Unit-4

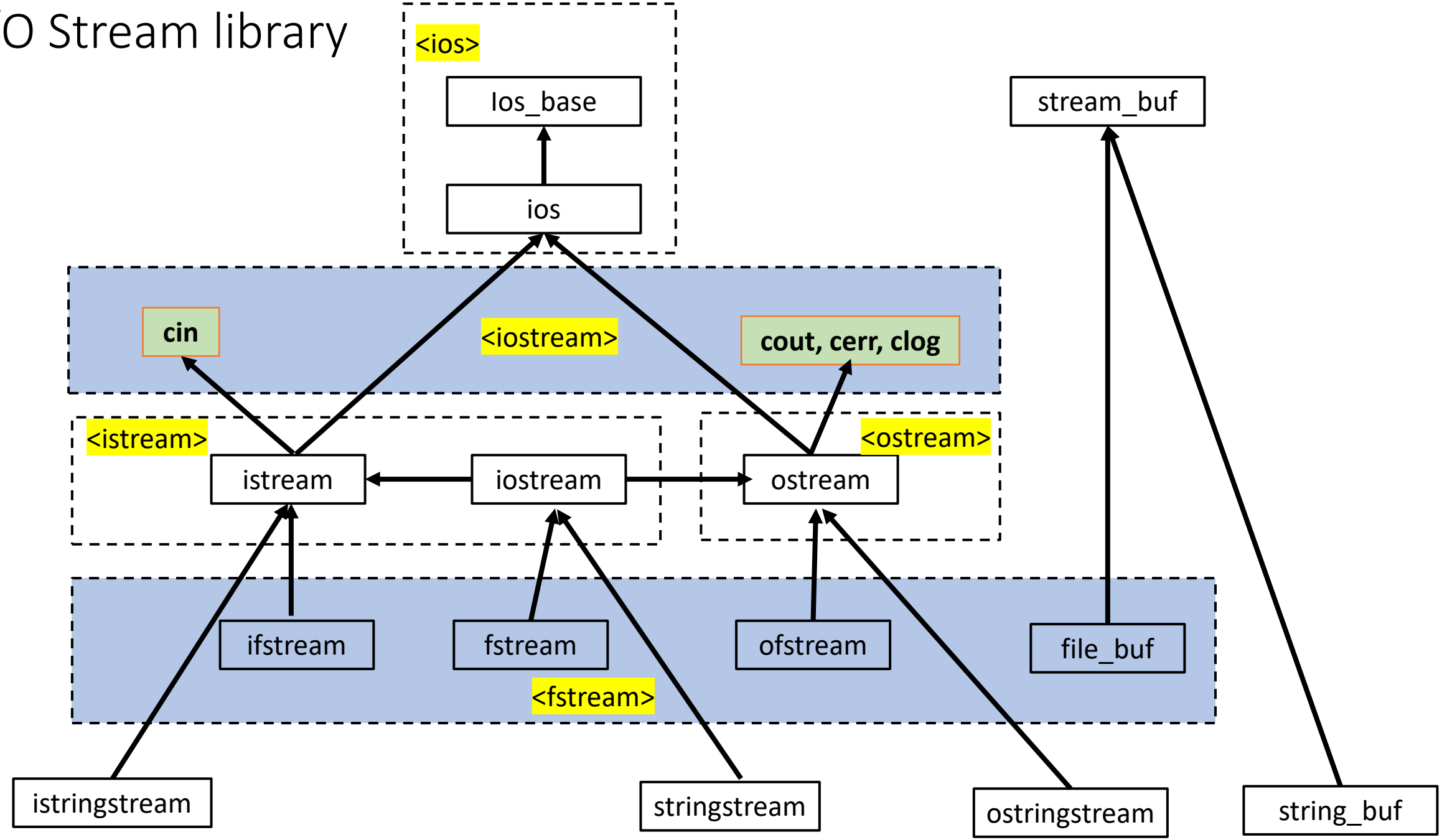
File Handling and Exception handling in C++

Unit-4 Content

Files: Introduction, File stream classes, Steps for file operations, Checking for errors, Finding end of file, File opening modes, File pointers and manipulators.

Exception Handling: Principles of Exception Handling, The Keywords try, throw and catch, Guidelines for Exception Handling, Multiple catch statements, Catching Multiple Exceptions, Re-Throwing Exceptions, Specifying Exceptions.

I/O Stream library



Steps for working with files

1. Create a file object using stream class
2. Open a file using open()
3. Read/write/append to/from a file
4. Close the file

```
ofstream yourfile;
```
Creates a file object

```
yourfile.open("new.txt");
```
Opens a file “new.txt in write mode using open()

```
ifstream myfile("test.txt");
```
Creates a file object and opens a file test.txt in read mode

```
ofstream yourfile("new.txt");
```
Creates a file object and opens a file new.txt in write mode

```
ofstream yourfile("new.txt‟ , ios::app);
```
Creates a file object and opens a file new.txt in append mode

flags	Description
ios::in	Opens an existing file in input mode
los::out	Opens a file for output
ios::app	Opens a file for output at the end-of-file
ios::ate	Open and seek to end immediately after opening.
los::trunc	An existing file is truncated to zero length
los::binary	Perform input and output in binary mode

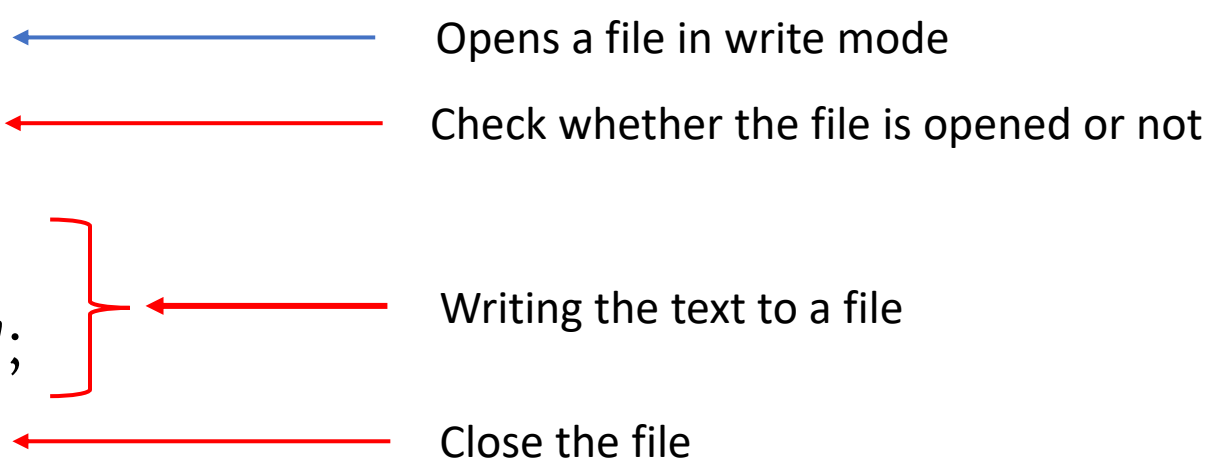
Default values in open() method	
`ifstream`	`ios::in`
`ofstream`	`ios::out | ios::trunc`
`fstream`	`ios::in | ios::out`

Reading from file

```
#include <iostream>
#include <fstream>
using namespace std;
int main () {
      string line;
      ifstream myfile ("example.txt");
      if (myfile.is_open())
      {
            while ( getline (myfile,line) ){
                  cout << line << '\n';
            }
            myfile.close();
      }
      else
            cout << "Unable to open file";
      return 0;
}
```

Writing to a file

```
#include <iostream> // for cout
#include <fstream> // for ofstream
using namespace std;
int main () {
  ofstream myfile ("example.txt");
  if (myfile.is_open())
  {
    myfile << "This is a line.\n";
    myfile << "This is another line.\n";
    myfile.close();
  }
  else cout << "Unable to open file";
  return 0;
}
```



Manipulators

Manipulators are functions specifically designed to be used in conjunction with the insertion (<<) and extraction (>>) operators on stream objects.

Set flag	Description	Unsets flag
boolalpha	Alphanumerical bool values	noboolalpha
showbase	Show numerical base prefixes	noshowbase
showpoint	Show decimal point	noshowpoint
showpos	Show positive signs	noshowpos
skipws	Skip whitespaces	noskipws
unitbuf	Flush buffer after insertions	nounitbuf
uppercase	Generate upper-case letters	nouppercase

- When the boolalpha format flag is set, bool values are inserted/extracted by their textual representation: either true or false, instead of integral values.

```
int main () {
  bool b = true;
  cout << boolalpha << b << '\n';    // prints true
  cout << noboolalpha << b << '\n'; // prints 1
  return 0;
}
```

Contd..

Manipulator	Description
Numerical base format flags	
dec	Use decimal base
hex	Use hexadecimal base
oct	Use octal base
Floating-point format flags	
fixed	Use fixed floating-point notation
scientific	Use scientific floating-point notation
Adjustment format flags	
internal	Adjust field by inserting characters at an internal position
left	Adjust output to the left
right	Adjust output to the right
Input Manipulator	
ws	Extract Whitespaces
Output Manipulator	
endl	Insert newline and flush
ends	Insert null character
flush	Flush stream buffer

Parameterized manipulators

Parameterized manipulators	Description
setiosflags	Set format flags
resetiosflags	Reset format flags
setbase	Set basefield flag
setfill	Set fill character
setprecision	Set decimal precision
setw	Set field width

These manipulator are declared in header <iomanip>.

```
// set/resetiosflags example
#include <iostream>     // cout, hex, endl
#include <iomanip>      // setiosflags, resetiosflags
using namespace std;
int main () {
        cout << hex << setiosflags (ios::showbase);
        cout << 100 << endl;                                    // print 0x64
        cout << resetiosflags(ios::showbase) ;          // unsets the showbase flag
        cout<< 100 << endl;
      return 0;
}
```

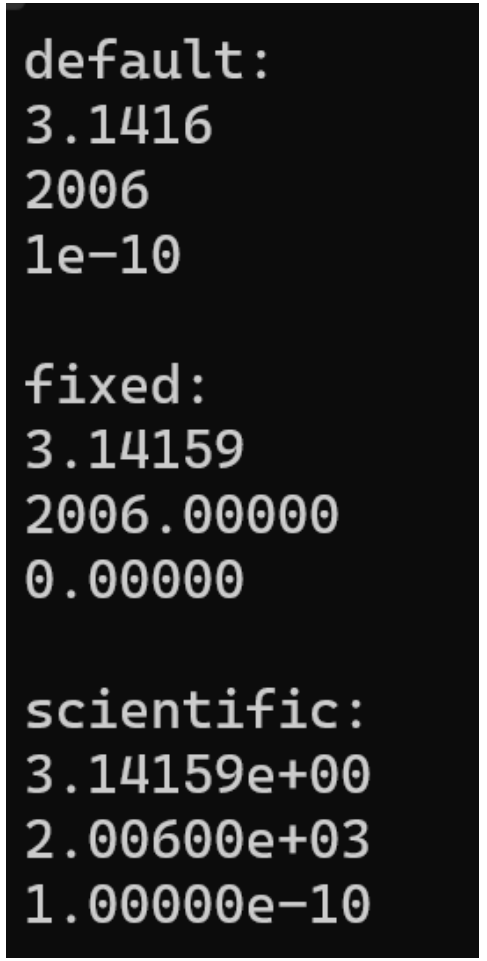
sets the hex flag

sets showbase flag using setiosflags() function

```
// modifying flags with setf/unsetf
#include <iostream>     // std::cout, std::ios
using namespace std;
int main () {
        cout.setf ( ios::hex, ios::basefield );  // set hex as the basefield
        cout.setf ( ios::showbase );              // activate showbase
        cout << 100 << '\n';
        cout.unsetf ( ios::showbase );            // deactivate showbase
        cout << 100 << '\n';
        return 0;
}
```

```
#include <iostream>     // std::cout, std::fixed
#include <iomanip>      // std::setprecision
using namespace std;
int main () {
  double f =3.14159;
  cout << setprecision(5) << f << '\n';
  cout << setprecision(9) << f << '\n';
  cout << fixed;
  cout << setprecision(5) << f << '\n';
  cout << setprecision(9) << f << '\n';
  return 0;
}
```

```
#include <iostream>     // std::cout, std::fixed, std::scientific
using namespace std;
int main () {
        double a = 3.1415926534;
        double b = 2006.0;
        double c = 1.0e-10;
        cout.precision(5);
        cout << "default:\n";
        cout << a << '\n' << b << '\n' << c << '\n';
        cout << '\n';
        cout << "fixed:\n" << fixed;
        cout << a << '\n' << b << '\n' << c << '\n';
        cout << '\n';
        cout << "scientific:\n" << scientific;
        cout << a << '\n' << b << '\n' << c << '\n';
        return 0;
}
```



On the default floating-point notation, the *precision field* specifies the maximum number of meaningful digits to display both before and after the decimal point, while in both the fixed and scientific notations, the *precision field* specifies exactly how many digits to display *after* the decimal point, even if they are trailing decimal zeros.

```
#include <iostream>     // std::cout, std::ios
using namespace std;
int main () {
        double f = 3.14159;
        cout.unsetf ( ios::floatfield );          // floatfield not set
        cout.precision(5);
        cout << f << '\n';
        cout.precision(10);
        cout << f << '\n';
        cout.setf( ios::fixed, ios::floatfield ); // floatfield set to fixed
        cout << f << '\n';
        return 0;
}
```

```
3.1416
3.14159
3.1415900000
```