

C++ I/O

Chapter 13



What is covered?

- What is a stream?
- Pre-defined streams in C++
- Formatting the program output using pre-defined stream classes
- Understanding use of manipulators in formatting the output
- Performing File I/O operations
- Files operating on built-in and user-defined data types
- Handling exceptions during I/O operations
- Character versus binary mode of file operations
- Random Access Files



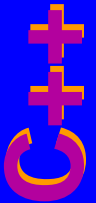
Streams

- A stream is a logical device that either produces or consumes the data
- It is a medium through which a read/write operation is performed. The medium may be a source of data or a consumer of data
 - *Input stream*
 - *Output stream*

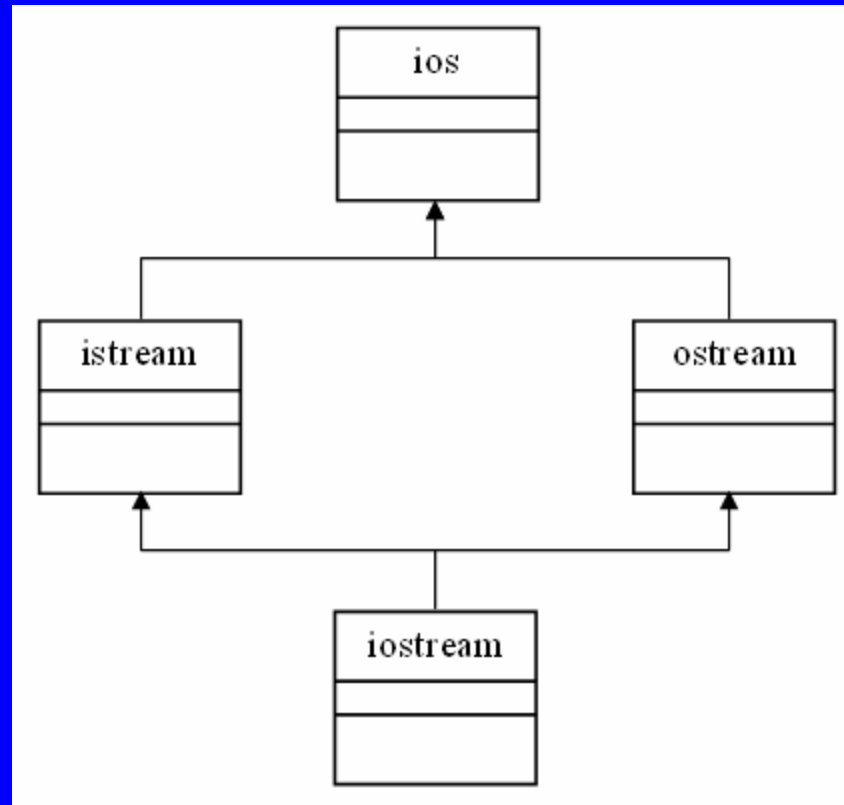


Input stream

- Input stream helps in reading data from a medium and the output stream helps in writing data to a medium
- The entire new I/O system of C++ is based on a template hierarchy. At the root, we have ios class
 - **istream** – Input stream
 - **ostream** – Output stream
 - **iostream** – Input/Output stream



Class Hierarchy





Predefined Streams

- The C++ runtime provides four predefined streams:
 - cin – Standard Input
 - cout – Standard Output
 - cerr – Standard Error Output
 - clog – Buffered Version of cerr



Formatting Output

- Let us consider that you wish to print a following invoice on the printer or on the user console:

```
P(IV) Computer      +10000.00
Less Discount       -500.00
Tax                  +100.00
-----
Total:              +9600.00
```



Program Example

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int main()
{
    cout.setf (ios::fixed);
    cout.precision(2);
    cout.setf (ios::showpos);
    cout.setf (ios::left);
    cout.width (30);
    cout << "P(IV) Computer";
```



Program Example - Continued

```
cout.setf (ios::right);  
cout.width(10);  
cout << 10000.00 << endl;  
cout.unsetf(ios::right);  
cout.setf (ios::left);  
cout.width (30);  
cout << "Less Discount";  
cout.setf (ios::right);  
cout.width(10);  
cout << -500.00 << endl;
```



Program Example - Continued

```
cout.unsetf(ios::right);  
cout.setf (ios::left);  
cout.width (30);  
cout << "Tax";  
cout.setf (ios::right);  
cout.width(10);  
cout << 100.00 << endl;
```

```
cout.unsetf(ios::right);  
cout.setf (ios::left);  
cout.width (40);  
char Separator[41];
```



Program Example - Continued

```
for (int i=0; i< 40; i++)  
    Separator [i] = '-';  
Separator [40] = '\0';  
cout << Separator << endl;
```

```
cout.unsetf(ios::right);  
cout.setf (ios::left);  
cout.width (30);  
cout << "Total: ";  
cout.setf (ios::right);  
cout.width(10);  
cout << 9600.00 << endl;  
return 0;  
}
```



Program Output

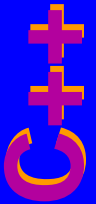
P(IV) Computer	+10000.00
Less Discount	-500.00
Tax	+100.00

Total:	+9600.00



The ios Class Formatting flags

- The ios class provides several formatting flags such as
 - fixed
 - showpos
 - left
 - right



The boolalpha flag

This flag is used to convert a `bool` variable to strings such as `true` or `false`. We set this flag on a given stream using the `setf` method

Example:

```
stream.setf (ios::boolalpha);
```



The boolalpha flag - Program Example

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int main()
{
    bool b;
    cout.setf (ios::boolalpha);
    cin.setf (ios::boolalpha);
    cout << "Enter a boolean: ";
    cin >> b;
    cout << "You Entered " << b << endl;
    return 0;
}
```



Program Output

```
Enter a boolean: true
```

```
You Entered true
```

```
Enter a boolean: false
```

```
You Entered false
```



The scientific flag

Once set, this flag causes the floating-point numbers to be printed in exponential format. The following code snippet illustrates the use of this flag

Example:


```
cout.setf (ios::scientific);  
cout << "Scientific format: " << 100.00 << endl;
```



Output

The output produced by the above code would be as follows:

```
Scientific format: 1.000000e+002
```



Manipulators

- The manipulators are special functions that are included in an I/O expression to format the data
- C++ defines several manipulators that provide functionality equivalent to the formatting functions of ios class



The endl manipulator

- The endl manipulator causes a newline character ('\n') to be output. It also flushes the output stream

Example:

```
cout << "C++ endl manipulator" << endl;
```

The hex/oct/dec/showbase manipulators

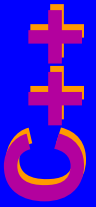
- The `showbase` manipulator prefixes the given number with appropriate characters while displaying the number in the respective base formats

The hex/oct/dec/showbase manipulators

Example:

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int main()
{
    int i = 256;
    cout << "Octal: " << showbase << oct << i << endl;
    cout << "Hex: " << showbase << hex << i << endl;
    cout << "Decimal: " << showbase << dec << i << endl;

    return 0;
}
```



Program Output

Octal: 0400

Hex: 0x100

Decimal: 256



The uppercase Manipulator

The uppercase manipulator prints the leading format characters for hexadecimal numbers in upper case

Example:

```
cout << "Hex: " << showbase << uppercase << hex  
<< i << endl;
```

Output:

```
Hex: 0X100
```



Floating Point Number Manipulators


- The following program statement illustrates how to format a floating-point number while displaying it on the console

```
cout << setfill('*') << setw(10) << fixed <<  
setprecision(2) << 10000.00 << endl;
```



File I/O

- C++ provides several classes for creating user-defined streams
- Three types of streams,
 - input stream
 - output stream and
 - input/output stream

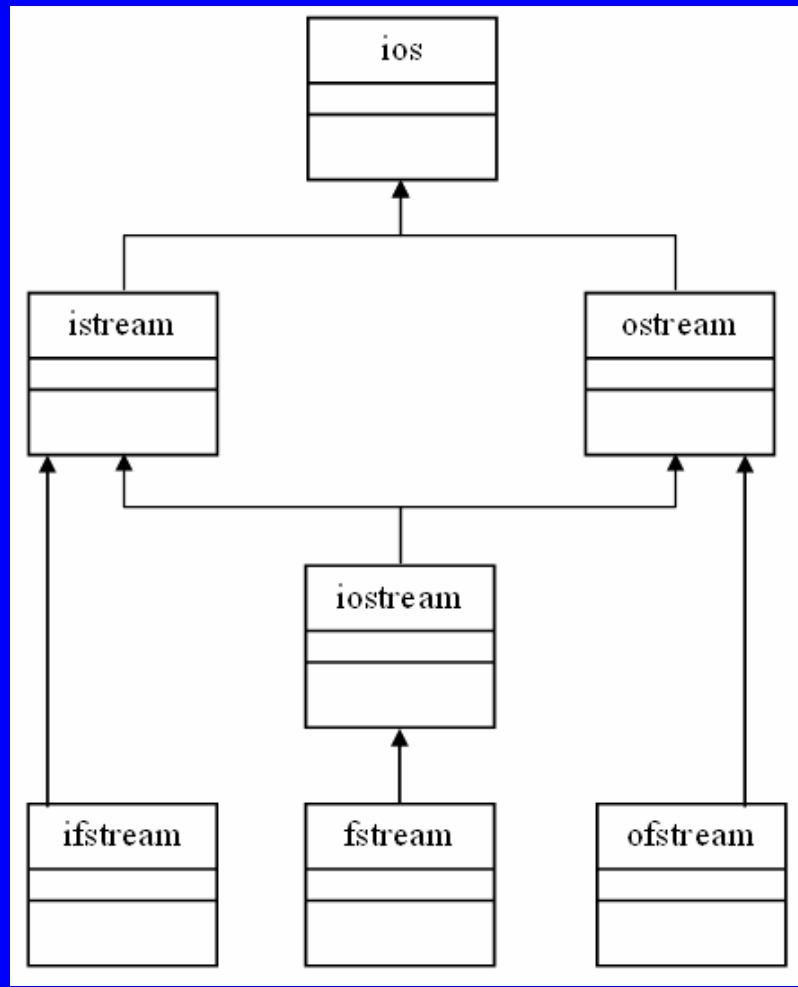


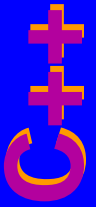
File I/O Classes

- **ifstream**
 - derives from the istream class and is used for reading data from a file
- **ofstream**
 - derives from ostream class and is used for writing data to a file
- **fstream**
 - derives from iostream class and is used to perform both read and write operations on a file



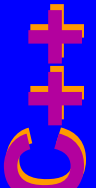
Class Hierarchy





Opening File

- Two ways to open the file:
 - Pass the filename to the stream constructor
 - Call the open member function of the stream class



Using Constructor

```
ifstream infile ("autoexec.bat");
```

```
ofstream outfile ("MyFile.dat");
```

Note:

- The above method call destroys the original file contents if one such exists
- if the file does not exist it creates a new file using the specified name



Using open Method

- For Reading:

```
ifstream is;  
is.open ("autoexec.bat");
```

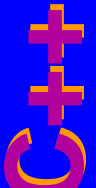
- For writing

```
ofstream os;  
os.open ("MyFile.dat");
```



Reading and writing files

- To read data from the file, we use the overloaded `<<` operator
- To write data to a file, we use the overloaded `>>` operator



Program Example

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
using namespace std;
void main()
{
    cout << "Opening/Creating MyFile.txt for writing" << endl;
    ofstream writer("MyFile.txt");
    cout << "Writing three data items 12, 18, 24 ..." << endl;
    writer<<12<< " " << 18 <<" " <<24;
    cout << "Closing output file" << endl;
```



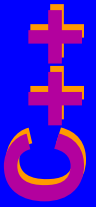
Program Example - Continued

```
writer.close();  
cout << endl;  
cout << "Opening MyFile.txt for reading" << endl;  
ifstream reader("MyFile.txt");  
int num1, num2, num3;  
cout << "Reading three data items ..." << endl;  
reader>>num1;  
reader>>num2;  
reader>>num3;
```



Program Example - Continued

```
cout<< "Data Item 1: " <<num1<<endl;
cout<< "Data Item 2: " <<num2<<endl;
cout<< "Data Item 3: " <<num3<<endl;
cout << "Closing file" << endl;
reader.close();
}
```



Output

```
Opening/Creating MyFile.txt for writing
Writing three data items 12, 18, 24 ...
Closing output file
```

```
Opening MyFile.txt for reading
Reading three data items ...
Data Item 1: 12
Data Item 2: 18
Data Item 3: 24
Closing file
```



Handling Errors

```
ofstream writer("MyFile.txt");  
if (!writer)  
    cout << "Could not open file for writing" << endl;
```

- Likewise, when you open the file for reading, check for the null value of the returned stream object as shown below:

```
if (!reader)  
    cout << "Could not open file for reading" << endl;
```



Reading/Writing Strings

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
using namespace std;
void main()
{
    cout << "Opening/Creating Myfile.txt for writing"
    << endl;
    ofstream writer("MyFile.txt");
```



Program Example - Continued

```
if (!writer)
    cout << "Could not open file for writing" << endl;
cout << "Writing three test strings ..." << endl;
writer<<"Test String 1" << endl;
writer<<"Test String 2" << endl;
writer<<"Test String 3" << endl;
cout << "Closing file" << endl;
writer.close();
cout << endl;
```



Program Example - Continued

```
cout << "Opening Myfile.txt for reading" << endl;
ifstream reader("MyFile.txt");
if (!reader)
    cout << "Could not open file for reading" <<
    endl;
char str[50];
cout << "Reading three strings ..." << endl;
```



Program Example - Continued

```
for (int i=0; i<3; i++)
{
    reader.getline(str, 50);
    cout<< str << endl;
}
cout << "Closing file" << endl;
reader.close();
}
```



Output

```
Opening/Creating Myfile.txt for writing
```

```
Writing three test strings ...
```

```
Closing file
```

```
Opening Myfile.txt for reading
```

```
Reading three strings ...
```

```
Test String 1
```

```
Test String 2
```

```
Test String 3
```

```
Closing file
```



Reading/Writing User-defined Types

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
using namespace std;
class Employee
{
    int EmpID;
    float Salary;
public:
    Employee (int id, float salary)
    {
        EmpID = id;
        Salary = salary;
    }
}
```



Program Example - Continued

```
Employee ()
{
    EmpID = 0;
    Salary = 0;
}
void Dump ()
{
    cout << "Employee ID = " << EmpID << " " << " Salary = " << Salary
    << endl;
}
};
int main(int argc, char* argv[])
{
    cout << "Creating three employee objects ..." << endl;
    Employee emp1 (1, 10000);
    Employee emp2 (2, 20000);
    Employee emp3 (3, 8000);
```



Program Example - Continued

```
cout << "Creating/Opening payroll.dat file for writing" <<
endl;
ofstream os ("Payroll.dat", ios::out | ios::binary);
if (!os)
{
    cout << "Could not open output file" << endl;
    exit(0);
}
cout << "Writing three employee objects ..." << endl;
os.write((char *)&emp1, sizeof(Employee));
os.write((char *)&emp2, sizeof(Employee));
os.write((char *)&emp3, sizeof(Employee));
cout << "Closing file" << endl;
```



Program Example - Continued

```
os.close();
cout << endl;
cout << "Opening payroll.dat file for reading" <<
endl;
ifstream is ("Payroll.dat", ios::in | ios::binary);
if (!is)
{
    cout << "Could not open input file" << endl;
    exit(0);
}
```



Program Example - Continued

```
Employee emp;
cout << "Reading three employee objects ..." << endl;
for (int i=0; i<3; i++)
{
    is.read((char *)&emp, sizeof (Employee));
    emp.Dump();
}
cout << "Closing file" << endl;
is.close();
return 0;
}
```



Output

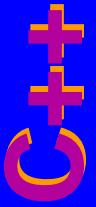
```
Creating three employee objects ...
Creating/Opening payroll.dat file for writing
Writing three employee objects ...
Closing file
```

```
Opening payroll.dat file for reading
Reading three employee objects ...
Employee ID = 1   Salary = 10000
Employee ID = 2   Salary = 20000
Employee ID = 3   Salary = 8000
Closing file
```



Character v/s Binary Mode

- Use binary mode for storing/retrieving objects of user-defined data types
- Access the file in character mode if we are reading/writing strings



The type/cat Utility

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
using namespace std;
int main(int argc, char* argv[])
{
    if (argc != 2)
    {
        cout << "Usage:type filename" << endl;
        exit(0);
    }
}
```



Program Example

```
ifstream is (argv[1]);
if (!is) {
    cout << "Could not open input file" << endl;
    exit(0);
}
char ch[256];
while (!is.eof())
{
    is.getline(ch, 255);
    cout << ch << endl;
}
cout << endl;
return 0;
}
```



File Dump Utility

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;
int main(int argc, char* argv[])
{
    if (argc != 2)
    {
        cout << "Usage: cat filename" << endl;
        exit(0);
    }
}
```



Program Example - Continued

```
ifstream in (argv[1], ios::in | ios::binary);
if (!in) {
    cout << "Could not open input file" << endl;
    exit(0);
}
cout.setf (ios::uppercase);
cout.fill ('0');
char ch;
char count = 0;
```



Program Example - Continued

```
while (!in.eof())
{
    in.get(ch);
    cout << setw(2) << hex << (int) ch << " ";
    count++;
    if (count == 16)
    {
        cout << endl;
        count = 0;
    }
}
cout << endl;
return 0;
}
```



Output

```

75 74 2E 66 69 6C 6C 20 28 27 30 27 29 3B 0D 0A
0D 0A 09 63 68 61 72 20 63 5B 31 36 5D 3B 0D 0A
09 63 68 61 72 20 63 68 3B 0D 0A 09 63 68 61 72
20 63 6F 75 6E 74 20 3D 20 30 3B 0D 0A 09 77 68
69 6C 65 20 28 21 69 6E 2E 65 6F 66 28 29 29 20
0D 0A 09 7B 0D 0A 09 09 69 6E 2E 67 65 74 28 63
68 29 3B 0D 0A 09 09 63 6F 75 74 20 3C 3C 20 73
65 74 77 28 32 29 20 3C 3C 20 68 65 78 20 3C 3C
20 28 69 6E 74 29 20 63 68 20 3C 3C 20 22 20 22
3B 0D 0A 09 09 63 6F 75 6E 74 2B 2B 3B 0D 0A 09
09 69 66 20 28 63 6F 75 6E 74 20 3D 3D 20 31 36
29 0D 0A 09 09 7B 0D 0A 09 09 09 63 6F 75 74 20
3C 3C 20 65 6E 64 6C 3B 0D 0A 09 09 09 63 6F 75
6E 74 20 3D 20 30 3B 0D 0A 09 09 7D 0D 0A 09 7D
0D 0A 09 63 6F 75 74 20 3C 3C 20 65 6E 64 6C 3B
0D 0A 09 72 65 74 75 72 6E 20 30 3B 0D 0A 7D 0D
0A 0A

```



File Copy Utility

```
// cp.cpp - copies source file to destination
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;
int main(int argc, char* argv[])
{
    if (argc != 3)
    {
        cout << "Usage: copy source destination" << endl;
        exit(0);
    }
}
```



Program Example - Continued

```
ifstream in (argv[1], ios::in | ios::binary);
if (!in) {
    cout << "Could not open input file" << endl;
    exit(0);
}
ofstream out (argv[2], ios::out | ios::binary);
if (!out) {
    cout << "Could not open output file for writing" <<
endl;
    exit(0);
}
```



Program Example - Continued

```
char ch;
while (true)
{
    in.get(ch);
    if (in.eof())
        break;
    out.put (ch);
}
return 0;
}
```



Run The Program

```
C:\>cp sourcefile destinationfile
```



Random Access Files

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
using namespace std;
class Employee
{
    int EmpID;
    float Salary;
public:
    Employee (int id, float salary)
    {
        EmpID = id;
        Salary = salary;
    }
}
```



Program Example - Continued

```
Employee ()
{
    EmpID = 0;
    Salary = 0;
}
void SetSalary (float salary)
{
    Salary = salary;
}
void Dump ()
{
    cout << "Employee ID: " << EmpID << " " << " Salary: " <<
        Salary << endl;
}
};
```



Program Example - Continued

```
int main(int argc, char* argv[])
{
    Employee emp1 (1, 10000);
    Employee emp2 (2, 20000);
    Employee emp3 (3, 8000);
    cout << "Creating payroll data" << endl;
    ofstream os ("Payroll.dat", ios::out | ios::binary);
    if (!os)
    {
        cout << "Could not open output file" << endl;
        exit(0);
    }
}
```



Program Example - Continued

```
os.write((char *)&emp1, sizeof(Employee));
os.write((char *)&emp2, sizeof(Employee));
os.write((char *)&emp3, sizeof(Employee));
os.close();
fstream stream ("Payroll.dat", ios::in | ios::out |
ios::binary);
if (!stream)
{
    cout << "Could not open input file" << endl;
    exit(0);
}
```



Program Example - Continued

```
cout << "The current data" << endl;
Employee emp;
for (int i=0; i<3; i++)
{
    stream.read((char *)&emp, sizeof (Employee));
    emp.Dump();
}
cout << endl;
cout << "Retrieving second record" << endl;
stream.seekg(1*sizeof (Employee), ios::beg);
stream.read((char *)&emp, sizeof (Employee));
```



Program Example - Continued

```
cout << "Modifying salary for second employee" << endl << endl;
emp.SetSalary(12000);
stream.seekg(1*sizeof (Employee), ios::beg);
stream.write((char *)&emp, sizeof(Employee));
cout << "Modified data" << endl;
stream.seekg (0, ios::beg);
for (i=0; i<3; i++)
{
    stream.read((char *)&emp, sizeof (Employee));
    emp.Dump();
}
stream.close();
return 0;
}
```



Program Output

Creating payroll data

The current data

Employee ID: 1 Salary: 10000

Employee ID: 2 Salary: 20000

Employee ID: 3 Salary: 8000

Retrieving second record

Modifying salary for second employee

Modified data

Employee ID: 1 Salary: 10000

Employee ID: 2 Salary: 12000

Employee ID: 3 Salary: 8000



Formatting Output (seekp function)

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;
int main(int argc, char* argv[])
{
    char str[50];
    ofstream os ("TextFile.txt", ios::binary);
    for (int i=0; i< 100; i++)
```



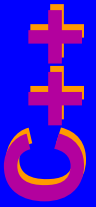
Program Example - Continued

```
{
    os << setw(2) << setfill('0') << i << " ";
}
os.close();
ifstream is ("TextFile.txt", ios::binary);
while (!is.eof())
    cout << (char)is.get();
is.close();
cout << endl << endl;
fstream stream ("TextFile.txt",
    ios::in | ios::out | ios::binary);
```



Program Example - Continued

```
for (i=1; i< 10; i++)
{
    stream.seekp(30*i-1, ios::beg);
    stream.put('\n');
}
stream.close();
cout << "dumping new file contents" << endl;
ifstream isf ("TextFile.txt");
while (!isf.eof())
    cout << (char) isf.get();
isf.close();
cout << endl << endl;
}
```



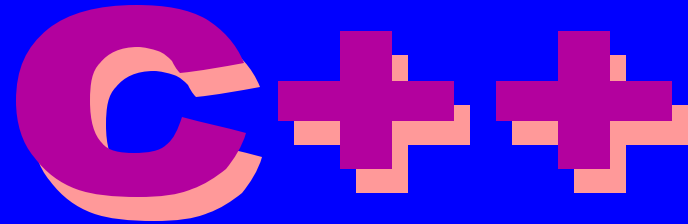
Program Output

```
00 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
```



Summary

- **The file I/O operations allow you to read/write data**
- **A stream is a logical device that either produces or consumes the data**
- **C++ libraries define standard streams such as cin, cout, cerr and clog**
- **A stream may of input or output type**
- **The data that is output may be formatted using**
 - formatting flags provided in the ios class
 - manipulators
- **A file may be opened for reading/writing in either the character mode or binary mode**
 - The character mode of file operation is used if you are dealing with files containing character strings
 - Binary mode is used when you want to deal with the file data at byte level
- **A file may be read sequentially or at random**



Conclusion