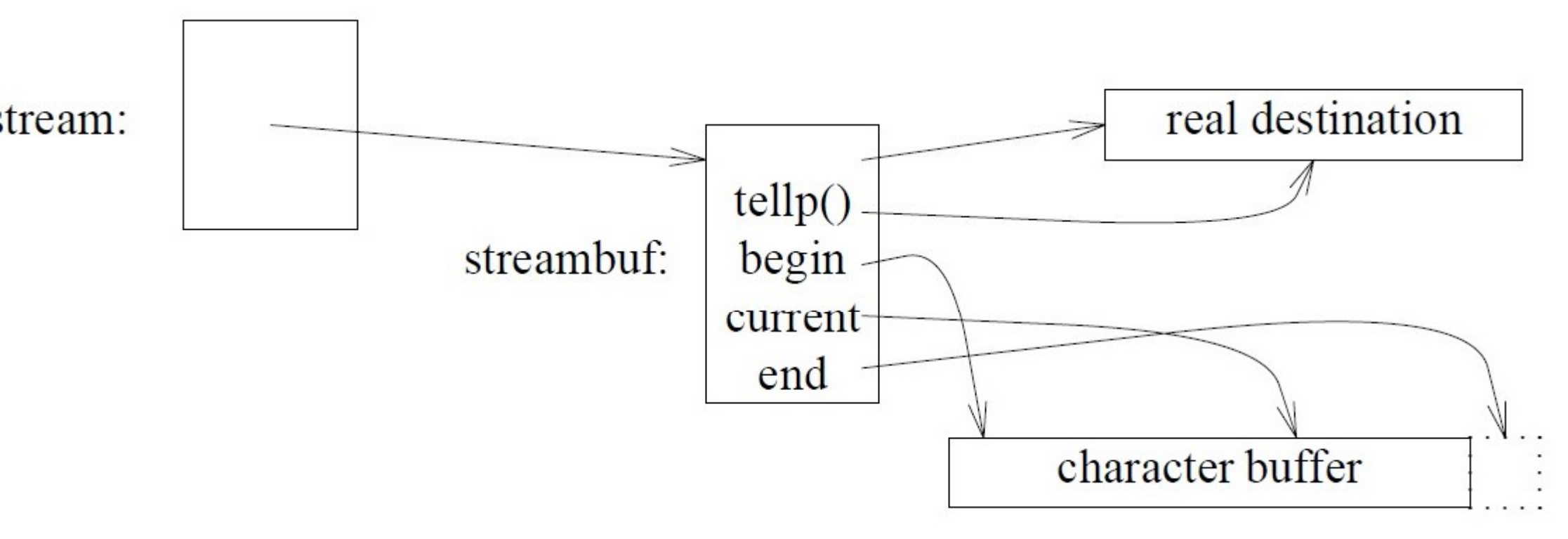
get ( ) Example

```
#include <iostream>
#include <fstream>
using namespace std;
int main(int argc, char *argv[])
{
char ch;
if(argc!=2) {
cout << "Usage: PR <filename>\n";
return 1;
}
ifstream in(argv[1], ios::in | ios::binary);
if(!in) {
cout << "Cannot open file.";
return 1;
}
while(in) { // in will be false when eof is reached
in.get(ch);
if(in) cout << ch;
}
return 0;
}
```

Buffering

- An output stream puts characters into a buffer
- The characters are written to wherever they are supposed to go after a while.
- This buffer is called a *streambuf.*
- *streambuf* stores characters in an array.
- When an *overflow* occurs, it writes the characters to the desired destination.

Buffering Cont..



read( ) and write( ) functions

- Block-oriented file i/o is done by using read( ) and write( ) functions.

- Prototypes:
- ofstream &write(const char *buf, streamsize num);
  - The write( ) function writes num characters to the invoking stream from the buffer pointed to by buf.

- ifstream &read(char *buf, streamsize num);
  - The read( ) function reads num characters from the invoking stream and puts them in the buffer pointed to by buf.

Random Access

- In C++'s I/O system, we can perform random access by using the seekg( ) and seekp( ) functions.

- Prototypes:
  - istream & seekg(off_type offset, seekdir origin);
  - ostream & seekp(off_type offset, seekdir origin);
  - where off_type is an integer type defined by ios that is capable of containing the largest valid value that offset can have.
  - seekdir is an enumeration defined by ios that determines how the seek will take place.

Random Access Cont..

- Two pointers associated with a file.
- The get pointer, which specifies where in the file the next input operation will occur.
- The put pointer, which specifies where in the file the next output operation will occur.
- Each time an input or output operation takes place, the appropriate pointer is automatically sequentially advanced.
- Using the seekg( ) and seekp( ) functions allows you to access the file in a nonsequential fashion.

Random Access Cont..

- The seekg( ) function moves the associated file's current get pointer offset number of characters from the specified origin, which must be one of these three values:

- ios::beg Beginning-of-file
- ios::cur Current location
- ios::end End-of-file

- The seekp( ) function moves the associated file's current put pointer offset number of characters from the specified origin, which must be one of the values shown above.

References

- C++: The Complete Reference, 4th Edition by Herbert Schildt , McGraw-Hill
- Teach Yourself C++ 3rd Edition by Herbert Schildt,
- The C+ + Programming Language, Third Edition by Bjarne Stroustrup, Addison Wesley